Scilab Textbook Companion for Process Dynamics And Controls by D. E. Seborg, T. F. Edgar, D. A. Mellichamp And F. J. Doyle III[1]

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

May 24, 2016

[1]Funded by a grant from the National Mission on Education through ICT, http://spoken-tutorial.org/NMEICT-Intro. This Textbook Companion and Scilab codes written in it can be downloaded from the "Textbook Companion Project" section at the website http://scilab.in

Book Description

Title: 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

List of Figures

Chapter 2

Theoretical Models of Chemical Processes

Scilab code Exa 2.1 Stirred tank blending process

```
clear
clc

//Example 2.1
disp('Example 2.1')

w1bar=500;
w2bar=200;
x1bar=0.4;
x2bar=0.75;
wbar=w1bar+w2bar;
t=0:0.1:25; //Time scale for plotting of graphs

//(a)
xbar=(w1bar*x1bar+w2bar*x2bar)/wbar;
printf('\n (a) The steady state concentration is %f
   \n',xbar)
```

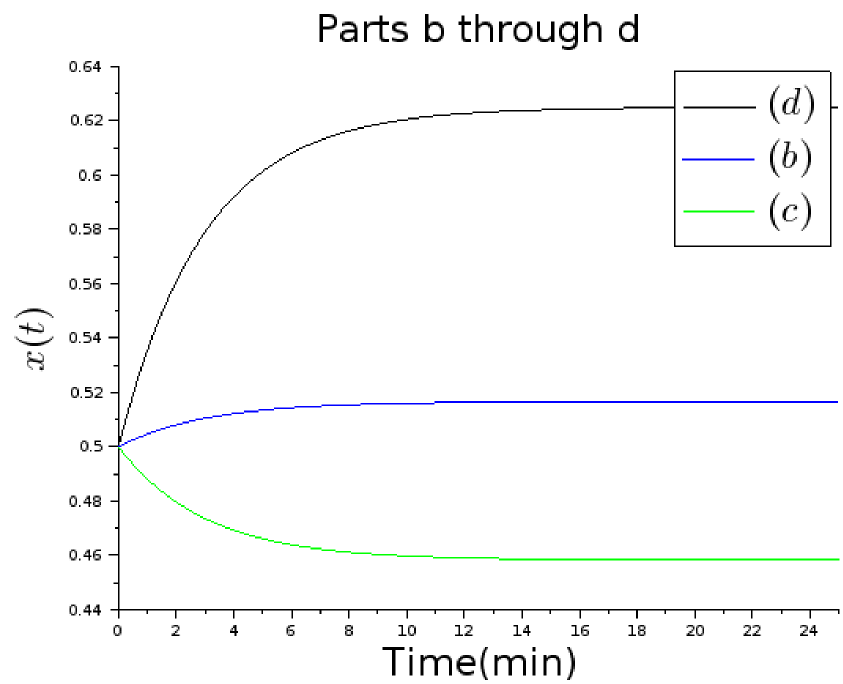


Figure 2.1: Stirred tank blending process

```

//(b)
w1bar=400; //flow rate changes, rest remains same
wbar=w1bar+w2bar;
tau=3;
x0=0.5;
Cstarb=(w1bar*x1bar+w2bar*x2bar)/wbar;  //C* variable
printf('\n (b) The value of C* is %f',Cstarb)
printf('\n x(t)=0.5exp(-t/3)+%f(1-exp(-t/3)) \n',Cstarb);
xtd=0.5*exp(-t/3)+Cstarb*(1-exp(-t/3));

xtb=0.5*exp(-t/3)+Cstarb*(1-exp(-t/3)); //x(t) for part (b)

//(c)
w1bar=500;w2bar=100; //flow rate changes, rest remains same
wbar=w1bar+w2bar;
tau=3;
x0=0.5;
Cstarc=(w1bar*x1bar+w2bar*x2bar)/wbar;  //C* variable
printf('\n (c) The value of C* is %f',Cstarc)
printf('\n x(t)=0.5exp(-t/3)+%f(1-exp(-t/3)) \n',Cstarc);
xtc=0.5*exp(-t/3)+Cstarc*(1-exp(-t/3));

//(d)
w1bar=500;w2bar=100;x1bar=0.6;x2bar=0.75; //flow rate changes, rest remains same
wbar=w1bar+w2bar;
tau=3;
x0=0.5;
Cstard=(w1bar*x1bar+w2bar*x2bar)/wbar;  //C* variable
printf('\n (d) The value of C* is %f',Cstard)
```

```
printf('\n x(t)=0.5exp(-t/3)+%f(1-exp(-t/3)) \n',
    Cstard);
xtd=0.5*exp(-t/3)+Cstard*(1-exp(-t/3));

plot2d(t,[xtd',xtb',xtc'])
xtitle('Parts b through d','Time(min)','$x(t)$');
a=legend("$(d)$","$(b)$","$(c)$",position=1);
a.font_size=5;
a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
c=a.y_label;c.font_size=5;

//(e)
xNb=(xtb-x0)/(Cstarb-x0); //Normalized response for
    part b
xNc=(xtc-x0)/(Cstarc-x0); //Normalized response for
    part c
xNd=(xtd-x0)/(Cstard-x0); //Normalized response for
    part d

scf() //Creates new window for plotting
plot2d(t,[xNd',xNb',xNc'],style=[1 1 1])
//Style sets the color, -ve values means discrete
    plotting, +ve means color
xtitle('Part e','Time(min)','Normalized response');
a=legend("$(e)$",position=1);
a.font_size=5;
a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
c=a.y_label;c.font_size=5;
```

Scilab code Exa 2.2 Degrees of freedom 1

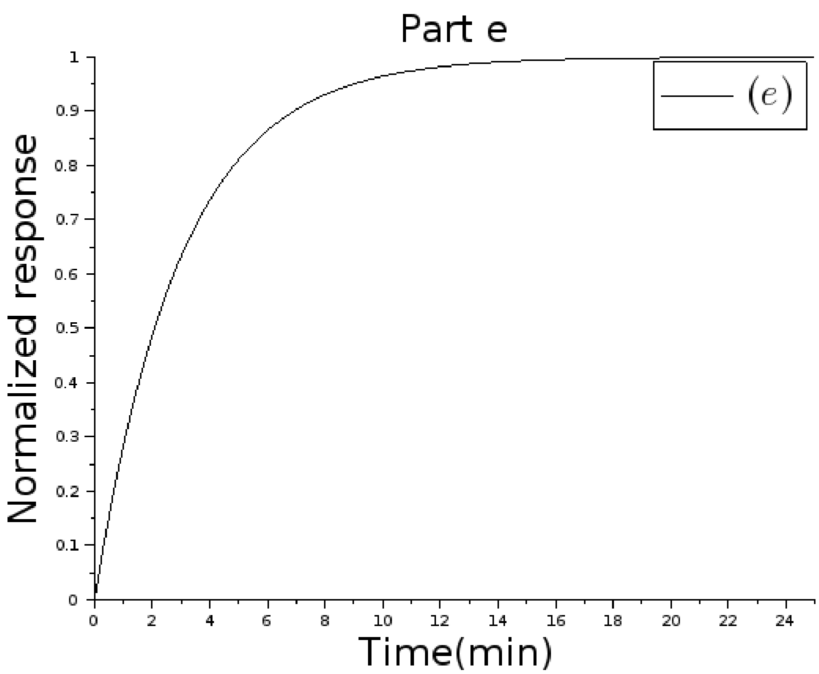


Figure 2.2: Stirred tank blending process

```
clear
clc

//Example 2.2
disp('Example 2.2')

N_V=4;
N_E=1;
N_F=N_V-N_E;
printf('\n Degrees of freedom N_F= %i \n',N_F)
```

Scilab code Exa 2.3 Degrees of freedom 2

```
clear
clc

//Example 2.3
disp('Example 2.3')


N_V=7;
N_E=2;
N_F=N_V-N_E;
printf('\n Degrees of freedom N_F= %i \n',N_F)
```

Scilab code Exa 2.4 Electrically heated stirred tank process

```
clear
clc

//Example 2.4
```

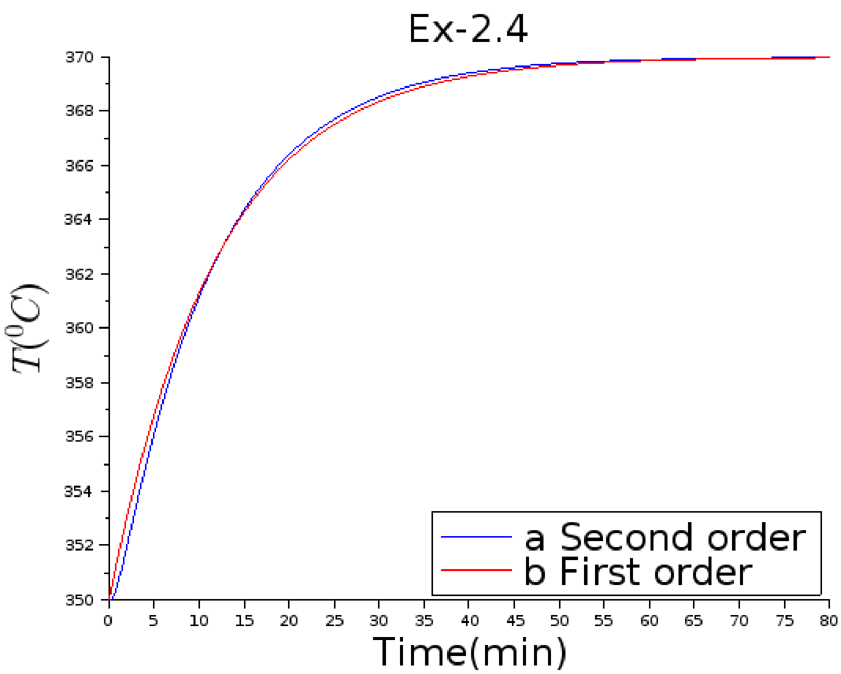


Figure 2.3: Electrically heated stirred tank process

```
disp('Example 2.4')

mprintf('\n Important Note: Errata for book: Values of the parameters \n...
 meCe/heAe and meCe/wC should be 1 min each and not 0.5 min %s \n','')

Tibar=100;//deg C
Qbar=5000;//kcal/min
wc_inv=0.05;// 1/wc degC min/kcal

//(a)
Tbar=Tibar+wc_inv*Qbar;
mprintf('\n (a) Nominal steady state temperature= %i',Tbar)
mprintf(' degree celsius %s \n','')

//(b)
mprintf('\n Eqn 2-29 becomes 10 d2T/dt2 + 12 dT/dt + T = 370  with T(0)=350 %s \n','')
t=0:0.1:80; //Time values
Tt_2=350+20*(1-1.089*exp(-t/11.099)+0.084*exp(-t/0.901));//T(t) from order 2 equation

//(c)
mprintf('\n Eqn 2-29 becomes 12 dT/dt + T = 370 with T(0)=350 %s \n','')
Tt_1=350+20*(1-exp(-t/12));//T(t) from order 1 equation


plot2d(t,[Tt_2',Tt_1'],[2 5],rect=[0 350 80 370])
xtitle('Ex-2.4','Time(min)','$T(^0C)$');
a=legend("a Second order","b First order",position=4);
a.font_size=5;
a=get("current_axes");b=a.title;b.font_size=5;c=a.x_label;c.font_size=5;
```

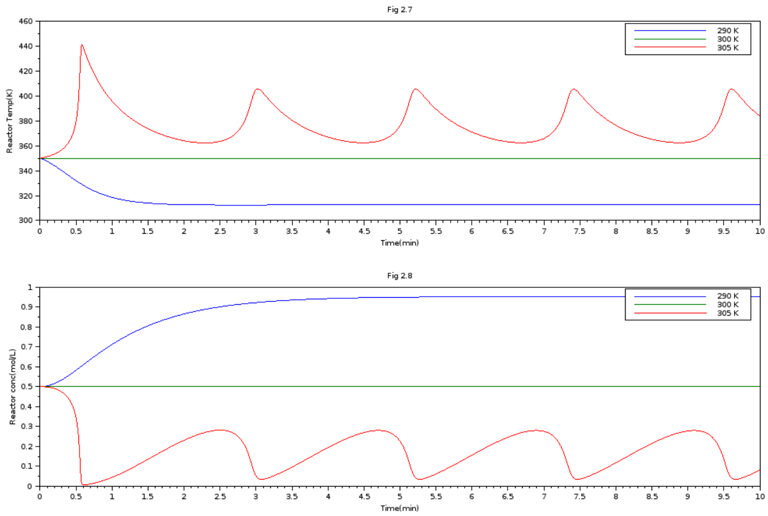


Figure 2.4: Nonlinear dynamic behavior of CSTR

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

Scilab code Exa 2.5 Nonlinear dynamic behavior of CSTR

```

clear
clc

//Example 2.5
disp('Example 2.5')

function ydot=CSTR(t,y,Tc) //y is [Conc Temp]' Tc is coolant temp
    q=100;ci=1;V=100;rho=1000;C=0.239;deltaHR=5E4;k0=7.2E10;UA=5E4;Er=8750;
    Ti=350;
```

```
    c=y(1);T=y(2);
    k=k0*exp(-Er/T);//Er=E/R
    ydot(1)=1/V*(q*(ci-c)-V*k*c);  //ydot(1) is
       dc_dt
    ydot(2)=1/(V*rho*C)*(q*rho*C*(Ti-T)+deltaHR*V*k*
       c+UA*(Tc-T))//ydot(2) is dT_dt
endfunction

c0=0.5;T0=350;
y0=[c0 T0]';
t0=0;
t=0:0.01:10;
Tc=[290 305];
y1 = ode(y0,t0,t,list(CSTR,Tc(1)));
y2 = ode(y0,t0,t,list(CSTR,Tc(2)));
y3=[0.5 0;0 350]*ones(2,length(t))
//Temp plot
subplot(2,1,1);
plot(t,[y1(2,:)' y3(2,:)' y2(2,:)']);
xtitle("Fig 2.7","Time(min)","Reactor Temp(K)");
legend("290 K","300 K","305 K")
//conc plot
subplot(2,1,2);
plot(t,[y1(1,:)' y3(1,:)' y2(1,:)']);
xtitle("Fig 2.8","Time(min)","Reactor conc(mol/L)");
legend("290 K","300 K","305 K");
```

Chapter 6

Development of Empirical Models from Process Data

Scilab code Exa 6.1 Gas turbine generator

```
clear
clc

//Example 6.1
disp('Example 6.1')

//Fuel flow rate appended with ones for intercept in
    regression
fuel=[1 2.3 2.9 4 4.9 5.8 6.5 7.7 8.4 9];
X=[ones(1,10);fuel]';
Y=[2 4.4 5.4 7.5 9.1 10.8 12.3 14.3 15.8 16.8]'; //
   Power generated

Bhat=inv(X'*X)*X'*Y;

mprintf('\n Linear model \n B1_hat=%f \n B2_hat=%f',
   Bhat')


```

```
//For better accuracy we can fit higher order model
X_new=[ones(1,10);fuel;fuel.^2]';
Bhat_new=inv(X_new'*X_new)*X_new'*Y;
mprintf('\n \n Quadratic model \n B1_hat=%f \n B2_hat=%f \n B3_hat=%f',Bhat_new')
Output_table=[fuel' Y X*Bhat X_new*Bhat_new];

//mprintf('\n Fuel      Power Generated      Linear Model     Quadratic Model %f %f',Output_table(:,1),Output_table(:,2))
//disp(Output_table)

//Table 6.1
mprintf('\n \n Table 6.1  %s','')
mprintf('\n        ui         yi         Linear Model        Quadratic Model  %s','')

mprintf('\n    %f   %f    %f    %15f',Output_table)


//Error calculations ----(This is not given in book-requires understanding of statistics)
Yhat=X*Bhat; //Predicted Y from regression variables
S_lin=(Y-Yhat)'*(Y-Yhat); //Sum of errors in Y for linear model --eqn 6.9
S_quad=(Y-X_new*Bhat_new)'*(Y-X_new*Bhat_new); //Errors in Y for quadratic model
mprintf('\n    %25s%f    %10s%f','S=',S_lin,'S=',S_quad)

n=length(fuel);
sigma=S_lin/(n-1)*(inv(X'*X));
bounds=(sigma.^0.5)/sqrt(n)*2.262;

mprintf('\n The errors in Bhats are not calculated because the procedure is not...
\n given in the solution of the example')
```

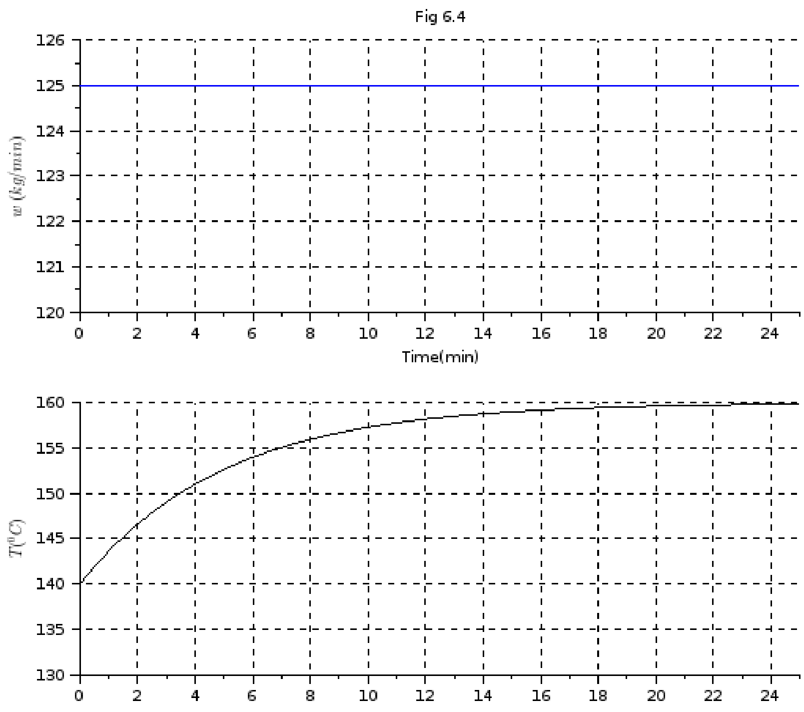


Figure 6.1: Continuous stirred tank reactor

Scilab code Exa 6.2 Continuous stirred tank reactor

```
clear
clc

//Example 6.2
disp('Example 6.2')

deltaw=5;//kg/min
deltaT=20;//deg C
```

```
K=deltaT/deltaw
tau=5//min
T=140+0.632*20;//152.6 deg C

s=%s;
G=4/(5*s+1); //G=T'(s)/W'(s)

mprintf('\n T(s)/W(s)=%s','')
disp(G)

t=0:0.01:25;
n=length(t);
w=5*ones(1,n);
yt=csim(w,t,G)+140;
wt=w+120;
subplot(2,1,2);
plot2d(t,yt,rect=[0,130,25,160]);
xtitle("","Time(min)","$T(^0C)$")
xgrid();
subplot(2,1,1);
plot2d(t,wt,rect=[0,120,25,126],style=2)
xtitle("Fig 6.4","Time(min)","$w\ (kg/min)$")
xgrid();
```

Scilab code Exa 6.3 Off gas C02 concentration response

```
clear
clc

//Example 6.3
disp('Example 6.3')


```

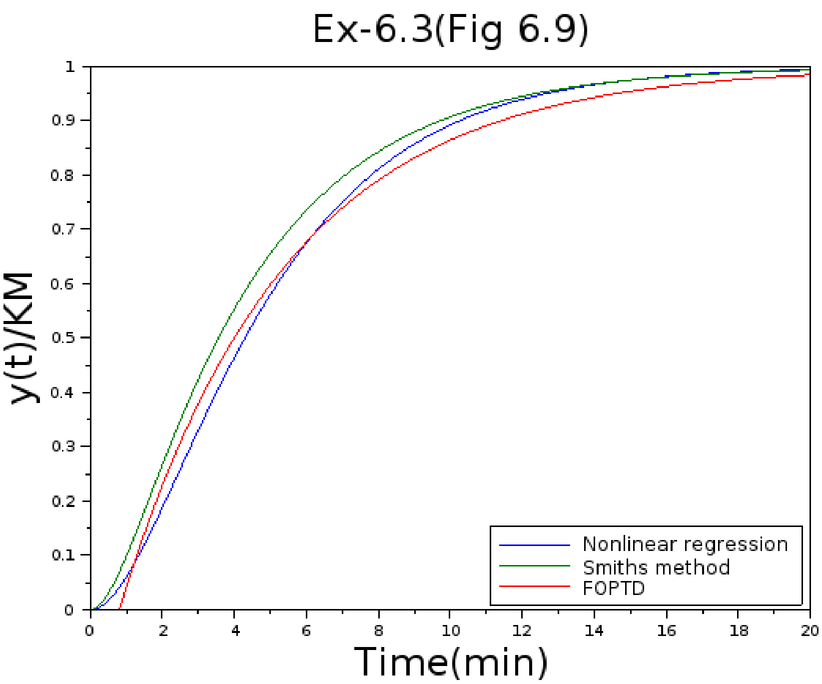


Figure 6.2: Off gas C02 concentration response

```
//Smith's method
t20=1.85;//min
t60=5;//min
ratio=t20/t60;
zeta=1.3;//Zeta obtained from Fig 6.7 page 109
tau=t60/2.8//Value 2.8 obtained from Fig 6.7

tau1=tau*zeta+tau*sqrt(zeta^2-1);
tau2=tau*zeta-tau*sqrt(zeta^2-1);

mprintf('From Smiths method \n  tau1=%f min\n  tau2=%f min \n',[tau1 tau2])

//Nonlinear regression
//This method cannot be directly used here because we do not have the data with us
//Had the data been given in tabular form we could have performed a regression
//Converting graphical data(Fig 7.8) printed in textbook to tabular form is not practical
//Towards the end of the program however a roundabout way has been implemented so
//that the user can learn the technique of non-linear optimization



s=%s;
G2=1/((tau1*s+1)*(tau2*s+1)) //Smith's method
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
//Also note that we could have used Pade's approximation but that works well only for very small time delays
G1=1/((3.34*s+1)*(1.86*s+1)); //Nonlinear regression

Glist=syslin('c',[G1 G2 G3]') //Simply collating
```

```
them together for further simulation

G=syslin('c',Glist);
t=0:0.1:20;
y=csim('step',t,G);
y(3,:)=[zeros(1,8) y(3,1:$-8)] //This is the roundabout trick to introduce time lag by manually
//shifting the respone by 0.7 min
plot(t,y)
xtitle('Ex-6.3(Fig 6.9)','Time(min)','y(t)/KM');
a=legend("Nonlinear regression","Smiths method","FOPTD",position=4);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=5;c=a.x_label;c.font_size=5;
c=a.y_label;c.font_size=5;


//====NON-LINEAR REGRESSION====//
//Now that we have the response data and also taking the word from the book that
//simulation from Excel/Matlab is identical to experimental data, we can actually
//take this simulation and use it for showing regression
//So our experimental data is y(1)
//For nonlinear regression we define a cost function which we have to minimize
function err=experiment(tau)
    s=%s;
    G=syslin('c',1/((tau(1)*s+1)*(tau(2)*s+1)));
    t=0:0.1:20;
    response=csim('step',t,G);
    err=sum((response-y(1,:)).^2);
endfunction

//f is value of cost function, g is gradient of cost
```

```
    function ,
//ind is just a dummy variable required by optim
   function
function [f,g,ind]=cost(tau,ind)
    f=experiment(tau)
    g=numdiff(experiment,tau)
endfunction

x0=[3 1]'; //Initial guess for optimization routine
[cost_opt, tau_opt]=optim(cost,x0)
mprintf('\n Optimization using optim function done
   successfully \n')
mprintf('\n From nonlinear regression \n  tau1=%f
   min\n  tau2=%f min \n',[tau_opt]')



//Formatted output...
mprintf('\n                      tau1(min) tau2(min)
   Sum of squares')
mprintf('\n       Smith          %f   %f   %f'
   ,3.81,0.84,0.0769)
mprintf('\n First Order\n(delay=0.7min)        %f
       %s      %f',4.60,'--',0.0323)
mprintf('\n Excel and Matlab     %f   %f   %f \n'
   ,3.34,1.86,0.0057)
```

Scilab code Exa 6.5 Estimation of model parameters

```
clear
clc

//Example 6.5
disp('Example 6.5')

```

```
k=0:10';
yk=[0 0.058 0.217 0.360 0.488 0.600 0.692 0.772 0.833 0.888 0.925]';

Y=yk(2:$);
n=length(Y);

yk_1=[yk(1:$-1)];
yk_2=[0;yk(1:$-2)];
uk_1=ones(n,1);
uk_2=[0;uk_1(1:$-1)];

X=[yk_1 yk_2 uk_1 uk_2];

Bhat=inv(X'*X)*X'*Y;//Equation 6-10
//a1, a2, b1, b2 are components of Bhat for linear regression
K_lin=(Bhat(3)+Bhat(4))/(1-Bhat(1)-Bhat(2)); //Equation 6-42

//=====NON-LINEAR REGRESSION=====//
//Now that we have the response data we can do non-linear regression through
//the transfer function approach. Total no. of variables to be regressed are
//three: K, tau1, tau2.
//For nonlinear regression we define a cost function which we have to minimize


function err=experiment(tau)
    K=tau(3);tau1=tau(1);tau2=tau(2);
    t=k';
    response=tau(3)*(1-(tau1*exp(-t/tau1)-tau2*exp(-t/tau2))/(tau1-tau2))
    err=sum((response-[yk]).^2);
endfunction

```

```
//f is value of cost function, g is gradient of cost function,
//ind is just a dummy variable required by optim function
function [f,g,ind]=cost(tau,ind)
    f=experiment(tau)
    g=numdiff(experiment,tau)
endfunction

x0=[1 3 1]'; //Initial guess for optimization routine
[cost_opt, tau_opt]=optim(cost,x0)
mprintf('\n Optimization using optim function done successfully \n')
mprintf('\n From nonlinear regression \n  tau1=%f \n  tau2=%f min\n  K=%f min \n',[tau_opt]')

//Now we have to get discrete ARX model parameters from transfer function parameters
//For this we use Equation nos.: 6-32 to 6-36

deltat=1;taua=0;tau1=tau_opt(1);tau2=tau_opt(2);K=tau_opt(3);
a1=exp(-deltat/tau1)+exp(-deltat/tau2);
a2=-exp(-deltat/tau1)*exp(-deltat/tau2);
b1=K*(1+(taua-tau1)/(tau1-tau2)*exp(-deltat/tau1)-(taua-tau2)/(tau1-tau2)*exp(-deltat/tau2));
b2=K*(exp(-deltat*(1/tau1+1/tau2))+(taua-tau1)/(tau1-tau2)*exp(-deltat/tau2)-(taua-tau2)/(tau1-tau2)*exp(-deltat/tau1));

mprintf("\n         Linear Regression          Non-Linear Regression")
mprintf("\n    %s       %f          %20f",["a1";"a2";"b1";"b2";"K"],[[Bhat;K_lin] [a1;a2;b1;b2;K]])

yL_hat=X*Bhat;
yN_hat=X*[a1;a2;b1;b2];
```

```
mprintf("\n \n     n              y             yL_hat           yN_hat")
mprintf("\n      %f     %f     %f        %f",[1:10]',yk(2:$),yL_hat,yN_hat)

mprintf("\n \n Note that values of coefficients for non-linear regression \n are different...
from that of linear regression, but the\n output is the same \n...
hence this shows that the coefficients need not be unique....
\n the coefficients for nonlinear regression given in book and from this scilab code\n...
 both are correct")
```

Scilab code Exa 6.6 Step test of distillation column

```
clear
clc

//Example 6.6
disp('Example 6.6')

mprintf("\n  It is not possible to fit Model 1 or \n  plot it because experimental data...
  has not been given in the book. \n  Hence we simply plot Model 2,3,4 \n")


//Model 2
a=[3.317 -4.033 2.108 0.392 ]'
b=[-0.00922 0.0322 -0.0370 0.0141]';
```

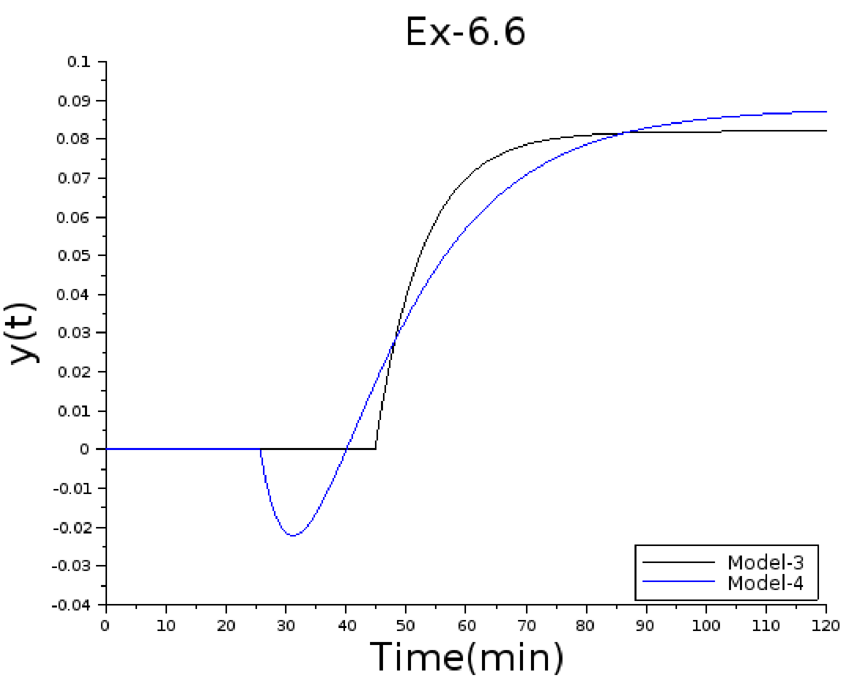


Figure 6.3: Step test of distillation column

```
uk=[ones(120,1)];    //Inputs-step at t=1 min
yk=zeros(120,1);     //Outputs initialization

for k=5:120
    yk(k)=a(1)*yk(k-1)+a(2)*yk(k-2)+a(3)*yk(k-3)+a(4)*yk(k-4)...
            +b(1)*uk(k-1)+b(2)*uk(k-2)+b(3)*uk(k-3)+b(4)*uk(k-4);
end
//Model 2 trial with transfer function
//a=-flipdim([-1 3.317 -4.033 2.108 0.392 ]',1);
//b=flipdim([-0.00922 0.0322 -0.0370 0.0141]',1);
//
//Gz=poly(b,"z","coeff")/poly(a,"z","coeff");
//u=ones(120,1);
//Gz=syslin('d',Gz);
//yk=flts(u',Gz)

//Although the code is correct, the values given in the book for the coeffs
//of the ARX model are wrong and hence the system diverges(blows up)

mprintf('Although the code is correct, the values \n given in the book for the coeffs of model 2...
\n of the ARX model are wrong and hence the system diverges(blows up)')

//Model 3
s=%s;
G3=0.082/(7.95*s+1);//We have to add delay of 44.8 min
//Model 4
G4=0.088*(1-12.2*s)/(109.2*s^2+23.1*s+1);//We have to add delay of 25.7 min

G=syslin('c',[G3;G4]);
t=[0:0.1:120]';
```

```
y=csim('step',t,G);

y(1,:)=[zeros(1,448) y(1,1:$-448)]
y(2,:)=[zeros(1,257) y(2,1:$-257)]
plot2d(t,y')

xtitle('Ex-6.6','Time(min)','y(t)');
a=legend("Model-3","Model-4",position=4);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
c=a.y_label;c.font_size=5;
```

Chapter 8

Control System Instrumentation

Scilab code Exa 8.2 Pump and heat exchanger system

```
clear
clc

//Example 8.2
disp('Example 8.2')

//Eqn 8-6

//Pump characteristics
q=0:0.1:240;
Phe=30*(q/200).^2;
plot2d(q,Phe,rect=[0,0,240,40]);
xgrid()
xtitle("Fig 8.13 Pump characteristics","q,gal/min","
   P,psi")
scf();

q=200;//Flow rate in gal/min
Phe=30*(q/200).^2;
```

```
Pv=40-Phe; //Eqn 8-8


//(a)
l=0.5;Pv=10;
Cv=q/l/sqrt(Pv);

mprintf("(a) The value of coefficient Cv is %f",Cv)

//plotting valve characteristic curve

l=[0:0.01:0.8]';
n=length(l);
Cv=125;

function y=valve_1(q)
    Pv=40-30*(q/200).^2;
y=Cv*l.*sqrt(Pv)-q;
endfunction

[q_valve1,f1]=fsolve(200*ones(n,1),valve_1); //200*ones(n,1) is the initial guess for q

plot2d(l,q_valve1);

//(b)
q=200*110/100; //110% flow rate
Phe=30*(q/200).^2;
Pv=40-Phe; //Eqn 8-8
l=1;
Cv=q/sqrt(Pv)/l;
mprintf("\n(b) The value of coefficient Cv is %f",Cv)

//We use Cv=115;
Cv=115;
l=[0.2:0.01:0.9]';
n=length(l);
```

```
R=50;

function y=valve_2(q)
    Pv=40-30*(q/200).^2;
    y=[R^(l-1)]*Cv.*sqrt(Pv)-q;
endfunction
[q_valve2,f2]=fsolve(150*ones(n,1),valve_2);
plot2d(l,q_valve2,style=2)

//(c)
Cv=1.2*115;
mprintf("\n(c) The value of coefficient Cv is %f",Cv)

l=[0.2:0.01:0.9]';
n=length(l);
R=50;

function y=valve_3(q)
    Pv=40-30*(q/200).^2;
    y=[R^(l-1)]*Cv.*sqrt(Pv)-q;
endfunction
[q_valve3,f3]=fsolve(linspace(60,200,n)',valve_3); //Initial guess has to be smart for each valve,
//since we want near linear profile we can give a linear initial guess
plot2d(l,q_valve3,style=3)

//(d)
Cv=0.8*115;
mprintf("\n(c) The value of coefficient Cv is %f",Cv)

l=[0.2:0.01:0.9]';
n=length(l);
R=50;

function y=valve_4(q)
```

```
    Pv=40-30*(q/200).^2;
    y=[R^(l-1)]*Cv.*sqrt(Pv)-q;
endfunction
[q_valve4,f4]=fsolve(linspace(60,200,n)',valve_4); //Initial guess has to be smart for each valve,
//since we want near linear profile we can give a linear initial guess
plot2d(l,q_valve4,style=4,rect=[0,0,1,240])

xtitle('Ex-8.2 Installed valve characteristics','$l$','q gal/min');
a=legend("Valve 1, linear Cv=125","Valve 2, Equal% Cv=115","Valve 3, Equal% Cv=138","Valve 4, Equal% Cv=92",position=4);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=3;c=a.x_label;c.font_size=5;
c=a.y_label;c.font_size=5;
```

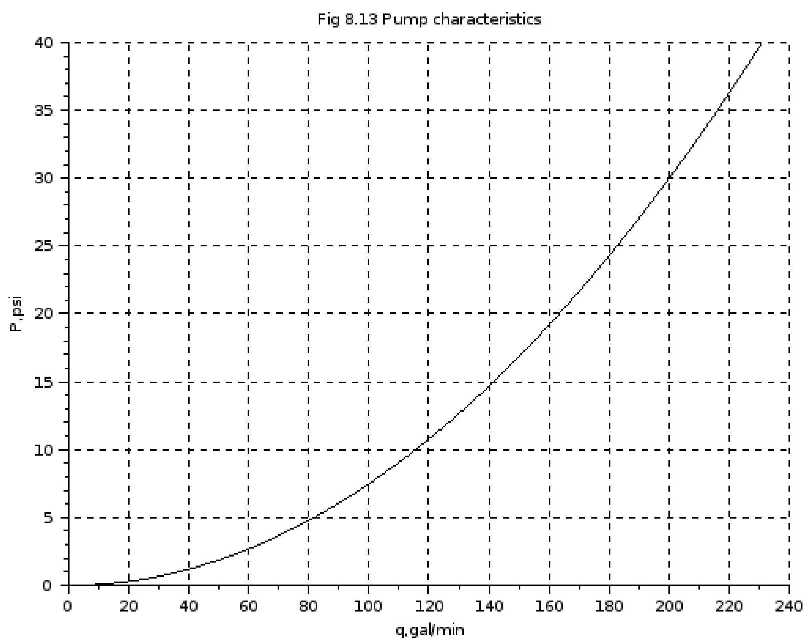


Figure 8.1: Pump and heat exchanger system

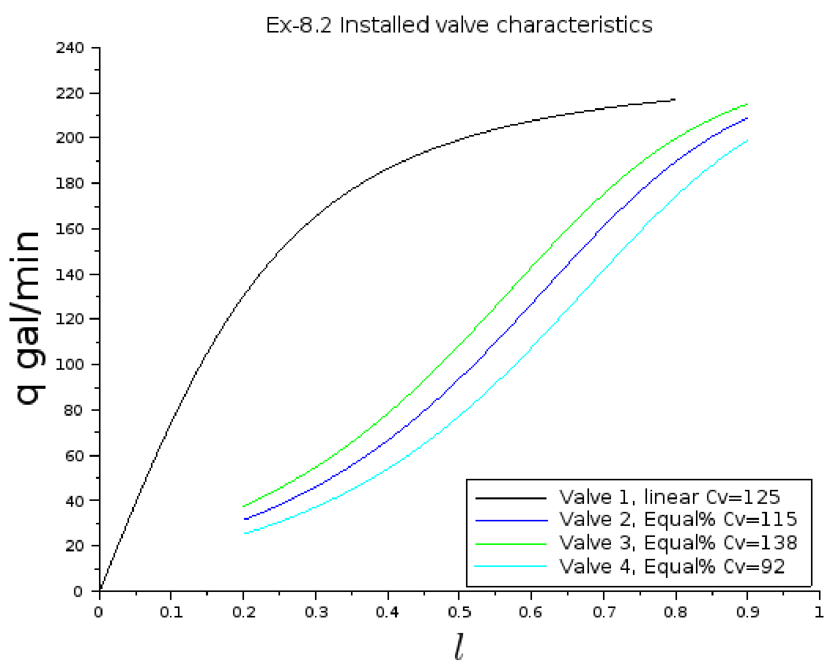


Figure 8.2: Pump and heat exchanger system

Chapter 9

Process Safety and Process Control

Scilab code Exa 9.1 Liquid surge system

```
clear
clc

//Example 9.1
disp('Example 9.1')

q1=12;//cub ft/min
q2=6;
q3=13;
A=%pi*3^2;//ft^2
delta_t=10;//min
delta_h_max=delta_t*(q1+q2-q3)/A;

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

```
clear
clc

//Example 9.2
disp('Example 9.2')

mu=[0.5 0.8 0.2]; //population means of z y x
S=[0.01 0.020 0.005];//population std dev of z y x

z=[0.485]; //steady state values
y=[0.825];
x=0.205;

F=4;D=2;B=2; //flow rates

Ec=F*z-D*y-B*x;

disp(Ec,"Ec=")

sigma_Ec=sqrt(F^2*S(1)^2+D^2*S(2)^2+B^2*S(3)^2)

disp(sigma_Ec,"sigma_Ec")



Z=(Ec-0)/sigma_Ec;

disp(Z,"Z=");

[P,Q]=cdfnor("PQ",0.120,0,sigma_Ec);

//Since P is close to 1, we use Q

Probability=1-2*Q;

disp(Probability,"Probability of abnormal event=")
```

Scilab code Exa 9.3 Reliability of flow control loop

```

clear
clc

//Example 9.3
disp('Example 9.3')

mu=[1.73 0.05 0.49 0.60 0.44]';//failures/yr
R=exp(-mu);
P=1-R;

R_overall=prod(R);
P_overall=1-R_overall;
mu_overall=-log(R_overall);
MTBF=1/mu_overall;

mprintf("MTBF= %f yr",MTBF)
```

Chapter 10

Dynamic behavior

Scilab code Exa 10.2 Set point response of level control system

```
clear
clc

//Example 10.2
disp('Example 10.2')

A=%pi*0.5^2; //Square meters
R=6.37;
Kp=R//min/sq.m=R
tau=R*A;

Km=100/2; //% per meter

l=0.5;
q=0.2*30^(l-1);
dq_dl=0.2*log(30)*30^(l-1); //cu.meter/min Eqn 10-48

Kip=(15-3)/100;//psi/%
dl_dpt=(1-0)/(15-3); //psi^-1
```

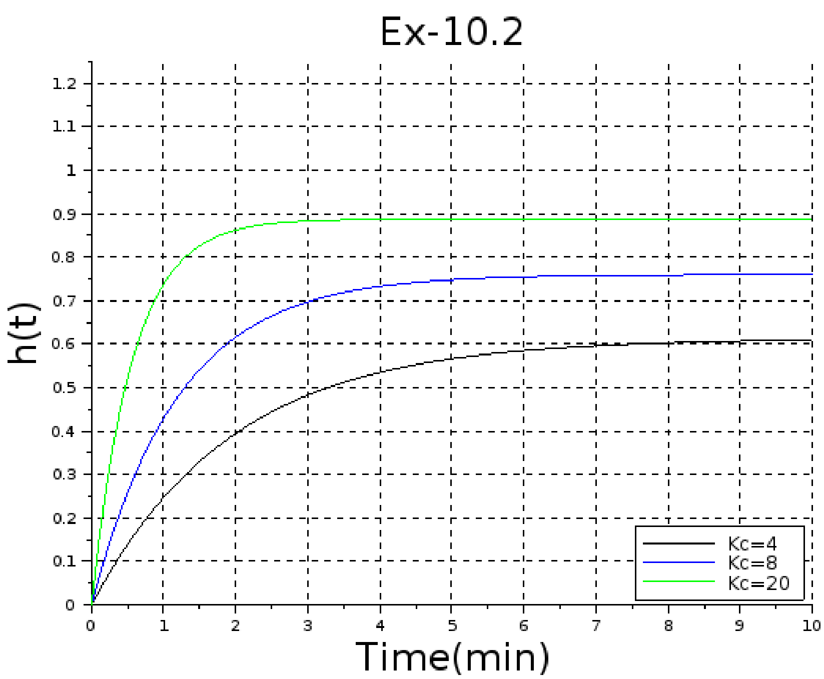


Figure 10.1: Set point response of level control system

```

Kv=dq_dl*dl_dpt //Eqn 10-50

Kc=[4 8 20]'; //different values of Kc that we have to try
K_OL=Kc*Kv*Kp*Km*Kip;//Open loop gain Eqn 10-40

K1=K_OL./(1+K_OL);//Eqn 10-38
tau1=tau./(1+K_OL); //Eqn 10-39

//Now we simulate the close loop process for these different values of K1 and tau1
s=%s;
G=K1./(tau1*s+1);
G=syslin('c',G);
t=[0:0.1:10]'; //time in minutes
hdash=csim('step',t,G)';

plot2d(t,hdash,rect=[0,0,10,1.25])
xgrid()
xtitle('Ex-10.2','Time(min)','h(t)');
a=legend("Kc=4","Kc=8","Kc=20",position=4);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=5;c=a.x_label;c.font_size=5;
c=a.y_label;c.font_size=5;
```

Scilab code Exa 10.3 Disturbance response of level control system

```
clear
clc

//Example 10.3
```

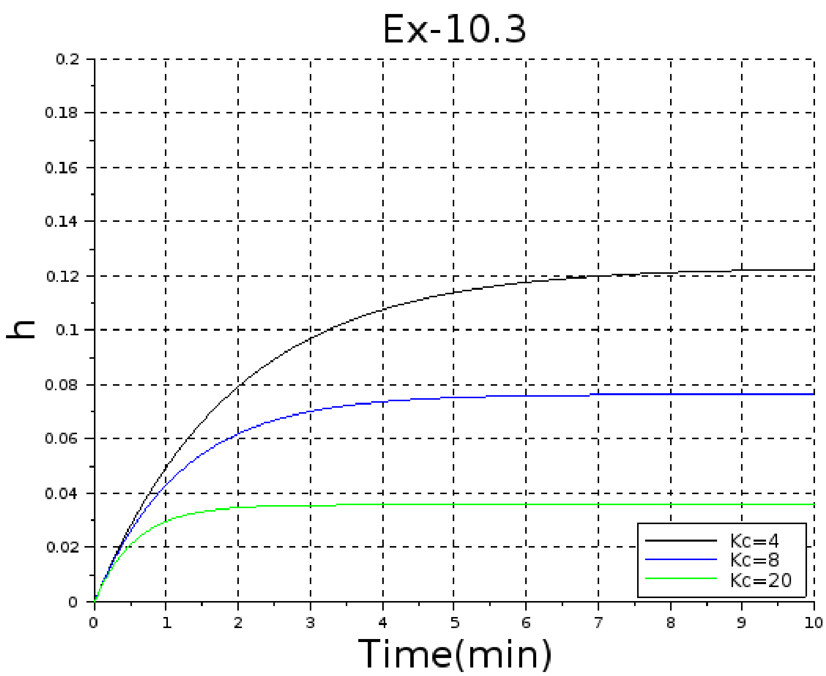


Figure 10.2: Disturbance response of level control system

```
disp('Example 10.3')

//This example draws upon the calculations of Ex 10.2 and hence it has been
//reproduced
A=%pi*0.5^2; //Square meters
R=6.37;
Kp=R//min/sq.m=R
tau=R*A;
Km=100/2 //% per meter
l=0.5;
q=0.2*30^(l-1);
dq_dl=0.2*log(30)*30^(l-1); //cu.meter/min Eqn 10-48
Kip=(15-3)/100;//psi/%
dl_dpt=(1-0)/(15-3); //psi^-1
Kv=dq_dl*dl_dpt //Eqn 10-50
Kc=[4 8 20]'; //different values of Kc that we have to try
K_OL=Kc*Kip*Kv*Kp*Km;//Open loop gain Eqn 10-40
K1=K_OL./(1+K_OL);//Eqn 10-38
tau1=tau./(1+K_OL); //Eqn 10-39

//==========Example 11.3 now starts here==========//
//Now we simulate the close loop process for these different values of K2 and tau1
M=0.05;//Magnitude of step
K2=Kp./(1+K_OL);
s=%s;
G=K2./(tau1*s+1);
G=syslin('c',G);
t=[0:0.1:10]'; //time in minutes
hdash=M*csim('step',t,G)';

plot2d(t,hdash,rect=[0,0,10,0.2])
xgrid()
xtitle('Ex-10.3','Time(min)','h');
a=legend("Kc=4","Kc=8","Kc=20",position=4);
a.font_size=2;
```

```
a=get("current_axes");b=a.title;b.font_size=5;c=a.x_label;c.font_size=5;
c=a.y_label;c.font_size=5;

offset=-Kp*M./(1+K_OL);

mprintf("   Kc     Offset")
mprintf("\n%f    %f",[Kc offset])

mprintf("\n\nNote that the book has a mistake in the question and legend of fig 10.19\n...
the values of Kc should be 4,8,20 and not 1,2,5\n...
this mistake is there because in the second edition of the book\n...
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

```
clear
clc

//Example 10.4
disp('Example 10.4')

Km=1; //We take set point gain as 1 for illustrative purposes
Kc=[15 6 2]'; //different values of Kc for which we will simulate
Gc=Kc;
s=%s;
Gv=1/(2*s+1);
Gd=1/(5*s+1);
```

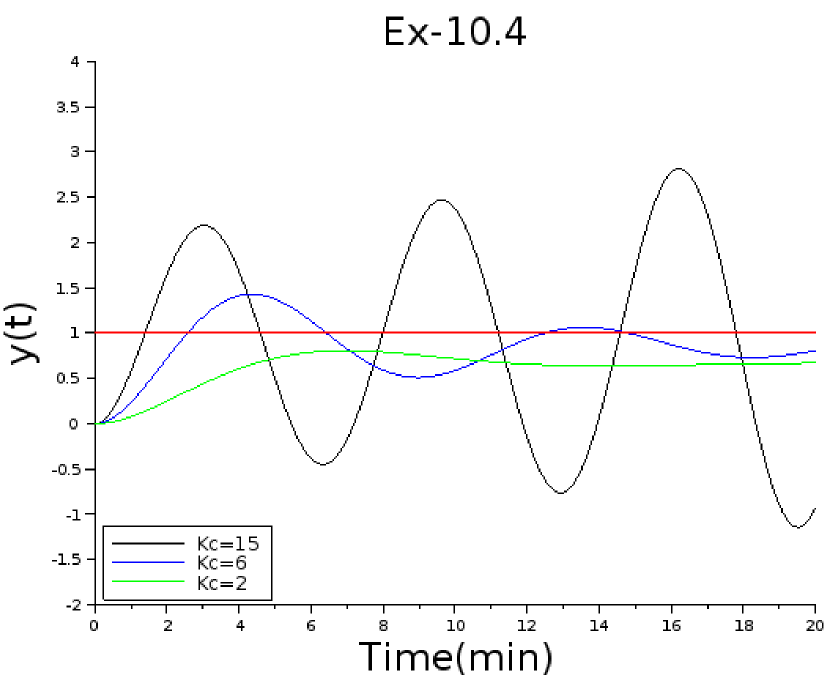


Figure 10.3: Stability of feedback control system

```
Gp=Gd;
Gm=1/(s+1);

G=Km*Gc*Gv*Gp./(1+Km*Gc*Gv*Gp*Gm); //Eqn 10-75 G=Y/
    Ysp

//Now we simulate the close loop process for these
    different values of Kc

G=syslin('c',G);
t=[0:0.1:20]'; //time in minutes
Y=csim('step',t,G)';

plot2d(t,Y,rect=[0,-2,20,4])
plot2d(t,ones(length(t),1),style=5)
xtitle('Ex-10.4','Time(min)','y(t)');
a=legend("Kc=15","Kc=6","Kc=2",position=3);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
c=a.y_label;c.font_size=5;
```

Scilab code Exa 10.10 Routh Array 1

```
clear
clc

//Example 10.10
disp('Example 10.10')


s=%s;
Gp=1/(5*s+1);
Gm=1/(s+1);
Gv=1/(2*s+1);
```

```
Ys=Gv*Gp*Gm

Routh=routh_t(Ys,poly(0,"Kc")); // produces routh table for polynomial 1+Kc*Ys
disp(Routh)
K1=roots(numer(Routh(3,1)));
K2=roots(numer(Routh(4,1)));

mprintf('K lies between %f and %f for system to be stable', K2,K1)
```

Scilab code Exa 10.11 Routh Array 2

```
clear
clc

//Example 10.11
disp('Example 10.11')

Kc=poly(0,"Kc");//defining a polynomial variable
a2=2.5;a1=5.5-Kc;a0=1+2*Kc;//a# are coefficients
b1=(a1*a0-a2*0)/a1;
mprintf("Routh Array is")
A=[a2 a0;a1 0;b1 0]
disp(A)

mprintf("\n All entries in first column should be positive")

Kc_up=roots(a1);//upper limit for Kc by solving third row column 1 of array
b1=numer(b1);//This is done to extract the numerator from rational c1
//without extracting numerator we cannot find roots using "roots" function
```

```
Kc_ul=roots(b1);//lower limit for Kc

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

```
clear
clc

//Example 10.12
disp('Example 10.12')

w=poly(0,"w")
s=%i*w;  //j or iota is i
G=10*s^3+17*s^2+8*s+1;//Kc has been removed here because in a single expression
//two polynomials are not allowed
w=roots(imag(G));
p=-real(G)//Real part of G
Kc=horner(p,w)

mprintf("The values outside which system is unstable \nare %f and %f",Kc(1),Kc(3))
```

Scilab code Exa 10.13 Root Locus

```
clear
clc

//Example 10.13
```

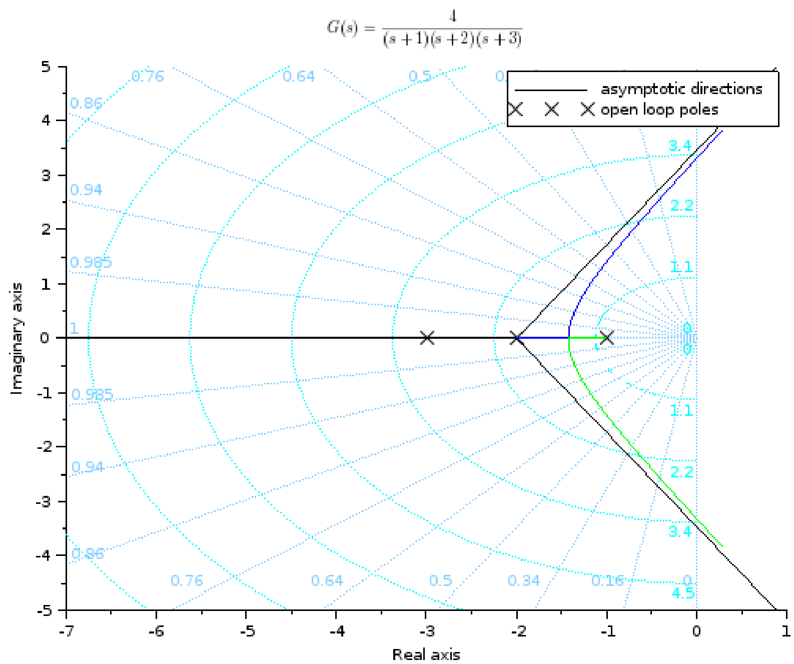


Figure 10.4: Root Locus

```
disp('Example 10.13')

s=%s;
G=4/((s+1)*(s+2)*(s+3));
G=syslin('c',G);
[ki,s_i]=kpure(G);
evans(G,ki*1.5); // plots for until K = 1.5*ki
//disp(G,"For G=");disp(ki,"K=")
disp(ki,"Max value of k for which we have closed loop stability",G,"For G=")
xtitle("$G(s)=\frac{4}{(s+1)(s+2)(s+3)}$")
sgrid();
```

Scilab code Exa 10.14 Transient response from root locus

```
clear
clc

//Example 10.14
disp('Example 10.14')

s=%s;
G=4/((s+1)*(s+2)*(s+3));
K=10; //given in question
p=1+K*G;//characteristic equation
q=roots(numer(p));

q_abs=abs(q);
q_real=real(q);
q_imag=imag(q);
d=q_abs(2);
psi=%pi-acos(q_real./q_abs);//angle in radians
tau=1/d;
eta=cos(psi)

```

```
mprintf("\nd=%f\npsi=%f degrees\ntau=%f time units\neta=%f",d,psi(2)*180/%pi,tau,eta(2))

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

```
clear
clc

//Example 11.1
disp('Example 11.1')

//(a) Desired closed loop gain=1 and tau=[1 3 10]
s=%s;
tauc=[1 3 10]';
tau1=10;tau2=5;K=2;theta=1; //Time delay
Y_Ysp=(1)./(tauc*s+1); //Y/Ysp=delay/(tau*s+1)  Eqn
    11-6

//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
delay=(1-theta/2*s)/(1+theta*s/2);//first order Pade
     approx

```

```
G=(K)./((tau1*s+1)*(tau2*s+1))*delay;
G_tilda=G//Model transfer function

//Eqn-11-14
Kc=1/K*(tau1+tau2)./(tauc+theta);tauI=tau1+tau2;tauD=tau1*tau2/(tau1+tau2);
Gc=Kc*(1+1/tauI/s+tauD*s); //PID without derivative filtering
G_CL=syslin('c',Gc/delay*G./(1+Gc*G));//closed loop transfer function
t=0:160;
y=csim('step',t,G_CL);
//plot(t,y)

t_d=81:160;
G_CL_dist=syslin('c',G/delay./(1+Gc*G));//closed loop wrt disturbance
u_d=[0 ones(1,length(t_d)-1)]
y_d=csim('step',t_d,G_CL_dist);
y(:,81:160)=y(:,81:160)+y_d
plot(t,y)

xgrid()
xtitle('Ex-11.1 Correct Model','Time(min)','y(t)');
a=legend("$\tau_c=1$","$\tau_c=3$","$\tau_c=10$",position=4);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=5;c=a.x_label;c.font_size=5;
c=a.y_label;c.font_size=5;

mprintf("\n                          tauc=1        tauc=3        tauc=10")
mprintf("\n Kc(K_tilda=2)  %10f     %f        %f",Kc');


//Simulation for model with incorrect gain
scf()
```

```
K_tilda=0.9

//Eqn-11-14
Kc=1/K_tilda*(tau1+tau2)./(tauc+theta);tauI=tau1+tau2;tauD=tau1*tau2/(tau1+tau2);
Gc=Kc*(1+1/tauI/s+tauD*s)
mprintf("\n Kc(K_tilda=0.9) %10f    %f       %f",Kc');
mprintf("\n tauI %20f    %f       %f",tauI*ones(1,3));
mprintf("\n tauD %20f    %f       %f",tauD*ones(1,3));

G_CL=syslin('c',Gc*G./(1+Gc*G));//closed loop transfer function
t=0:160;
y=csim('step',t,G_CL);

t_d=81:160;
G_CL_dist=syslin('c',G./(1+Gc*G));//closed loop wrt disturbance
y_d=csim('step',t_d,G_CL_dist);
y(:,81:160)=y(:,81:160)+y_d
plot(t,y)

xgrid()
xtitle('Ex-11.1 Model with incorrect gain','Time(min)','y(t)');
a=legend("$\tau_c=1$","$\tau_c=3$","$\tau_c=10$",position=4);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=5;c=a.x_label;c.font_size=5;
c=a.y_label;c.font_size=5;

mprintf('\n \nThere is a slight mis-match between graphs from scilab code\n...
and those given in the book because of Pade approx
```

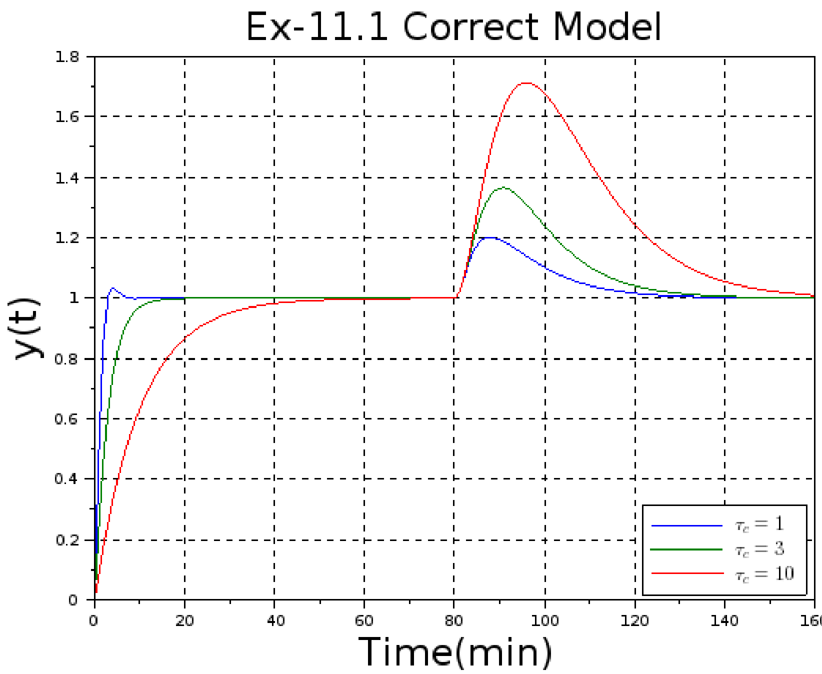


Figure 11.1: Direct synthesis for PID

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

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

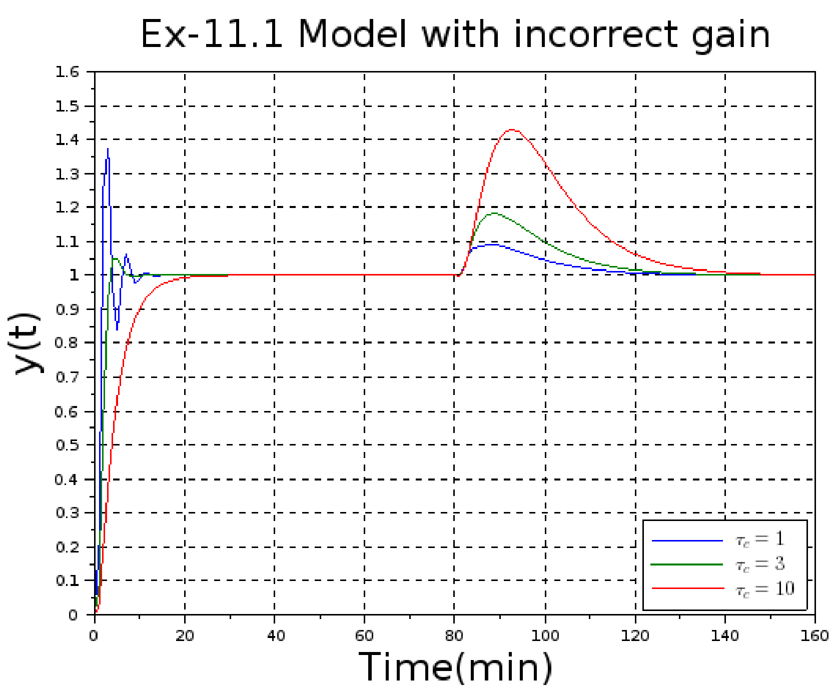


Figure 11.2: Direct synthesis for PID

```
clear
clc

//Example 11.3
disp('Example 11.3')

//(a)
K=0.2;theta=7.4;tauc=[8 15]';

Kc1=1/K*(2*tauc+theta)./(tauc+theta).^2; //Row M
Kc2=1/K*(2*tauc+theta)./(tauc+theta/2).^2; //Row N
tauI=2*tauc+theta;
tauD=(tauc*theta+theta^2/4)./(2*tauc+theta);

mprintf('                    Kc              tauI              tauD')
mprintf('\nPI(tauC=8)    %f      %f       %f',Kc1(1),tauI(1),0)
mprintf('\nPI(tauC=15)    %f      %f       %f',Kc1(2),tauI(2),0)
mprintf('\nPID(tauC=8)    %f      %f       %f',Kc2(1),tauI(1),tauD(1))
mprintf('\nPID(tauC=15)    %f      %f       %f',Kc2(2),tauI(2),tauD(2))

s=%s;

//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
delay=(1-theta/2*s+theta^2/10*s^2)/(1+theta/2*s+theta^2/10*s^2);//second order pade approx
//delay=(1-theta/2*s)/(1+theta/2*s);//first order pade approx
G=K*delay/s;
Gc1=Kc1.*(1+(1)./tauI/s)
Gc2=Kc2.*(1+(1)./tauI/s+tauD*s./(0.1*tauD*s+1));//PID with derivative filtering
```

```
G_CL1=syslin('c',Gc1*G./(1+Gc1*G));
G_CL2=syslin('c',Gc2*G./(1+Gc2*G));
t=0:300;
y1=csim('step',t,G_CL1);
y2=csim('step',t,G_CL2);
y1(:,1:theta)=0;//accounting for time delay--this is required otherwise
//an unrealistic inverse response is seen due to the pade approx
y2(:,1:theta)=0;

t_d=151:300;
G_CL_dist1=syslin('c',G./(1+Gc1*G));//closed loop wrt disturbance
G_CL_dist2=syslin('c',G./(1+Gc2*G));//closed loop wrt disturbance
y_d1=csim('step',t_d,G_CL_dist1);
y_d1(:,1:theta)=0;//accounting for time delay
y_d2=csim('step',t_d,G_CL_dist2);
y_d2(:,1:theta)=0;//accounting for time delay
y1(:,t_d)=y1(:,t_d)+y_d1;
y2(:,t_d)=y2(:,t_d)+y_d2;

//plot(t,y1)
//xgrid()
//xtitle('Ex-11.3 PI control','Time(min)','y(t)');
//a=legend("$\tau_c=8$","$\tau_c=15$",position=1);
//a.font_size=2;
//a=get("current_axes");b=a.title;b.font_size=5;c=a.x_label;c.font_size=5;
//c=a.y_label;c.font_size=5;
//scf()
//
//plot(t,y2)
//xgrid()
//xtitle('Ex-11.3 PID control','Time(min)','y(t)');
//a=legend("$\tau_c=8$","$\tau_c=15$",position=1);
//a.font_size=2;
```

```
//a=get("current_axes");b=a.title;b.font_size=5;c=a.x_label;c.font_size=5;
//c=a.y_label;c.font_size=5;


mprintf('\n\nThere is uncertainty as to whether PID with derivative filtering\n...
to be used or not. Since one gets results by using PID with filtering\n...
it has been used here. Note that pade approx for delay=7.4\n...
is totally wrong because it is too gross an approx but we have no\n...
other way of making delay approx so we have to live with it.\n\n...')


//Part (b) Routh Array testing
//For frequency response refer to ch-13 for Bode Plots
G=(1-theta*s)/s;
poly_PI=Gc1*G;//denom(G_CL1);//G*Gc for PI controller
poly_PID=Gc2*G;//G*Gc for PID controller

Routh1=routh_t(poly_PI(1,1)/1,poly(0,"K")); //produces routh table for polynomial 1+Kc*poly
disp(Routh1,"Routh1=")
Kmax1=roots(numer(Routh1(1,1)));

Routh2=routh_t(poly_PI(2,1)/1,poly(0,"K")); //produces routh table for polynomial 1+Kc*poly
disp(Routh2,"Routh2=")
Kmax2=roots(numer(Routh2(1,1)));

Routh3=routh_t(poly_PID(1,1)/1,poly(0,"K")); //produces routh table for polynomial 1+Kc*poly
disp(Routh3,"Routh3=")
```

```
//Kmax3=roots(numer(Routh3(1,1)));

Routh4=routh_t(poly_PID(2,1)/1,poly(0,"K")); //
   produces routh table for polynomial 1+Kc*poly
disp(Routh4,"Routh4=")
//Kmax4=roots(numer(Routh4(1,1)));

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)
mprintf('\n\nAnswers to Kmax for PID controller
   using \n...
Routh Array in the book are wrong. This can be
   easily \n...
checked from Routh3 and Routh4 which are displayed\n
   ')
mprintf('\n\nFor frequency response refer to ch-13
   for Bode Plots\n')
```

Scilab code Exa 11.4 IMC for lag dominant model

```

clear
clc

//Example 11.4
disp('Example 11.4')

s=%s;
theta=1;tau=100;K=100;
delay=(1-theta/2*s+theta^2/10*s^2-theta^3/120*s^3)
```

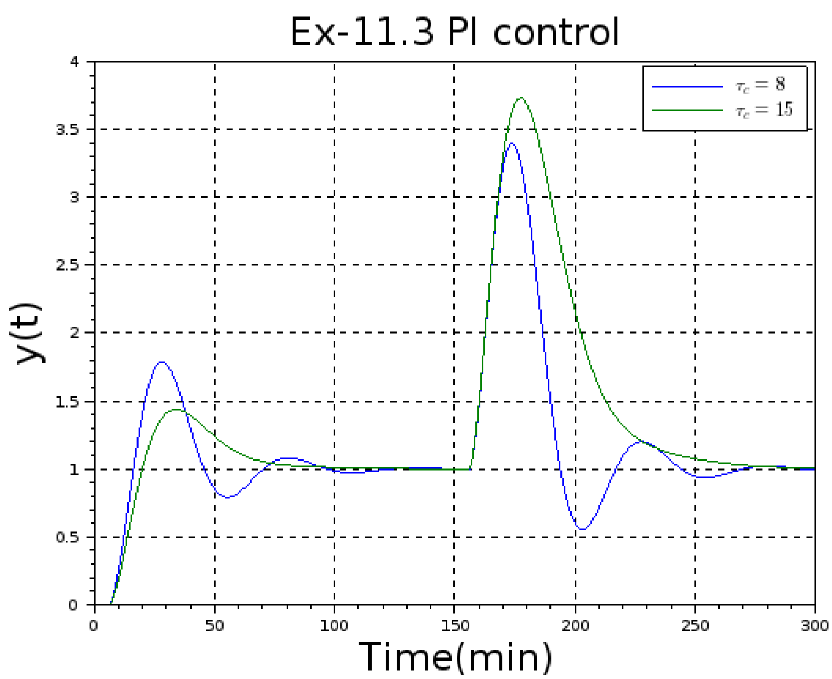


Figure 11.3: PI and PID control of liquid storage tank

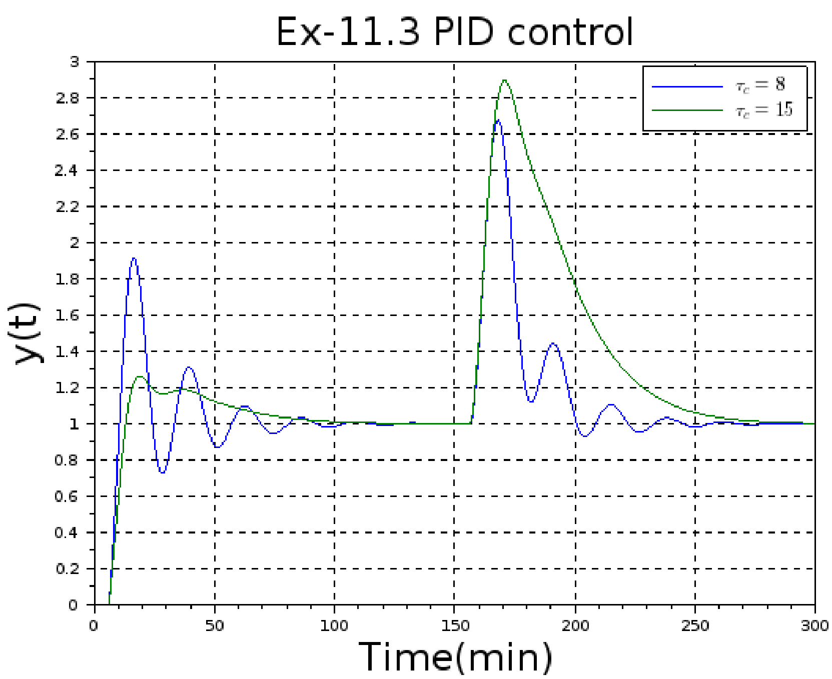


Figure 11.4: PI and PID control of liquid storage tank

```
/(1+theta/2*s+theta^2/10*s^2+theta^3/120*s^3);//Third order pade approx
G=K*delay/(tau*s+1);

//(a)
tauca=1;
Kc1=1/K*tau/(tauca+theta);taui1=tau;
//(b)
taucb=2;Kstar=K/tau;
Kc2=1/Kstar*(2*taucb+theta)./(taucb+theta).^2; //Row M
taui2=2*taucb+theta;
//(c)
taucc=1;
Kc3=Kc1;taui3=min(taui1,4*(taucc+theta))
//(d)
//Kc4=0.551;taui4=4.91;
//Chen and Seborg settings given in Second Edition of book

mprintf('                          Kc              tauI')
mprintf('\nIMC     %20f      %f',Kc1,taui1)
mprintf('\nIntegrator approx    %-5f     %f',Kc2,taui2)
mprintf('\nSkogestad    %15f      %f',Kc3,taui3)
//mprintf('\nDS-d    %20f      %f',Kc4,taui4)



Gc=[Kc1 Kc2 Kc3]'.*(1+(1)./([taui1 taui2 taui3]'*s))

G_CL=syslin('c',Gc*G./(1+Gc*G));
t=0:0.1:20;
y=csim('step',t,G_CL);
y(:,1:theta/0.1)=0;//accounting for time delay--this is required otherwise
//an unrealistic inverse response is seen due to the pade/taylor approx
```

```
plot(t,y);
xgrid()
xtitle('Ex-11.4 Tracking problem','Time(min)','y(t)');
a=legend("IMC","Integrator approx","Skogestad",position=4);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=5;c=a.x_label;c.font_size=5;
c=a.y_label;c.font_size=5;

scf()
t=0:0.1:60;
G_CL_dist=syslin('c',G./(1+Gc*G));//closed loop wrt disturbance
yd=csim('step',t,G_CL_dist);
yd(:,1:theta/0.1)=0;//accounting for time delay--this is required otherwise
//an unrealistic inverse response is seen due to the pade/taylor approx
plot(t,yd);

xgrid()
xtitle('Ex-11.4 Disturbance rejection','Time(min)','y(t)');
a=legend("IMC","Integrator approx","Skogestad",position=4);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=5;c=a.x_label;c.font_size=5;
c=a.y_label;c.font_size=5;
```

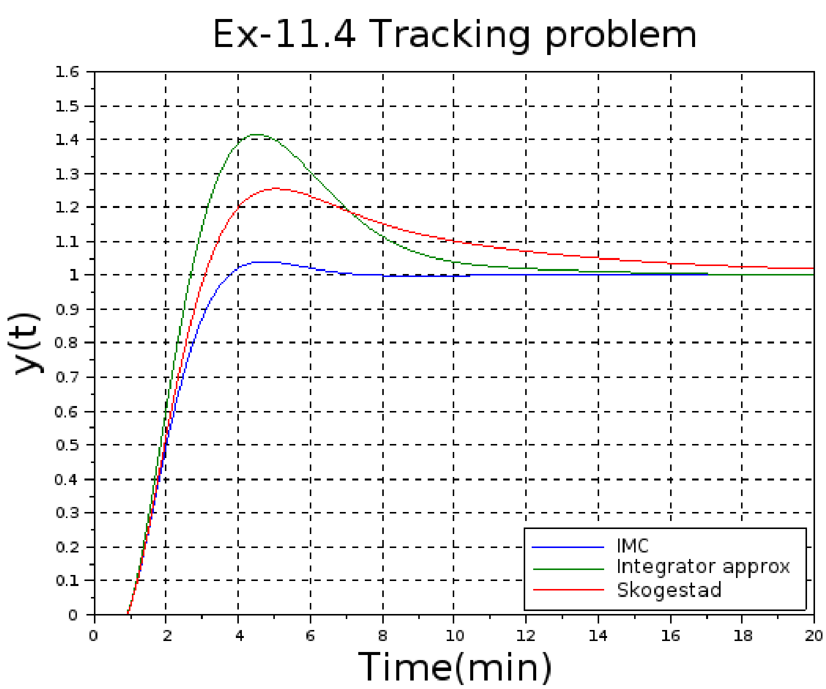


Figure 11.5: IMC for lag dominant model

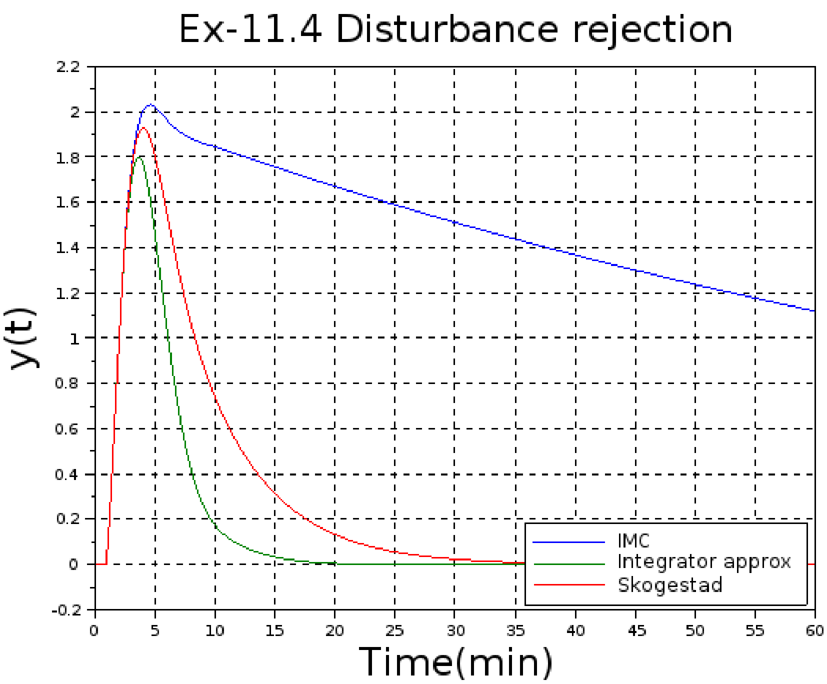


Figure 11.6: IMC for lag dominant model

Scilab code Exa 11.5 PI controller IMC ITAE

```
clear
clc

//Example 11.5
disp('Example 11.5')

K=1.54;theta=1.07;tau=5.93;


//(a)
tauca=tau/3;
Kc1=1/K*tau/(tauca+theta);taui1=tau; //Table 11.1
//(b)
taucb=theta;
Kc2=1/K*tau/(taucb+theta);taui2=tau; //Table 11.1
//(c)
//Table 11.3
Y=0.859*(theta/tau)^(-0.977);Kc3=Y/K;
taui3=tau*inv(0.674*(theta/tau)^-0.680);
//(d)
//Table 11.3
Kc4=1/K*0.586*(theta/tau)^-0.916;taui4=tau*inv(1.03-0.165*(theta/tau));

mprintf('                        Kc            tauI')
mprintf('\nIMC(tauC=tau/3)     %f      %f',Kc1,taui1)
mprintf('\nIMC(tauC=theta)     %-5f      %f',Kc2,taui2)
mprintf('\nITAE(disturbance)   %f      %f',Kc3,taui3)
mprintf('\nITAE(set point)   %10f      %f',Kc4,taui4)
```

Scilab code Exa 11.6 Controller with two degrees of freedom

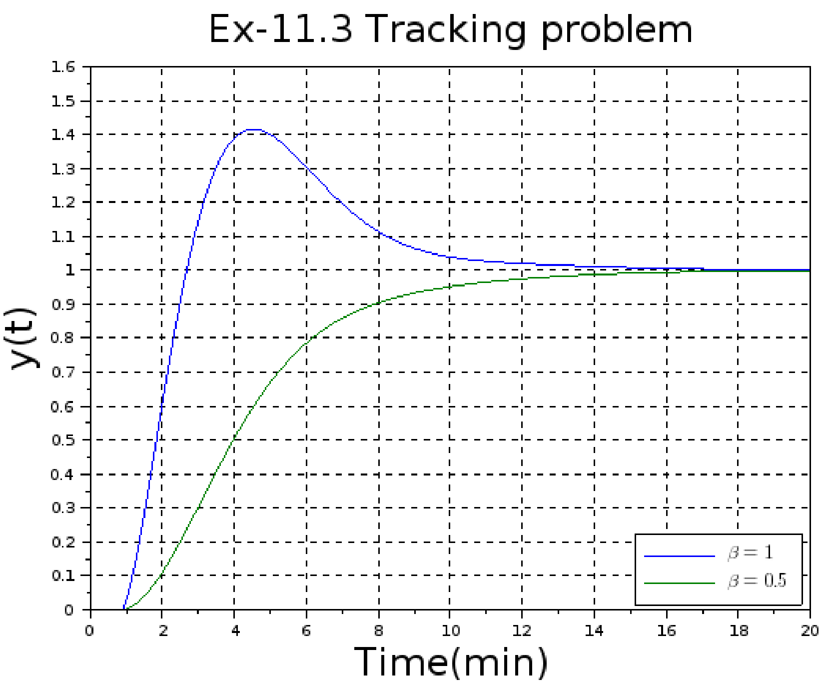


Figure 11.7: Controller with two degrees of freedom

```
clear
clc

//Example 11.6
disp('Example 11.6')

//Drawing on example 11.4
s=%s;
theta=1;tau=100;K=100;
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
G=K*delay/(tau*s+1);

Kc=0.556;taui=5;

Gc=Kc.*(1+(1)./([taui]*s))
G_CL=syslin('c',Gc*G./(1+Gc*G));
t=0:0.1:20;
y1=csim('step',t,G_CL);
y1(:,1:theta/0.1)=0;//accounting for time delay--this is required otherwise
//an unrealistic inverse response is seen due to the pade/taylor approx

beta=0.5;
G_CL2=syslin('c',(Gc+beta-1)*G./(1+Gc*G));//This can be obtained on taking
//laplace transform of eqn 11-39 and making a block diagram
//In Eqn 11-39 p refers to input to the process
t=0:0.1:20;
y2=csim('step',t,G_CL2);
y2(:,1:theta/0.1)=0;//accounting for time delay--this is required otherwise
//an unrealistic inverse response is seen due to the pade/taylor approx

```

```
plot(t,[y1; y2]);
xgrid()
xtitle('Ex-11.3 Tracking problem','Time(min)','y(t)'
    );
a=legend("$\beta=1$","$\beta=0.5$",position=4);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
c=a.y_label;c.font_size=5;

//Note that there is a slight mis-match between the
    plots obtained from scilab code
//and that of the book because of third order pade
    approximation
//The plots in the book have been produced using
    advanced proprietary software
//which supports using exact delays while scilab
    does not have that functionality
```

Scilab code Exa 11.7 Continuous cycling method

```
clear
clc

//Example 11.7
disp('Example 11.7')

K=2;theta=1;tau1=10;tau2=5; //Model parameters

s=%s;
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
G=K*delay/((tau1*s+1)*(tau2*s+1));
Ku=[8.01]';//Trials for various values of Ku can be
```

```
done by changing Ku
G_CL_trial=syslin('c',G*Ku./(1+G*Ku))
t=0:0.1:100;
y=csim('step',t,G_CL_trial);
plot(t,y)
xtitle('Ex-11.7 Finding ultimate gain Ku','Time(min)','y(t)');
a=legend("Closed loop test",position=4);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=5;c=a.x_label;c.font_size=5;
c=a.y_label;c.font_size=5;
//There isnot a sustained oscillation for Ku=7.88, in our simulation because
//we are using a third order Pade Approx for delay. But still we go ahead with it
//so that it matches with the example values. Our simulations give Ku=8
Ku=7.88;Pu=11.66;


//(a) Table 11.4 ZN
Kc1=0.6*Ku;taui1=Pu/2;tauD1=Pu/8;
//(b) Table 11.4 TL
Kc2=0.45*Ku;taui2=Pu*2.2;tauD2=Pu/6.3;
//(c) DS method
tauc=3;
Kc3=1/K*(tau1+tau2)/(tauc+theta);taui3=tau1+tau2;tauD3=tau1*tau2/(tau1+tau2);

mprintf('          Kc            tauI          tauD')
mprintf('\nZN    %f      %f        %f',Kc1,taui1,tauD1)
mprintf('\nTL    %f      %f        %f',Kc2,taui2,tauD2)
mprintf('\nDS    %f      %f        %f',Kc3,taui3,tauD3)


// Now we compare the performance of each controller setting
```

```
Gc1=Kc1.*(1+(1)./taui1/s+tauD1*s)
Gc2=Kc2.*(1+(1)./taui2/s+tauD2*s)
Gc3=Kc3.*(1+(1)./taui3/s+tauD3*s)
Gc=[Gc1 Gc2 Gc3]';
G_CL=syslin('c',Gc*G./(1+Gc*G));
t=0:160;
y=csim('step',t,G_CL);
y(:,1:theta)=0;//accounting for time delay--this is
    required otherwise
//an unrealistic inverse response is seen due to the
    pade/taylor approx


t_d=81:160;
G_CL_dist=syslin('c',G./(1+Gc*G));//closed loop wrt
    disturbance
yd=csim('step',t_d,G_CL_dist);
yd(:,1:theta)=0;//accounting for time delay
y(:,t_d)=y(:,t_d)+yd;

scf();
plot(t,y)
xgrid()
xtitle('Ex-11.7 Comparison of 3 controllers','Time(
    min)','y(t)');
a=legend("Ziegler-Nichols","Tyerus-Luyben","Direct
    Synthesis",position=4);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=5;c=a.
    x_label;c.font_size=5;
c=a.y_label;c.font_size=5;

mprintf('\n\nThere is slight mismatch between scilab
    simulation\n...
and book simulation due to Pade approx\n')
```

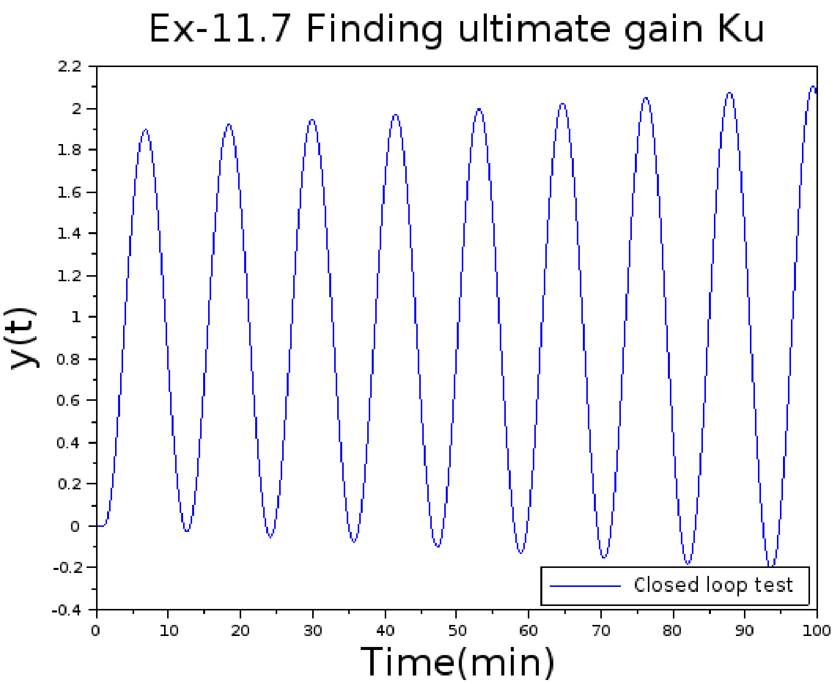


Figure 11.8: Continuous cycling method

Scilab code Exa 11.8 Reaction curve method

```
clear
clc

//Example 11.8
disp('Example 11.8')

```

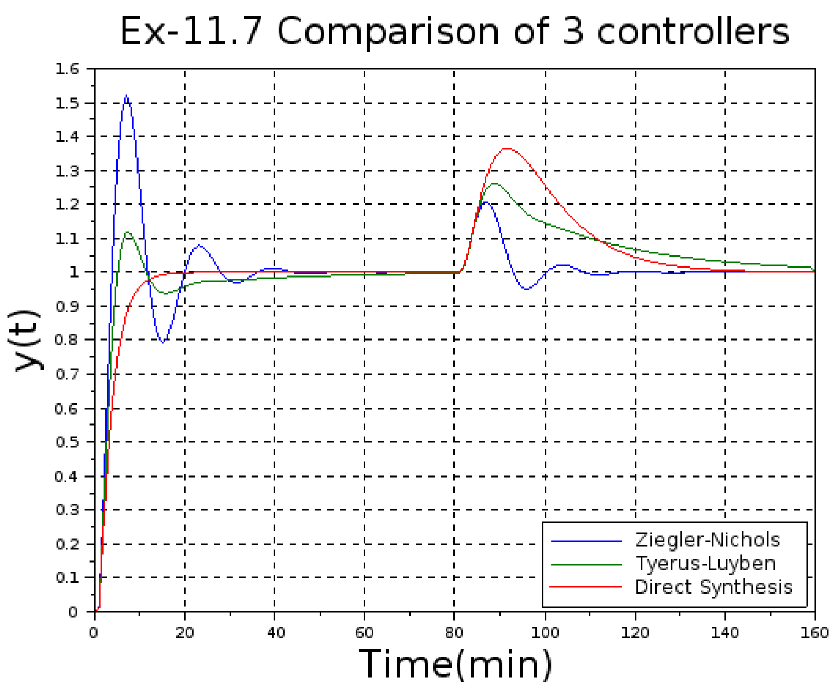


Figure 11.9: Continuous cycling method

```
S=(55-35)/(7-1.07);//%/min
delta_p=43-30;//%
R=S/delta_p;//min^-1

delta_x=55-35;//%
K=delta_x/delta_p;
theta=1.07;//min
tau=7-theta;//min

mprintf("\nThe resulting process model is with delay of 1.07 min\n")
s=%s;
G=K/(tau*s+1);
disp(G,'G=')
```

Chapter 12

Control Strategies at the Process Unit Level

Scilab code Exa 12.1 Degrees of freedom

```
clear
clc

//Example 12.1
disp('Example 12.1')

NE=3;
NV=6;
NF=NV-NE;
ND=2;
NFC=NF-ND;
mprintf("    NF=%i\n    NFC=%i",NF,NFC)
```

Chapter 13

Frequency response analysis and control system design

Scilab code Exa 13.3 Bode Plot

```
clear
clc

//Example 13.3
disp('Example 13.3')

function bodegen(num,den,w,lf,delay)
//Bode plot
//Numerator and denominator are passed as input
   arguments
//Both are polynomials in powers of s(say)

//This function has been modified from the original
   one
//written by Prof Kannan Moudgalya, Dept of ChemE,
   IIT-B
G = num/den;
```

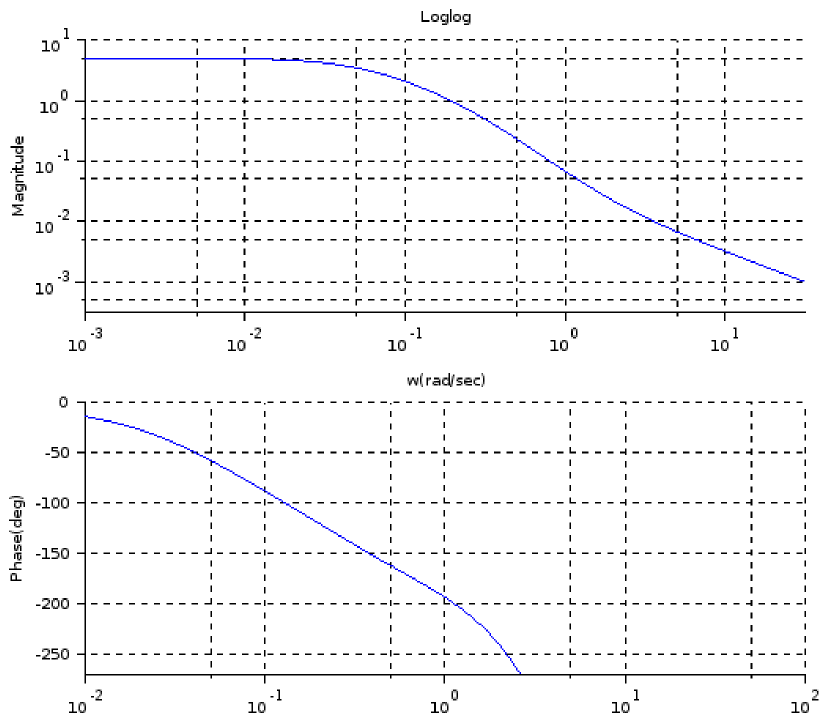


Figure 13.1: Bode Plot

```
G1 = horner(G,%i*w);
G1p = phasemag(G1)-delay*w*180/%pi;

if LF == "normal" then
   xset('window',0); clf();
   subplot(2,1,1)
   plot2d(w,abs(G1),logflag="nn",style = 2);
   xtitle('Normal scale','','Magnitude'); xgrid();
   subplot(2,1,2)
   plot2d1(w,G1p,logflag="nn",style = 2);
   xgrid();
   xtitle('w(rad/sec)','','Phase(deg)');
elseif LF == "semilog" then
   xset('window',1); clf();
   subplot(2,1,1)
   plot2d(w,20*log10(abs(G1)),logflag="ln",style = 2);
   xgrid();
   xtitle('Semilog','','Magnitude (dB)');
   subplot(2,1,2)
   plot2d1(w,G1p,logflag="ln",style = 2);
   xgrid();
   xtitle('w(rad/sec)','','Phase(deg)');
elseif LF == "loglog" then
   xset('window',2); clf();
   subplot(2,1,1)
   plot2d(w,abs(G1),logflag="ll",style = 2);
   xgrid();
   xtitle('Loglog','','Magnitude');
   subplot(2,1,2)
   plot2d1(w,G1p,logflag="ln",style = 2,rect=[0.01,-270,100,0]);//note the usage of rect for this particular example
   xgrid();
   xtitle('w(rad/sec)','','Phase(deg)');
end
endfunction;

```

```
s = %s;
num = 5*(0.5*s+1);
den = (20*s+1)*(4*s+1);
theta=1;

w = 0.001:0.002:10*%pi;
LF = "loglog" // Warning: Change this as necessary

bodegen(num,den,w,LF,theta);

//Checking using iodelay toolbox in scilab
//G=syslin('c',num/den);
//G=iodelay(G,1)
//bode(G,0.01,0.1)
```

Scilab code Exa 13.4 Bode

```
clear
clc

//Example 13.4
disp('Example 13.4')


s = %s;
num = 2;
den = (0.5*s+1)^3;
delay=0;
w = 0.001:0.002:100*%pi;
LF = "loglog" // Warning: Change this as necessary


```

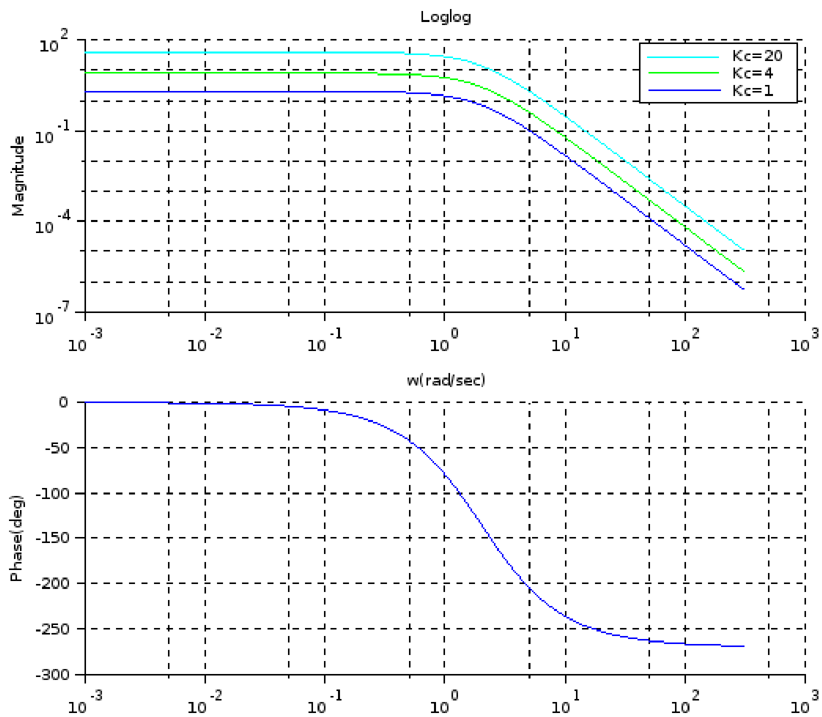


Figure 13.2: Bode

```
//Kc=1
G1 = num/den;
G1m = horner(G1,%i*w); //G1m denotes magnitude
G1p = phasemag(G1m)-delay*w*180/%pi; //G1p denotes
   phase

//Kc=4
G2 = 4*num/den;
G2m = horner(G2,%i*w);
G2p = phasemag(G2m)-delay*w*180/%pi;

//Kc=20
G3 = 20*num/den;
G3m = horner(G3,%i*w);
G3p = phasemag(G3m)-delay*w*180/%pi;

   xset('window',0);
   subplot(2,1,1)
   plot2d(w,abs(G3m),logflag="ll",style = 4);
   plot2d(w,abs(G2m),logflag="ll",style = 3);
   plot2d(w,abs(G1m),logflag="ll",style = 2);

   xgrid();
   xtitle('Loglog','','Magnitude');
   legend("Kc=20","Kc=4","Kc=1")
   subplot(2,1,2)
   plot2d1(w,G1p,logflag="ln",style = 2);
   xgrid();
   xtitle('w(rad/sec)','','Phase(deg)');
```

Scilab code Exa 13.5 PI control of overdamped second order process

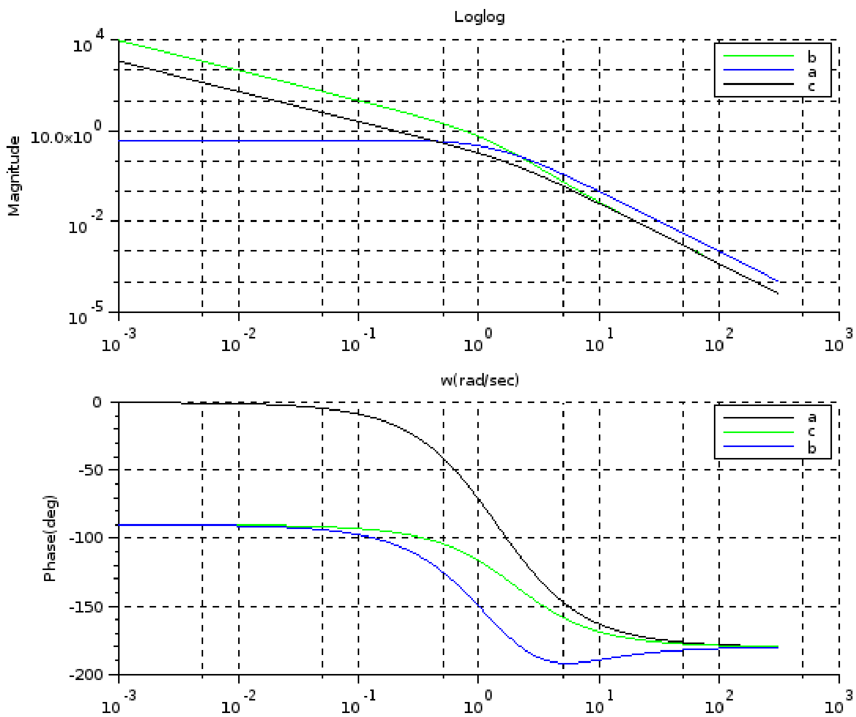


Figure 13.3: PI control of overdamped second order process

```
clear
clc

//Example 14.5
disp('Example 14.5')


s = %s;
num = 1;
den = (9*s+1)*(11*s+1);
delay=0.3;
w = 0.001:0.002:5*%pi;
LF = "loglog" // Warning: Change this as necessary

Gc=20*(1+1/2.5/s+s);
G1 = num/den*Gc;
G1m = horner(G1,%i*w); //G1m denotes magnitude
G1p = phasemag(G1m)-delay*w*180/%pi; //G1p denotes
   phase

   xset('window',0);
   subplot(2,1,1)
   plot2d(w,abs(G1m),logflag="ll",style = 3);
   xgrid();
   xtitle('Loglog','','Magnitude');
   subplot(2,1,2)
   plot2d1(w,G1p,logflag="ln",style = 1);
   xgrid();
   xtitle('w(rad/sec)','','Phase(deg)');
```

Scilab code Exa 13.6 Bode plot

```
clear
```

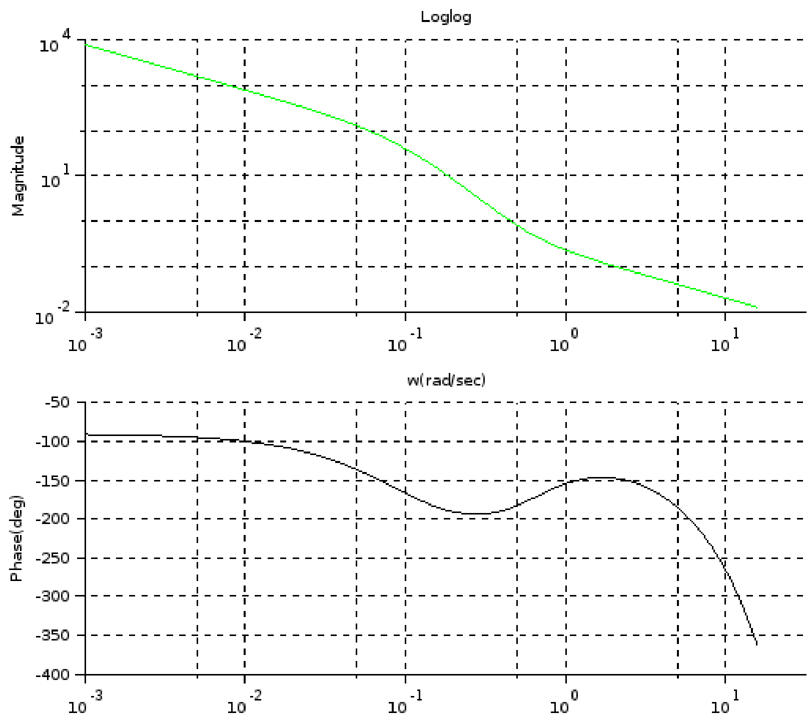


Figure 13.4: Bode plot

```
clc

//Example 13.6
disp('Example 13.6')


s = %s;
num = 1;
den = (9*s+1)*(11*s+1);
delay=0.3;
w = 0.001:0.002:5*%pi;
LF = "loglog" // Warning: Change this as necessary

Gc=20*(1+1/2.5/s+s);
G1 = num/den*Gc;
G1m = horner(G1,%i*w); //G1m denotes magnitude
G1p = phasemag(G1m)-delay*w*180/%pi; //G1p denotes
   phase

   xset('window',0);
   subplot(2,1,1)
   plot2d(w,abs(G1m),logflag="ll",style = 3);
   xgrid();
   xtitle('Loglog','','Magnitude');
   subplot(2,1,2)
   plot2d1(w,G1p,logflag="ln",style = 1);
   xgrid();
   xtitle('w(rad/sec)','','Phase(deg)');
```

Scilab code Exa 13.7 Bode Plot

```
clear
clc
```

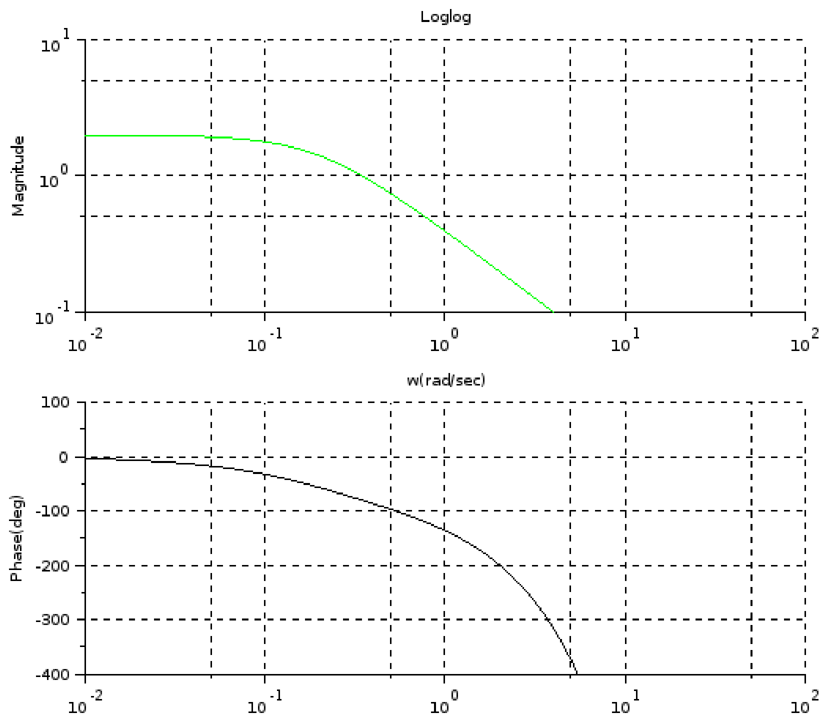


Figure 13.5: Bode Plot

```

//Example 13.7
disp('Example 13.7')


s = %s;
num = 4;
den = (5*s+1);
delay=1;
w = 0.001:0.002:10*%pi;
LF = "loglog" // Warning: Change this as necessary

Gv=2;Gm=0.25;Gc=1;
G1 = num/den*Gc*Gm*Gv;
G1m = horner(G1,%i*w); //G1m denotes magnitude
G1p = phasemag(G1m)-delay*w*180/%pi; //G1p denotes phase

   xset('window',0);
   subplot(2,1,1)
   plot2d(w,abs(G1m),logflag="ll",style = 3,rect=[0.01,0.1,100,10]);
   xgrid();
   xtitle('Loglog','','Magnitude');
   subplot(2,1,2)
   plot2d1(w,G1p,logflag="ln",style = 1,rect=[0.01,-400,100,100]);
   xgrid();
   xtitle('w(rad/sec)','','Phase(deg)');

//Example ends

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

```
the code below

////Please install IODELAY toolbox from Modeling and Control tools in ATOMS
////http://atoms.scilab.org/toolboxes/iodelay/0.4.5
////There is no inbuilt toolbox in scilab for introducing time delays other than
////above mentioned. The output of iodelay toolbox however does not work
////with csim and syslin commands
////The output of iodelay however can be used for frequency related analyses
////like bode and nyquist

//xset('window',1);
//G=syslin('c',G1);
//G=iodelay(G,delay);
//nyquist(G,%f);   //%f => asymmetric, see help nyquist
////************************************************************
```

Scilab code Exa 13.8 Maximum permissible time delay for stability

```
clear
clc

//Example 13.8
disp('Example 13.8')


s = %s;
```

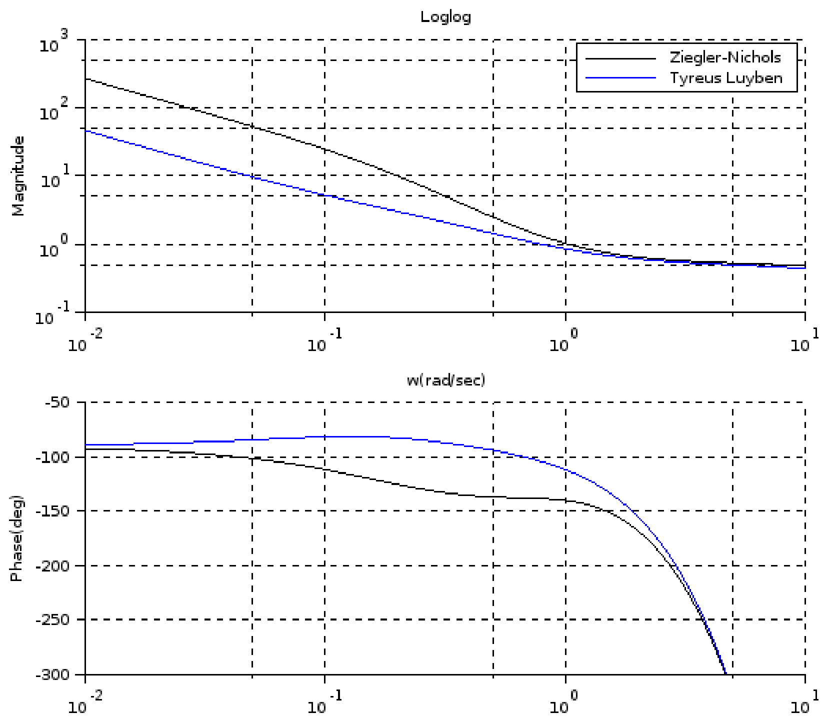


Figure 13.6: Maximum permissible time delay for stability

```
num = 4;
den = (5*s+1);
delay=1;
w = 0.001:0.002:10*%pi;
LF = "loglog" // Warning: Change this as necessary

Gv=2;Gm=0.25;

Ku=4.25;Pu=2*%pi/1.69;

//Ziegler Nichols
Kc1=0.6*Ku;taui1=Pu/2;tauD1=Pu/8;
//Tyreus Luyben
Kc2=0.45*Ku;taui2=Pu*2.2;tauD2=Pu/6.3;

mprintf('         Kc             tauI          tauD')
mprintf('\nZN    %f      %f        %f',Kc1,taui1,tauD1)
mprintf('\nTL    %f      %f        %f',Kc2,taui2,tauD2)

Gc_ZN=Kc1*(1+1/taui1/s+s*tauD1/(0.1*s*tauD1+1));
Gc_TL=Kc2*(1+1/taui2/s+s*tauD2/(0.1*s*tauD2+1)); //
   Filtered Controllers with filter constant as 0.1

G1 = num/den*Gc_ZN*Gm*Gv;
G1m = horner(G1,%i*w); //G1m denotes magnitude
Abs_G1m=abs(G1m)
G1p = phasemag(G1m)-delay*w*180/%pi; //G1p denotes
   phase


G2 = num/den*Gc_TL*Gm*Gv;
G2m = horner(G2,%i*w); //G1m denotes magnitude
Abs_G2m=abs(G2m);
G2p = phasemag(G2m)-delay*w*180/%pi; //G1p denotes
   phase

   xset('window',0);
   subplot(2,1,1)
```

```
    plot2d(w,Abs_G1m,logflag="ll",style = 1,rect=[0.01,0.1,10,1000]);
    plot2d(w,Abs_G2m,logflag="ll",style = 2,rect=[0.01,0.1,10,1000]);
    legend("Ziegler-Nichols","Tyreus Luyben")
    xgrid();
    xtitle('Loglog','','Magnitude');
    subplot(2,1,2)
    plot2d1(w,G1p,logflag="ln",style = 1,rect=[0.01,-300,10,-50]);
    plot2d1(w,G2p,logflag="ln",style = 2,rect=[0.01,-300,10,-50]);
//      legend("Ziegler-Nichols","Tyreus Luyben")
    xgrid();
    xtitle('w(rad/sec)','','Phase(deg)');

//G_ZN=syslin('c',G1);
//G_ZN=iodelay(G_ZN,delay);
//G_TS=syslin('c',G2);
//G_TS=iodelay(G_TS,delay);
//scf();nyquist(G_TS,%f)
//[gm_ZN,fr_ZN]=g_margin(G_ZN);[gm_TS,fr_TS]=g_margin(G_TS);
//[pm_ZN,fr_ZN_p]=p_margin(G_ZN);[pm_ZN,fr_ZN_p]=p_margin(G_TS);
//g_maring and p_margin do not support iodelay toolbox, hence we
//cannot use these so we try a workaround

//We can find w for which magnitude(AR) is 1 and
//and calculate phase corresponding to it which gives us phase margin
//Also we can find crossover frequency and thus find Gain Margin

     indices1=find(abs(Abs_G1m-1)<0.01) //We find those values of indices of Abs_G1m for which it is almost 1
```

```
    indices2=find(abs(Abs_G2m-1)<0.01)
    //size(indices)
    PM1=mean(G1p(indices1))+180
    PM2=mean(G2p(indices2))+180

    indices1_p=find(abs(G1p+180)<0.05) //We find those values of indices of G1p for which it is almost -180
    indices2_p=find(abs(G2p+180)<0.05)
    //size(indices)
    GM1=1/mean(Abs_G1m(indices1_p))
    GM2=1/mean(Abs_G2m(indices2_p))

    wc1=mean(w(indices1_p));
    wc2=mean(w(indices2_p));


mprintf('\n\n\n          GM            PM          wc')
mprintf('\nZN    %f      %f         %f',GM1,PM1,wc1)
mprintf('\nTL    %f      %f         %f\n',GM2,PM2,wc2)

theta=PM2*%pi/0.79/180;
disp(theta,'deltatheta(min)=')
```

Chapter 14

Feedforward and Ratio Control

Scilab code Exa 14.1 Ratio control

```
clear
clc

//Example 14.1
disp('Example 14.1')

Sd=30;
Su=15;
Rd=1/3;
K_R=Rd*Sd/Su;   //Eqn 14-3
mprintf("    K_R=%f",K_R)
```

Scilab code Exa 14.5 Feedforward control in blending process

```
clear
clc

```

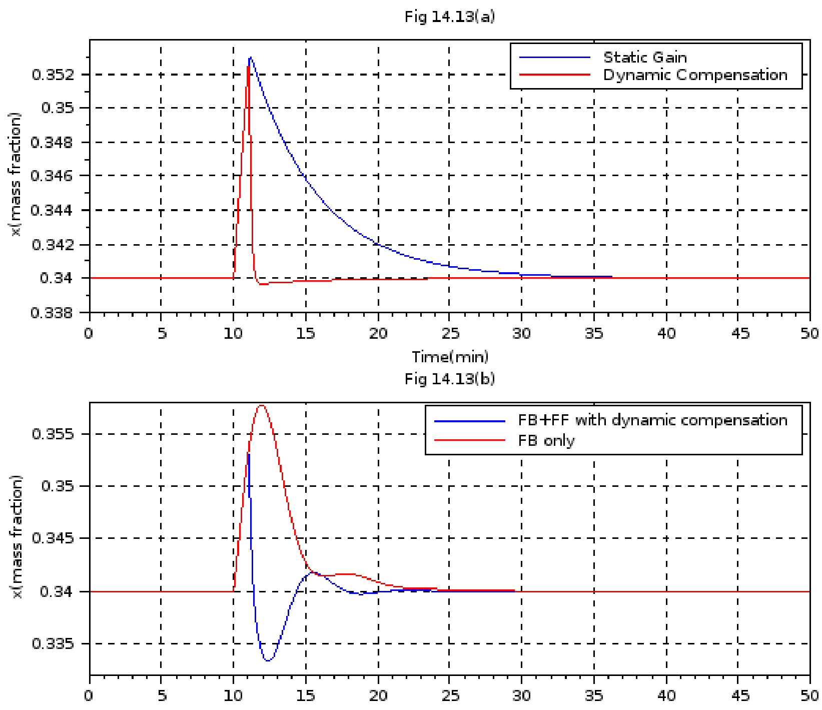


Figure 14.1: Feedforward control in blending process

```
//Example 14.5
disp('Example 14.5')

mprintf('\n\nThere are many errors in this example\n...
1.In Eqn 14-17 the value of w2_o is not zero. It is 50kg/min.\n...
This is so because otherwise current signal from p(t) ie ...
\n eqn 14-30 is more than 20mA which is not possible\n\n....
2.The step change in x1 is from 0.2 to 0.3 and not 0.2 to 0.4\n...
If there is a step change to x1=0.4, then with x2=0.6\n...
one cannot achieve output xsp=0.34 because it is less\n...
both x1 and x2.\n\n...
3.The gain of Gd is 0.65 which is correct because V\n...
has to be calculated using height=1.5meter ie\n...
how much the CSTR is filled and not h=3m, ie\n...
the capacity of CSTR. This is important because\n...
the person who has made solutions for the book has taken h=3m\n...
for generating graphs and hence the gain is twice. \n...
the graphs generated from this code are correct\n\n')

//part(a) //========Static feedforward controller===========//
K_IP=(15-3)/(20-4);
Kv=300/12;tauV=0.0833;
Kt=(20-4)/0.5;
w2_o=50;x1_o=0;//Zero of the instrument
w1bar=650;w2bar=350;//kg/min
C1=4-w2_o/Kv/K_IP; //Eqn 14-16 to 14-19
```

```
C2=w1bar/(Kv*K_IP*Kt);
C3=4+Kt*x1_o;
x1bar=0.2;x2bar=0.6;xbar=0.34;

mprintf('\nThe values of C1, C2, C3 in Eqns 14-16 to 14-19 are\n  %f,        %f,         %f',C1,C2,C3)

//part(b) //=======Dynamic feedforward controller=======//
s=%s;
theta=1;
V=%pi*1^2*1.5; //pi*r^2*h finding volume
rho=1000;//kg/m3
wbar=w1bar+w2bar;
tauD=V*rho/w2bar;tauP=V*rho/wbar;
Kp=(x2bar-xbar)/wbar;
Kd=w1bar/wbar;

Gv=Kv/(tauV*s+1);
Gd=Kd/(tauP*s+1);
Gt=Kt;delay=1;
Gp=Kp/(tauP*s+1);
Gf=-Gd/Gv/Gt/Gp/K_IP; //Equation 14-33 without exp(+s)
//Gt=32*(1-theta/2*s+theta^2/12*s^2)/(1+theta/2*s+theta^2/12*s^2);//second order Pade approx.
Gt=32*(1-theta/2*s)/(1+theta/2*s);//first order Pade approx.
alpha=0.1;
Gf=horner(Gf,0)*(1.0833*s+1)/(alpha*1.0833*s+1);//Eqn 14-34
disp(Gf,"Gf=")


//========Static feedforward controller simulation===========//
Ts=0.01;//sampling time in minutes
t=Ts:Ts:50;
```

```
xsp=0.34;//set point for conc. output of blender
x1step=0.2+[zeros(1,length(t)/5) 0.1*ones(1,length(t)*4/5)];//disturbance
//There is a one second lag in the measurement of the disturbance by Gt

delay=1;
d=[0.2*ones(1,delay/Ts) x1step(1,1:$-delay/Ts)];//measurement of disturbance with lag
x1m=4+Kt*d; //Eqn 14-12 where d=x1(t)-x1_o

//plot(t,[x1step' x1m'])
pt=C1+C2*(Kt*xsp-x1m+C3)/(x2bar-xsp);
//Now the values calculated by the above controller needs to be passed to the process
G_static=syslin('c',[Gd K_IP*Gv*Gp]);
//we take disturbance and control action in deviation variables
yt=0.34+csim([x1step-x1step(1,1);pt-pt(1,1)],t,G_static);
subplot(2,1,1)
plot(t,yt);
xtitle("Fig 14.13(a)","Time(min)","x(mass fraction)")
xgrid();

//==========Dynamic feedforward controller simulation======//

Ys_Ds=[Gd K_IP*32*Gf*Gv*Gp]; //Gt=32 without delay
Ys_Ds=syslin('c',Ys_Ds);
t=0.01:0.01:50;
d=[zeros(1,length(t)/5) 0.1*ones(1,length(t)*4/5)];//disturbance
d_shift=[zeros(1,1.1*length(t)/5) 0.1*ones(1,length(t)*3.9/5)];
//we delay the disturbance by one minute for the feedforward controller
```

```
//We do this because Pade approx is not good for delay of 1 minute
yt=0.34+csim([d;d_shift],t,Ys_Ds)
plot(t,yt,color='red')
legend("Static Gain","Dynamic Compensation")

//part(c)   //=======PI controller for Feedback ======//
Kcu=48.7;Pu=4;//min
Kc=0.45*Kcu;tauI=Pu/1.2;tauD=0;
Gc=Kc*(1+1/(tauI*s)+tauD*s/(1+tauD*s*0.1));
Gm=Gt;//sensor/transmitter


//==========Feedforward and feedback control with dynamic compensation======//
Ys_Ds=[Gd K_IP*32*Gf*Gv*Gp]/(1+K_IP*Gc*Gv*Gp*Gm);//32 is magnitude of Gt
Ys_Ds=syslin('c',Ys_Ds);
t=0.01:0.01:50;
d=[zeros(1,length(t)/5) 0.1*ones(1,length(t)*4/5)];//disturbance
yt=0.34+csim([d;d_shift],t,Ys_Ds)
//This shifting is better because Pade approx is not accurate. Note that there is
//pade approx in the denominator also(Gm) which we cant help.
subplot(2,1,2)
plot(t,yt)
xgrid();
xtitle("Fig 14.13(b)","Time(min)","x(mass fraction)")

//==========Feedback control only with dynamic compensation======//
Ys_Ds=(Gd)/(1+K_IP*Gc*Gv*Gp*Gm);
Ys_Ds=syslin('c',Ys_Ds);
d=[zeros(1,length(t)/5) 0.1*ones(1,length(t)*4/5)];
```

```
//disturbance
yt=0.34+csim(d,t,Ys_Ds)
plot(t,yt,color='red')
legend("FB+FF with dynamic compensation","FB only")

mprintf("\n\nNote the slight mismatch between response \n...
times due to pade approx the gain is half of that in the\n...
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

```
clear
clc

//Example 15.1
disp('Example 15.1')


s=%s;
Gp1=4/((4*s+1)*(2*s+1));Gp2=1;Gd2=1;Gd1=1/(3*s+1);
Gm1=0.05;Gm2=0.2;
Gv=5/(s+1);
Kc2=4;
Ys=Kc2*Gv*Gp1*Gm1/(1+Kc2*Gv*Gm2);

Routh=routh_t(Ys,poly(0,"Kc1")); // produces routh table for polynomial 1+Kc*Ys
disp(Routh)
K1=roots(numer(Routh(3,1)));
K2=roots(numer(Routh(4,1)));
```

```

mprintf('\n Kc1 lies between %f and %f \n for cascade system to be stable', K2,K1)

Ys2=Gv*Gp2*Gp1*Gm1;
Routh2=routh_t(Ys2,poly(0,"Kc1")); // produces routh table for polynomial 1+Kc*Ys
disp(Routh2)
K1_2=roots(numer(Routh2(3,1)));
K2_2=roots(numer(Routh2(4,1)));

mprintf('\n Kc1 lies between %f and %f \n for conventional system to be stable', K2_2,K1_2)



//We cannot find offset symbolically in Scilab because scilab does not support
//handling of two variables in single polynomial
//To find this limit one can use Sage
//However in this case the calculations can be done in a very easy way by hand
//and hence do not require to be computed from Sage
```

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

```
clear
clc

//Example 15.2
disp('Example 15.2')

s = %s;
```

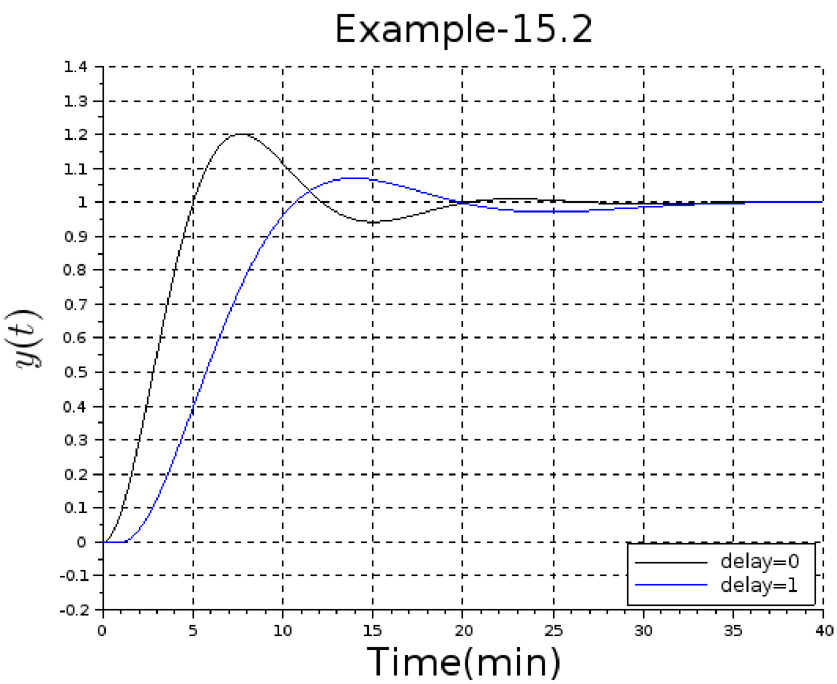


Figure 15.1: Set point response for second order transfer function

```
theta=1 //delay
delay=(1-theta/2*s+theta^2/12*s^2)/(1+theta/2*s+theta^2/12*s^2);//Second order pade approx
G=1/((5*s+1)*(3*s+1));
Gp=[G;delay*G];//Both models with and without delay
Gc=[3.02*(1+1/(6.5*s));1.23*(1+1/(7*s))];
G_CL=syslin('c',(Gp.*Gc)./(1+Gp.*Gc))
t=0:0.01:40;
yt=csim('step',t,G_CL)

plot2d(t',yt') //For plotting multiple graphs in one command make sure time is n*1 vector
//while yt is n*p vector where p are the no. of plots
xtitle('Example-15.2','Time(min)','$y(t)$');
xgrid();
a=legend("delay=0","delay=1",position=4);
a.font_size=2;
a=get("current_axes");b=a.title;b.font_size=5;c=a.x_label;c.font_size=5;
c=a.y_label;c.font_size=5;
```

Chapter 16

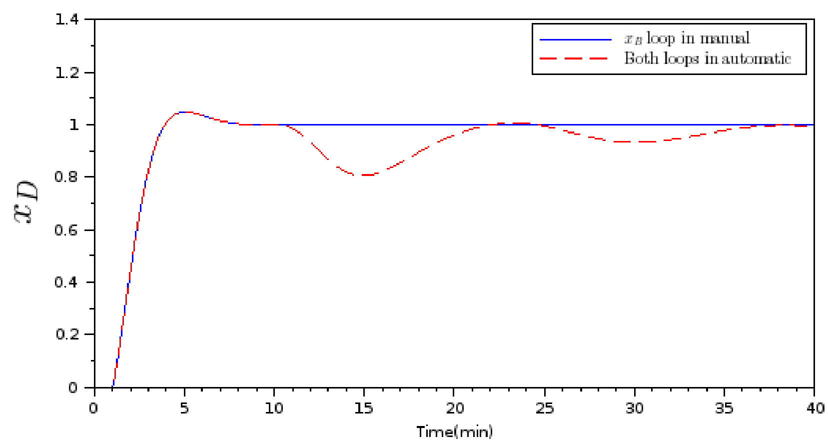
Multiloop and Multivariable Control

Scilab code Exa 16.1 Pilot scale distillation column

```
clear
clc

//Example 16.1
disp('Example 16.1')
K=[12.8 -18.9;6.6 -19.4];
tau=[16.7 21;10.9 14.4];
s=%s;
G=K./(1+tau*s);

//ITAE settings from Table 11.3
K1=12.8;tau1=16.7;theta1=1;K2=-19.4;tau2=14.4;theta2
    =3;
Kc1=1/K1*0.586*(theta1/tau1)^-0.916;taui1=tau1*inv
    (1.03-0.165*(theta1/tau1));
Kc2=1/K2*0.586*(theta2/tau2)^-0.916;taui2=tau2*inv
    (1.03-0.165*(theta2/tau2));
```



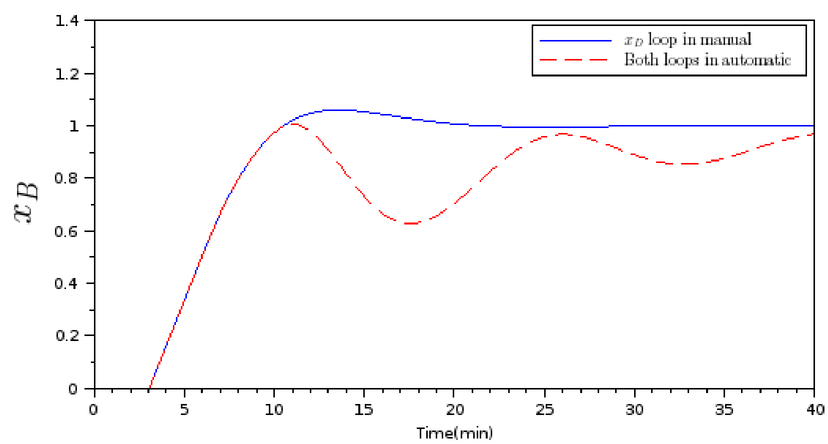


Figure 16.1: Pilot scale distillation column

```
mprintf('            Kc            tauI')
mprintf('\nx_D-R    %f     %f',Kc1,taui1)
mprintf('\nx_B-R    %f     %f',Kc2,taui2)

Kc=[Kc1;Kc2];
tauI=[taui1;taui2];

//====Making step response models of the continuos transfer functions====//
Ts=0.1;//Sampling time ie delta_T
delay3=3/Ts;
delay1=1/Ts;
delay7=7/Ts;
N=100/Ts;//Model Order
s=%s;
G=syslin('c',diag(matrix(G,1,4)));//Transfer function
t=0:Ts:N*Ts;
u_sim=ones(4,length(t));
//Modeling Output delays through input delay in steps
u_sim(1,1:(delay1))=zeros(1,delay1);
u_sim(3,1:(delay7))=zeros(1,delay7);
u_sim([2 4],1:(delay3))=zeros(2,delay3);
S=csim(u_sim,t,G)';//generating step response model for real plant
//plot(t,S);
S(1,:)=[];
//Now we have these step response models for each of the transfer functions
//[S1 S3
//S2 S4




T=120;//Simulation Run Time in minutes
```

```
n=T/Ts*2+1; //no. of discrete points in our domain of analysis

//==========================================================================//
//==========================================================================//
//==========================================================================//
//========Set point as +1 in X-D==X-B loop in manual===============//
//p is the controller output
p=zeros(n,2);
delta_p=zeros(n,2);
e=zeros(n,2); //errors=(ysp-y) on which PI acts
ysp=zeros(n,2);
ysp((n-1)/2+1:n,1)=ones(n-((n-1)/2+1)+1,1);

t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
y=zeros(n,2);


for k=(n-1)/2+1:n

    //Error  e
    e(k,:)=ysp(k-1,:)-y(k-1,:);
    delta_e(k,:)=e(k,:)-e(k-1,:);

    //Controller calculation----Digital PID----Eqn 7-28 Pg 136 (Velocity form)
    p(k,1)=p(k-1,1)+([delta_e(k,1)+e(k,1)*diag(Ts/taui1)]*diag(Kc1));
    //1-1/2-2 pairing

    delta_p(k,:)=p(k,:)-p(k-1,:);
```

```

    //Output
    y(k,1)=[S(1:N-1,1);S(1:N-1,3)]'...
        *[flipdim(delta_p(k-N+1:k-1,1),1);flipdim(delta_p(k-N+1:k-1,2),1)]...
        +[S(N,1) S(N,3)]*[p(k-N,1);p(k-N,2)];
    y(k,2)=[S(1:N-1,2);S(1:N-1,4)]'...
        *[flipdim(delta_p(k-N+1:k-1,1),1);flipdim(delta_p(k-N+1:k-1,2),1);]...
        +[S(N,2) S(N,4)]*[p(k-N,1);p(k-N,2)];
end

subplot(2,1,1);
plot(t',y(:,1),'b-');
set(gca(),"data_bounds",[0 40 0 1.4]); //putting bounds on display
l=legend("$x_B\text{ loop in manual}$",position=1);
xtitle("","Time(min)","$x_D$");
a=get("current_axes");
c=a.y_label;c.font_size=5;

//==========================================================================//
//==========================================================================//
//==========================================================================//
//========Set point as +1 in X-B==X-D loop in manual ===============//
//p is the controller output
p=zeros(n,2);
delta_p=zeros(n,2);
e=zeros(n,2); //errors=(ysp-y) on which PI acts
ysp=zeros(n,2);
ysp((n-1)/2+1:n,2)=ones(n-((n-1)/2+1)+1,1);
```

```

t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
y=zeros(n,2);


for k=(n-1)/2+1:n

    //Error  e
    e(k,:)=ysp(k-1,:)-y(k-1,:);
    delta_e(k,:)=e(k,:)-e(k-1,:);

    //Controller calculation----Digital PID----Eqn 7-28 Pg 136 (Velocity form)
    p(k,2)=p(k-1,2)+([delta_e(k,2)+e(k,2)*diag(Ts/taui2)]*diag(Kc2));
    //1-1/2-2 pairing

    delta_p(k,:)=p(k,:)-p(k-1,:);

    //Output
    y(k,1)=[S(1:N-1,1);S(1:N-1,3)]'...
        *[flipdim(delta_p(k-N+1:k-1,1),1);flipdim(delta_p(k-N+1:k-1,2),1)]...
        +[S(N,1) S(N,3)]*[p(k-N,1);p(k-N,2)];
    y(k,2)=[S(1:N-1,2);S(1:N-1,4)]'...
        *[flipdim(delta_p(k-N+1:k-1,1),1);flipdim(delta_p(k-N+1:k-1,2),1);]...
        +[S(N,2) S(N,4)]*[p(k-N,1);p(k-N,2)];
end

subplot(2,1,2);
plot(t',y(:,2),'b-');
set(gca(),"data_bounds",[0 40 0 1.4]); //putting bounds on display
l=legend("$x_D\text{ loop in manual}$",position=1);
xtitle("","Time(min)","$x_B$");
a=get("current_axes");
c=a.y_label;c.font_size=5;
```

```

//==========================================================================//
//==========================================================================//
//==========================================================================//
//========Set point as +1 in X-D==Both loops Automatic===========//
//p is the controller output
p=zeros(n,2);
delta_p=zeros(n,2);
e=zeros(n,2); //errors=(ysp-y) on which PI acts
ysp=zeros(n,2);
ysp((n-1)/2+1:n,1)=ones(n-((n-1)/2+1)+1,1);

t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
y=zeros(n,2);


for k=(n-1)/2+1:n

    //Error  e
    e(k,:)=ysp(k-1,:)-y(k-1,:);
    delta_e(k,:)=e(k,:)-e(k-1,:);

    //Controller calculation----Digital PID-----Eqn 7-28 Pg 136 (Velocity form)
//     p(k,:)=p(k-1,:)+flipdim([delta_e(k,:)+e(k,:)*diag(Ts./tauI)]*diag(Kc),2);
    p(k,:)=p(k-1,:)+([delta_e(k,:)+e(k,:)*diag(Ts./tauI)]*diag(Kc));
    //1-1/2-2 pairing

    delta_p(k,:)=p(k,:)-p(k-1,:);
```

```

    //Output
    y(k,1)=[S(1:N-1,1);S(1:N-1,3)]'...
        *[flipdim(delta_p(k-N+1:k-1,1),1);flipdim(delta_p(k-N+1:k-1,2),1)]...
        +[S(N,1) S(N,3)]*[p(k-N,1);p(k-N,2)];
    y(k,2)=[S(1:N-1,2);S(1:N-1,4)]'...
        *[flipdim(delta_p(k-N+1:k-1,1),1);flipdim(delta_p(k-N+1:k-1,2),1);]...
        +[S(N,2) S(N,4)]*[p(k-N,1);p(k-N,2)];
end

subplot(2,1,1);
plot(t',y(:,1),'r--');
set(gca(),"data_bounds",[0 40 0 1.4]); //putting bounds on display
l=legend("$x_B\text{ loop in manual}$","$\text{Both loops in automatic}$",position=1);
//l.font_size=5;
xtitle("","Time(min)","$x_D$");
a=get("current_axes");
c=a.y_label;c.font_size=5;


//=============================================================================//
//=============================================================================//
//=============================================================================//
//========Set point as +1 in X_B==Both loops Automatic===========//
//p is the controller output
p=zeros(n,2);
delta_p=zeros(n,2);
```

```
e=zeros(n,2); //errors=(ysp-y) on which PI acts
ysp=zeros(n,2);
ysp((n-1)/2+1:n,2)=ones(n-((n-1)/2+1)+1,1);

t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
y=zeros(n,2);


for k=(n-1)/2+1:n

    //Error  e
    e(k,:)=ysp(k-1,:)-y(k-1,:);
    delta_e(k,:)=e(k,:)-e(k-1,:);

    //Controller calculation----Digital PID----Eqn 7-28 Pg 136 (Velocity form)
//     p(k,:)=p(k-1,:)+flipdim([delta_e(k,:)+e(k,:)*diag(Ts./tauI)]*diag(Kc),2);
    p(k,:)=p(k-1,:)+([delta_e(k,:)+e(k,:)*diag(Ts./tauI)]*diag(Kc));
    //1-1/2-2 pairing

    delta_p(k,:)=p(k,:)-p(k-1,:);

    //Output
    y(k,1)=[S(1:N-1,1);S(1:N-1,3)]'...
        *[flipdim(delta_p(k-N+1:k-1,1),1);flipdim(delta_p(k-N+1:k-1,2),1)]...
        +[S(N,1) S(N,3)]*[p(k-N,1);p(k-N,2)];
    y(k,2)=[S(1:N-1,2);S(1:N-1,4)]'...
        *[flipdim(delta_p(k-N+1:k-1,1),1);flipdim(delta_p(k-N+1:k-1,2),1);]...
        +[S(N,2) S(N,4)]*[p(k-N,1);p(k-N,2)];
end

subplot(2,1,2);
plot(t',y(:,2),'r--');
set(gca(),"data_bounds",[0 40 0 1.4]); //putting
```

```
bounds on display
l=legend("$x_D\text{ loop in manual}$","$\text{Both loops in automatic}$",position=1);
xtitle("","Time(min)","$x_B$");
a=get("current_axes");
c=a.y_label;c.font_size=5;


//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

```

clear
clc

//Example 16.6
disp('Example 16.6')


K1=[1 0;10 1];//K with K12=0

eig1=spec(K1);
sigma1=spec(K1'*K1);
CN1=sqrt(max(sigma1)/min(sigma1))
mprintf('\nEigenvalues of K1 are %f and %f\n and CN is %f',eig1',CN1)




K2=[1 0.1;10 1];//K with K12=0.1

eig2=spec(K2);
```

```
sigma2=spec(K2'*K2);
CN2=sqrt(max(sigma2)/min(sigma2))

mprintf('\nEigenvalues of K2 are %f and %f\n and CN
    is %f',eig2',CN2)
```

Scilab code Exa 16.7 Preferred multiloop control strategy

```
clear
clc

//Example 16.7
disp('Example 16.7')


X=[0.48 0.9 -0.006;0.52 0.95 0.008; 0.90 -0.95
    0.020];
[U,S,V]=svd(X)

RGA=X.*([inv(X)]')  //Eqn 16-36

//Condition no. of X
CN=max(diag(S))/min(diag(S))

//Note that condition no. can also be found with
    command cond(X)

// The RGA given in the book is wrong! Eqn 16-73 is
    wrong.
mprintf('\n The RGA given in the book is wrong! Eqn
    16-73 is wrong.\n')
disp(RGA,'RGA=')

X1=X(1:2,1:2);
X2=X(1:2,[1 3]);
```

```
X3=X(1:2,2:3);

X4=X([1 3],1:2);
X5=X([1 3],[1 3]);
X6=X([1 3],2:3);

X7=X([2 3],1:2);
X8=X([2 3],[1 3]);
X9=X([2 3],2:3);

lamda1=max(X1.*inv(X1'));
lamda2=max(X2.*inv(X2'));
lamda3=max(X3.*inv(X3'));
lamda4=max(X4.*inv(X4'));
lamda5=max(X5.*inv(X5'));
lamda6=max(X6.*inv(X6'));
lamda7=max(X7.*inv(X7'));
lamda8=max(X8.*inv(X8'));
lamda9=max(X9.*inv(X9'));


mprintf('\n Pairing no.                CN          lambda
   \n')
mprintf('\n 1                %f          %f',cond(X1),
   lamda1)
mprintf('\n 2                %f          %f',cond(X2),
   lamda2)
mprintf('\n 3                %f          %f',cond(X3),
   lamda3)
mprintf('\n 4                %f          %f',cond(X4),
   lamda4)
mprintf('\n 5                %f          %f',cond(X5),
   lamda5)
mprintf('\n 6                %f          %f',cond(X6),
   lamda6)
mprintf('\n 7                %f          %f',cond(X7),
   lamda7)
mprintf('\n 8                %f          %f',cond(X8),
```

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

Chapter 17

Digital Sampling Filtering and Control

Scilab code Exa 17.1 Performance of alternative filters

```
clear
clc

//Example 17.1
disp('Example 17.1')

//In this solution we assume that a sampled signal
    is given to us at a very fast
//sampling rate and then we resample from it for our
    computations
//This depicts how data is in practical situations.
//Since computers are digital data is always
    discrete
//A more kiddish way of writing this code would have
    been to make a function
//which takes time as input and gives signal value
    as output ie generate a
//continuous signal definition. Then no matter what
    our sampling time is
```

```
//we can always get the desired values by calling the function say func(Ts*k)
//where Ts denotes sampling time and k is the index no.(ie deltaT*k)
//In principle this will also work fine and will reduce the length of the code
//but this will not lead to learning for coding in practical situations

Ts=0.001 //sampling time for analog
t=0:Ts:5;
n=length(t);
square_base=0.5*squarewave((t-0.5)*2*%pi/3)+0.5;
ym=square_base+0.25*sin(t*2*%pi*9);
subplot(2,2,1)
plot(t,[square_base' ym'])
xtitle('Fig 17.6 (a)','Time(min)','Output');


//Analog Filter
tauf1=0.1;tauf2=0.4;
s=%s;
F1=syslin('c',1/(tauf1*s+1));
F2=syslin('c',1/(tauf2*s+1));
yf1=csim(ym,t,F1);
yf2=csim(ym,t,F2);
subplot(2,2,2);
plot(t,[yf1' yf2' square_base'])
legend("$\tau_F=0.1\ min$","$\tau_F=0.4\ min$",position=3);
xtitle('Fig 17.6 (b)','Time(min)','Output');

 //Note that analog filtering can also be achieved by perfect sampling in EWMA digital filter
 //Since Exponentially weighted digital filter is an exact discretization of analog
 //filter if we take Ts=0.001 ie the perfect sampling of data we get identical answers
```

```
//from digital or analog filter. You can try this by chaning Ts1 or Ts2 to 0.001

//Digital filtering
Ts1=0.05;Ts2=0.1;
alpha1=exp(-Ts1/tauf1);
alpha2=exp(-Ts2/tauf1);
samples1=1:Ts1/Ts:n;
samples2=1:Ts2/Ts:n;
yf1=zeros(length(samples1),1);
yf2=zeros(length(samples2),1);

for k=1:length(samples1)-1
    yf1(k+1)=alpha1*yf1(k)+(1-alpha1)*ym(samples1(k));
end
for k=1:length(samples2)-1
    yf2(k+1)=alpha2*yf2(k)+(1-alpha2)*ym(samples2(k));
end

subplot(2,2,3);
plot(t(samples1)',[yf1],color='blue');
plot(t(samples2)',yf2,color='red');
plot(t,square_base,color='black');
legend("$\Delta t=0.05 \ min$","$\Delta t=0.1\  min$",position=3);
xtitle('Fig 17.6 (c)','Time(min)','Output');



//Moving Filter
N1=3;
N2=7;
yf1=zeros(1,length(samples1))
yf2=zeros(1,length(samples1))
for k=N1+1:length(samples1)
    yf1(k)=yf1(k-1)+1/N1*(ym(samples1(k))-ym(
```

```
        samples1(k-N1)));
end
for k=N2+1:length(samples1)
    yf2(k)=yf2(k-1)+1/N2*(ym(samples1(k))-ym(samples1(k-N2)));
end
//for k=N2+1:n
//      yf2(k)=yf2(k-1)+1/N2*(ym(k)-ym(k-N2));
//end
subplot(2,2,4);
plot(t(samples1),[yf1' yf2'])
plot(t,square_base,color='black');
legend("$N^*=3$","$N^*=7$",position=4);
xtitle('Fig 17.6 (d)','Time(min)','Output');



//Now for the gaussian noise
scf();
Ts=0.001 //sampling time for analog
t=0:Ts:5;
n=length(t);
square_base=0.5*squarewave((t-0.5)*2*%pi/3)+0.5;
ym=square_base+grand(1,length(t),"nor", 0, sqrt(0.1));//0.1 is for setting variance=0.1
subplot(2,2,1)
plot(t,[square_base' ym'])
xtitle('Fig 17.6 (a)','Time(min)','Output');


//Analog Filter
tauf1=0.1;tauf2=0.4;
s=%s;
F1=syslin('c',1/(tauf1*s+1));
F2=syslin('c',1/(tauf2*s+1));
yf1=csim(ym,t,F1);
yf2=csim(ym,t,F2);
subplot(2,2,2);
```

```
plot(t,[yf1' yf2' square_base'])
legend("$\tau_F=0.1\ min$","$\tau_F=0.4\ min$",position=3);
xtitle('Fig 17.6 (b)','Time(min)','Output');


//Digital filtering
Ts1=0.05;Ts2=0.1;
alpha1=exp(-Ts1/tauf1);
alpha2=exp(-Ts2/tauf1);
samples1=1:Ts1/Ts:n;
samples2=1:Ts2/Ts:n;
yf1=zeros(length(samples1),1);
yf2=zeros(length(samples2),1);

for k=1:length(samples1)-1
    yf1(k+1)=alpha1*yf1(k)+(1-alpha1)*ym(samples1(k));
end
for k=1:length(samples2)-1
    yf2(k+1)=alpha2*yf2(k)+(1-alpha2)*ym(samples2(k));
end

subplot(2,2,3);
plot(t(samples1)',[yf1],color='blue');
plot(t(samples2)',yf2,color='red');
plot(t,square_base,color='black');
legend("$\Delta t=0.05 \ min$","$\Delta t=0.1\  min$",position=3);
xtitle('Fig 17.6 (c)','Time(min)','Output');



//Moving Filter
N1=3;
N2=7;
yf1=zeros(1,length(samples1))
```

```
yf2=zeros(1,length(samples1))
for k=N1+1:length(samples1)
    yf1(k)=yf1(k-1)+1/N1*(ym(samples1(k))-ym(
        samples1(k-N1)));
end
for k=N2+1:length(samples1)
    yf2(k)=yf2(k-1)+1/N2*(ym(samples1(k))-ym(
        samples1(k-N2)));
end
//for k=N2+1:n
//     yf2(k)=yf2(k-1)+1/N2*(ym(k)-ym(k-N2));
//end
subplot(2,2,4);
plot(t(samples1),[yf1' yf2'])
plot(t,square_base,color='black');
legend("$N^*=3$","$N^*=7$",position=4);
xtitle('Fig 17.6 (d)','Time(min)','Output');



mprintf("Please note that for guassian noise\n
   results...
    will be different from book owing to randomness\
        n...
    we do not know the seed for the random noise")
```

Scilab code Exa 17.2 Response of first order difference equation

```
clear
clc

```

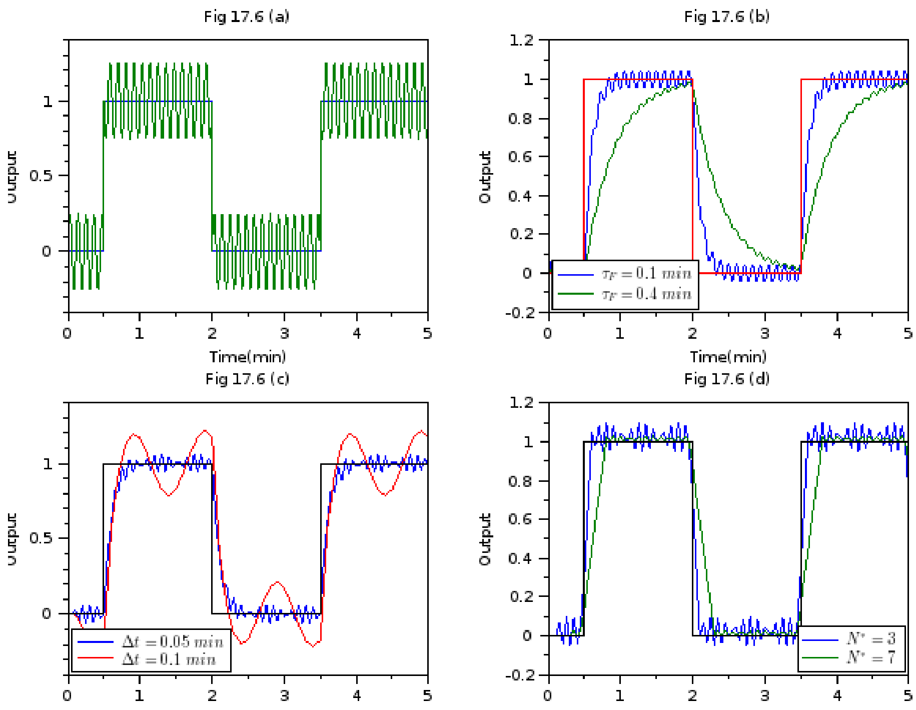


Figure 17.1: Performance of alternative filters

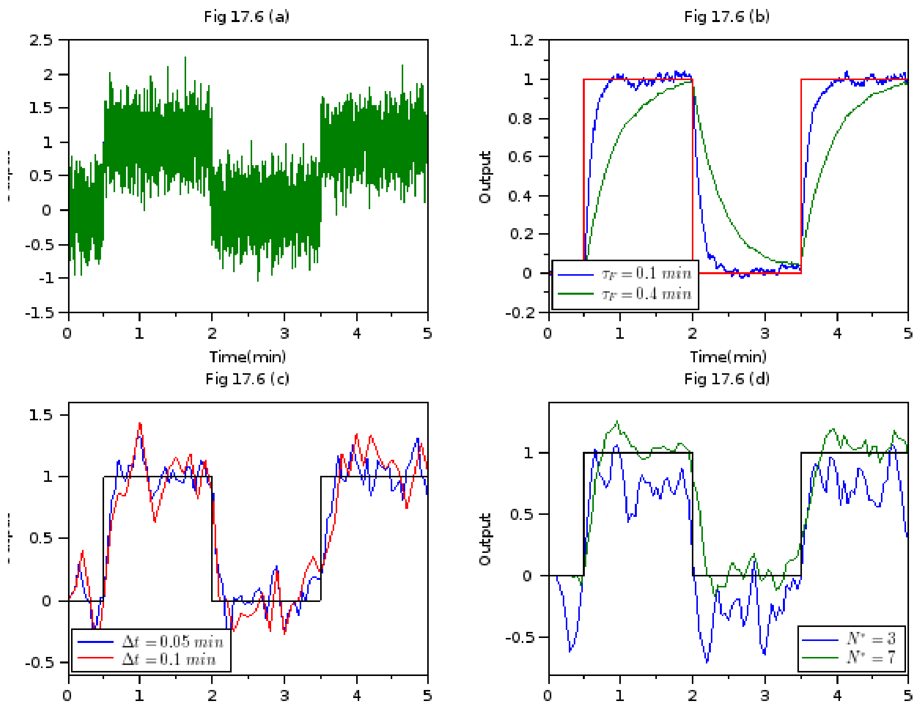


Figure 17.2: Performance of alternative filters

```
//Example 17.2
disp('Example 17.2')

Ts=1;//sampling time
K=2;
tau=1;
alpha=exp(-Ts/tau)
n=10;
y=zeros(n,1)
u=1; //input

for i=1:n
    y(i+1)=alpha*y(i)+K*(1-alpha)*u;
end

disp(y,'yk=')

mprintf("\n Note that in the book K=20 is wrong, it
   should be K=2\n...
 that is a first order function with gain 2 is given
     an input step")
```

Scilab code Exa 17.3 Recursive relation with inputs

```
clear
clc

//Example 17.3
disp('Example 17.3')

z=%z;
Gz=(-0.3225*z^-2+0.5712*z^-3)/(1-0.9744*z^-1+0.2231*
   z^-2);
G=tf2ss(Gz)
n=10;
```

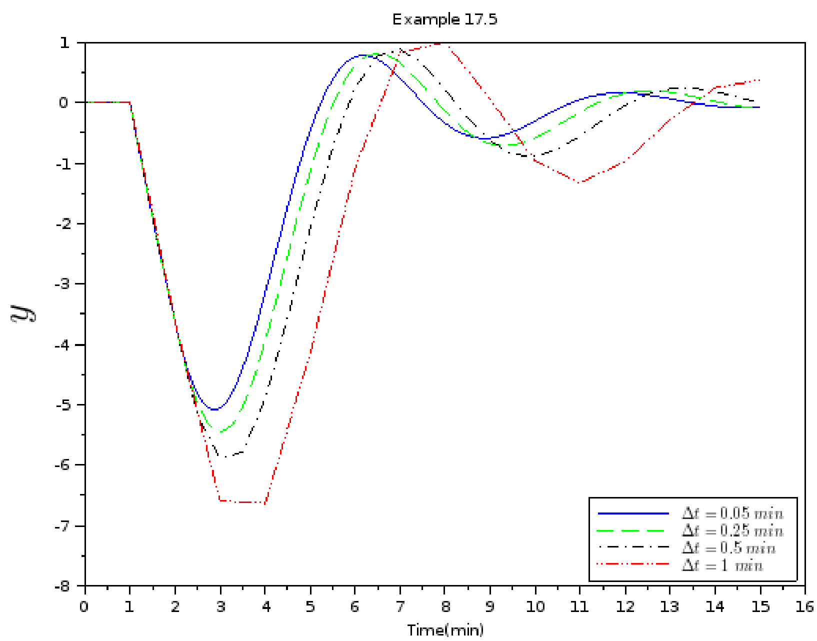


Figure 17.3: Digital control of pressure in a tank

```
u=ones(1,n);
y=dsimul(G,u);
disp(y','y=')

mprintf('\n\nAlternatively the simulation can also be done\n...
using syslin(d,Gz) and flts(u,Gz)\n\n')

Gz2=syslin('d',Gz);
y2=flts(u,Gz2)
disp(y2','y2=')
```

Scilab code Exa 17.5 Digital control of pressure in a tank

```
clear
clc

//Example 17.5
disp('Example 17.5')

deltaT=[0.05 0.25 0.5 1]';//sampling time
K=-20;
theta=1+(deltaT/2);//Add half of sampling time to delay for finding PI settings
tau=5;

//Table 11.3 ITAE disturbance settings
//Note that there is an error in book solution saying Table 11.2
//It should be table 11.3

Y=0.859*(theta/tau)^(-0.977);Kc=Y/K;
taui=tau*(0.674*(theta/tau)^-0.680).^-1;

mprintf('\n     ITAE(disturbance)    \n')
mprintf('   deltaT                Kc          tauI')
mprintf('\n  %f       %f     %f',deltaT,Kc,taui)

//Finding digital controller settings
//Eqn 17-55
a0=1+deltaT./taui;
a1=-(1); //since tauD=0
a2=0;
z=%z;

Gcz=Kc.*(a0+a1*z^-1)./(1-z^-1);

//Refer to table 17.1 to convert continuous transfer function to digital form
Gp=K*(1-exp(-1/tau*deltaT)).*z^(-1+(-1)./deltaT)
```

```
./(1-exp((-1)/tau*deltaT)*z^-1);//z^(-1/deltaT) for delay

G_CL=syslin('d',((Gp)./(Gcz.*Gp+1)));

t=0:deltaT(1):15
u=ones(1,length(t));
yt=flts(u,G_CL(1,1));
plot(t,yt,'-')

t=0:deltaT(2):15
u=ones(1,length(t));
yt=flts(u,G_CL(2,1));
plot(t,yt,'green--')

t=0:deltaT(3):15
u=ones(1,length(t));
yt=flts(u,G_CL(3,1));
plot(t,yt,'black-.')

t=0:deltaT(4):15
u=ones(1,length(t));
yt=flts(u,G_CL(4,1));
plot(t,yt,'red:')

set(gca(),"data_bounds",[0 15 -8 1]); //putting bounds on display
l=legend("$\Delta t=0.05\ min$","$\Delta t=0.25\ min$","$\Delta t=0.5\ min$","$\Delta t=1\ min$",position=4);
xtitle("Example 17.5","Time(min)","$y$");
a=get("current_axes");
c=a.y_label;c.font_size=5;

mprintf("\nNote that there is a mismatch between the book simulation and what\n...
what we get from SCILAB. The book is wrong. This has been crosschecked using\n...
```

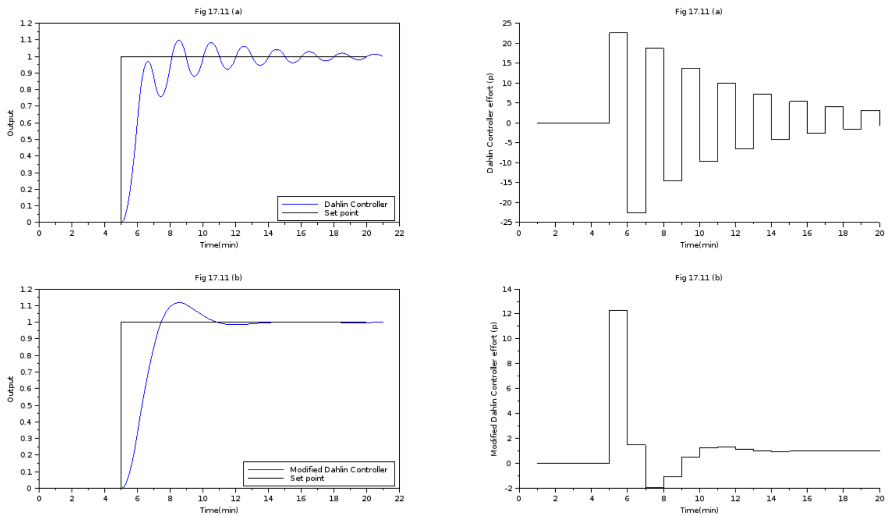


Figure 17.4: Dahlin controller

```
simulation in SIMULINK(MATLAB)")
```

Scilab code Exa 17.6 Dahlin controller

```
clear
clc

//Example 17.6
disp('Example 17.6')

//Note that for solving this example there are two ways
//One is to do this in xcos which is very easy to do
//and one can learn the same from example 17.5's solution
//To get the controller outputs at every point in
```

```
xcos
//just add a scope to the leg connecting controller and
//zero order hold unit before the continuous time block

//The other method is given here so that the reader learns more
//of what all can be done in scilab
//Here we deal with the controller in time domain rather than z domain

z=%z;
N=0;
a1=-1.5353;
a2=0.5866;
b1=0.0280;
b2=0.0234;
G=(b1+b2*z^-1)*z^(-N-1)/(1+a1*z^-1+a2*z^-2);

h=0;//no process delay
s=%s;
lamda=1;
Y_Ysp=1/(lamda*s+1);//exp(-h*s) is one because h=0 Eqn 17-62

Ts=1;//sampling time
A=exp(-Ts/lamda);
//Eqn 17-63
Y_Ysp_d=(1-A)*z^(-N-1)/(1-A*z^-1);

G_DC=1/G*(Y_Ysp_d)/(1-Y_Ysp_d); //Eqn 17-61



ysp=[zeros(1,4) ones(1,16)]
Gz_CL=syslin('d',G*G_DC/(G*G_DC+1));//Closed loop discrete system
```

```
yd=flts(ysp,Gz_CL) //Discrete Output due to set point change
//plot(yd)

e=ysp-yd; //Since we know set point and the output of the system we can use
//this info to find out the errors at the discrete time points
//note that here we have exploited in a very subtle way the property of a
//discrete system that only the values at discrete points matter for
//any sort of calculation

//Now this error can be used to find out the controller effort
e_coeff=coeff(numer(G_DC));
p_coeff=coeff(denom(G_DC));

n=20;//Time in minutes discretized with Ts=1 min
p=zeros(1,n); //Controller effort

for k=3:n
    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);
end
subplot(2,2,2)
plot2d2(p)
xtitle('Fig 17.11 (a)','Time(min)','Dahlin Controller effort (p)');

//Now we simulate the continuous version of the plant to get output in between
//the discrete point. This will help us ascertain the efficacy of the controller
//at points other than the discrete points
//Note that this is required to be checked because deltaT=1. had it been much
```

```
//smaller like 0.01 it would have been a good approx to a continuous system
//thus making this interpolation check redundant

s=%s;
Gp=syslin('c',1/(5*s+1)/(3*s+1));//continuous time version of process
Ts_c=0.01;//sampling time for continuous system
t=Ts_c:Ts_c:length([0 p])*Ts;
p_c=matrix(repmat([0 p],Ts/Ts_c,1),1,Ts/Ts_c*length([0 p]))//hack for zero order hold
//p_c means controller effort which is continous
yc=csim(p_c,t,Gp);
subplot(2,2,1)
plot(t,yc)
plot2d2(ysp)
legend("Dahlin Controller","Set point",position=4)
xtitle('Fig 17.11 (a)','Time(min)','Output');



//==============Now we do calculations for modified Dahlin controller========//
//==========================================================================================

//Y_Ysp_d=(1-A)*z^(-N-1)/(1-A*z^-1)*(b1+b2*z^-1)/(b1+b2); //Vogel Edgar

//Page 362 just after solved example
G_DC_bar=(1-1.5353*z^-1+0.5866*z^-2)/(0.0280+0.0234)*0.632/(1-z^-1);
//G_DC2=1/G*((1-A)*z^(-N-1))/(1-A*z^-1-(1-A)*z^(-N-1)); //Eqn 17-61
//G_DC=(1-1.5353*z^-1+0.5866*z^-2)/(0.0280+0.0234*z^-1)*0.632/(1-z^-1);

ysp=[zeros(1,4) ones(1,16)]
```

```
Gz_CL=syslin('d',G*G_DC_bar/(G*G_DC_bar+1));//Closed
    loop discrete system
yd=flts(ysp,Gz_CL) //Discrete Output due to set
   point change
//plot(yd)

e=ysp-yd; //Since we know set point and the output
   of the system we can use
//this info to find out the errors at the discrete
   time points
//note that here we have exploited in a very subtle
   way the property of a
//discrete system that only the values at discrete
   points matter for
//any sort of calculation

//Now this error can be used to find out the
   controller effort
e_coeff=coeff(numer(G_DC_bar));
p_coeff=coeff(denom(G_DC_bar));

n=20;//Time in minutes discretized with Ts=1 min
p=zeros(1,n); //Controller effort

for k=3:n
    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);
end
subplot(2,2,4)
plot2d2(p)
xtitle('Fig 17.11 (b)','Time(min)','Modified Dahlin
   Controller effort (p)');

//Now we simulate the continuous version of the
   plant to get output in between
//the discrete point. This will help us ascertain
   the efficacy of the controller
//at points other than the discrete points
```

```
//Note that this is required to be checked because deltaT=1. had it been much
//smaller like 0.01 it would have been a good approx to a continuous system
//thus making this interpolation check redundant

s=%s;
Gp=syslin('c',1/(5*s+1)/(3*s+1));//continuous time version of process
Ts_c=0.01;//sampling time for continuous system
t=Ts_c:Ts_c:length([0 p])*Ts;
p_c=matrix(repmat([0 p],Ts/Ts_c,1),1,Ts/Ts_c*length([0 p]))//hack for zero order hold
//p_c means controller effort which is continous
yc=csim(p_c,t,Gp);
subplot(2,2,3)
plot(t,yc)
plot2d2(ysp)
legend("Modified Dahlin Controller","Set point",position=4)
xtitle('Fig 17.11 (b)','Time(min)','Output');
```

Scilab code Exa 17.7 Non ringing Dahlin controller

```
clear
clc

//Example 17.7
disp('Example 17.7')

//Note that for solving this example there are two ways
//One is to do this in xcos which is very easy to do
```

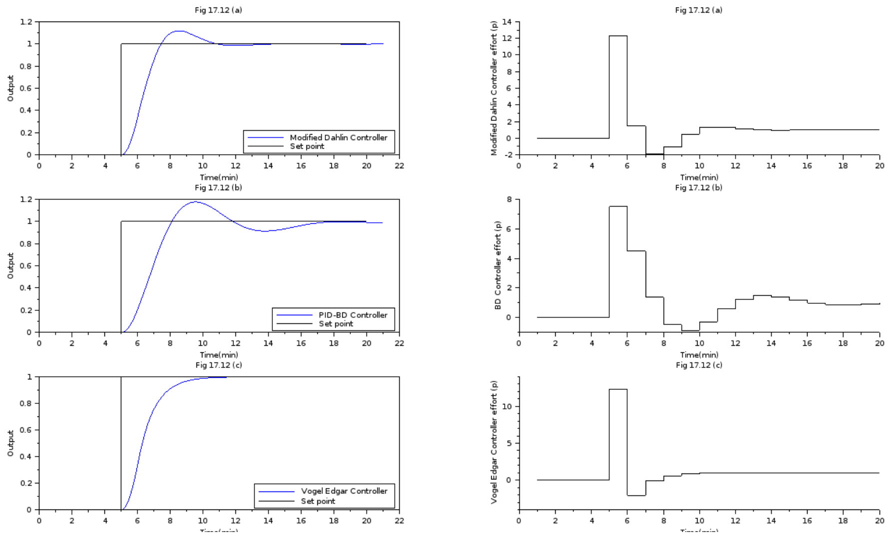


Figure 17.5: Non ringing Dahlin controller

```
//and one can learn the same from example 17.5's solution
//To get the controller outputs at every point in xcos
//just add a scope to the leg connecting controller and
//zero order hold unit before the continuous time block

//The other method is given here so that the reader learns more
//of what all can be done in scilab
//Here we deal with the controller in time domain rather than z domain

z=%z;
N=0;
a1=-1.5353;
a2=0.5866;
b1=0.0280;
```

```
b2=0.0234;
G=(b1+b2*z^-1)*z^(-N-1)/(1+a1*z^-1+a2*z^-2);

h=0;//no process delay
s=%s;
lamda=1;
Y_Ysp=1/(lamda*s+1);//exp(-h*s) is one because h=0 Eqn 17-62

Ts=1;//sampling time
A=exp(-Ts/lamda);


//============Now we do calculations for modified Dahlin controller========//
//==================================================================


//Page 362 just after solved example
G_DC_bar=(1-1.5353*z^-1+0.5866*z^-2)/(0.0280+0.0234)*0.632/(1-z^-1);

ysp=[zeros(1,4) ones(1,16)]
Gz_CL=syslin('d',G*G_DC_bar/(G*G_DC_bar+1));//Closed loop discrete system
yd=flts(ysp,Gz_CL) //Discrete Output due to set point change
//plot(yd)

e=ysp-yd; //Since we know set point and the output of the system we can use
//this info to find out the errors at the discrete time points
//note that here we have exploited in a very subtle way the property of a
//discrete system that only the values at discrete points matter for
```

```
//any sort of calculation

//Now this error can be used to find out the controller effort
e_coeff=coeff(numer(G_DC_bar));
p_coeff=coeff(denom(G_DC_bar));

n=20;//Time in minutes discretized with Ts=1 min
p=zeros(1,n); //Controller effort

for k=3:n
    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);
end
subplot(3,2,2)
plot2d2(p)
xtitle('Fig 17.12 (a)','Time(min)','Modified Dahlin Controller effort (p)');

//Now we simulate the continuous version of the plant to get output in between
//the discrete point. This will help us ascertain the efficacy of the controller
//at points other than the discrete points
//Note that this is required to be checked because deltaT=1. had it been much
//smaller like 0.01 it would have been a good approx to a continuous system
//thus making this interpolation check redundant

s=%s;
Gp=syslin('c',1/(5*s+1)/(3*s+1));//continuous time version of process
Ts_c=0.01;//sampling time for continuous system
t=Ts_c:Ts_c:length([0 p])*Ts;
p_c=matrix(repmat([0 p],Ts/Ts_c,1),1,Ts/Ts_c*length([0 p]))//hack for zero order hold
//p_c means controller effort which is continous
```

```
yc=csim(p_c,t,Gp);
subplot(3,2,1)
plot(t,yc)
plot2d2(ysp)
legend("Modified Dahlin Controller","Set point",position=4)
xtitle('Fig 17.12 (a)','Time(min)','Output');




//==============Now we do calculations for PID-BD controller=========//
//==================================================================================

G_BD=4.1111*(3.1486-5.0541*z^-1+2.0270*z^-2)/(1.7272-2.4444*z^-1+0.7222*z^-2)


ysp=[zeros(1,4) ones(1,16)]
Gz_CL=syslin('d',G*G_BD/(G*G_BD+1));//Closed loop discrete system
yd=flts(ysp,Gz_CL) //Discrete Output due to set point change
//plot(yd)

e=ysp-yd; //Since we know set point and the output of the system we can use
//this info to find out the errors at the discrete time points
//note that here we have exploited in a very subtle way the property of a
//discrete system that only the values at discrete points matter for
//any sort of calculation

//Now this error can be used to find out the
```

```
controller effort
e_coeff=coeff(numer(G_BD));
p_coeff=coeff(denom(G_BD));

n=20;//Time in minutes discretized with Ts=1 min
p=zeros(1,n); //Controller effort

for k=3:n
    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);
end
subplot(3,2,4)
plot2d2(p)
xtitle('Fig 17.12 (b)','Time(min)','BD Controller effort (p)');

//Now we simulate the continuous version of the plant to get output in between
//the discrete point. This will help us ascertain the efficacy of the controller
//at points other than the discrete points
//Note that this is required to be checked because deltaT=1. had it been much
//smaller like 0.01 it would have been a good approx to a continuous system
//thus making this interpolation check redundant

s=%s;
Gp=syslin('c',1/(5*s+1)/(3*s+1));//continuous time version of process
Ts_c=0.01;//sampling time for continuous system
t=Ts_c:Ts_c:length([0 p])*Ts;
p_c=matrix(repmat([0 p],Ts/Ts_c,1),1,Ts/Ts_c*length([0 p]))//hack for zero order hold
//p_c means controller effort which is continous
yc=csim(p_c,t,Gp);
subplot(3,2,3)
plot(t,yc)
```

```
plot2d2(ysp)
legend("PID-BD Controller","Set point",position=4)
xtitle('Fig 17.12 (b)','Time(min)','Output');



//===============Now we do calculations for Vogel Edgar Dahlin controller========//
//==================================================================================================

Y_Ysp_d=(1-A)*z^(-N-1)/(1-A*z^-1)*(b1+b2*z^-1)/(b1+b2); //Vogel Edgar Eqn 17-70

G_VE=1/G*(Y_Ysp_d)/(1-Y_Ysp_d); //Eqn 17-61


ysp=[zeros(1,4) ones(1,16)]
Gz_CL=syslin('d',G*G_VE/(G*G_VE+1));//Closed loop discrete system
yd=flts(ysp,Gz_CL) //Discrete Output due to set point change
//plot(yd)

e=ysp-yd; //Since we know set point and the output of the system we can use
//this info to find out the errors at the discrete time points
//note that here we have exploited in a very subtle way the property of a
//discrete system that only the values at discrete points matter for
//any sort of calculation

//Now this error can be used to find out the controller effort
e_coeff=coeff(numer(G_VE));
p_coeff=coeff(denom(G_VE));
```

```

n=20;//Time in minutes discretized with Ts=1 min
p=zeros(1,n); //Controller effort

for k=3:n
    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);
end
subplot(3,2,6)
plot2d2(p)
xtitle('Fig 17.12 (c)','Time(min)','Vogel Edgar Controller effort (p)');

//Now we simulate the continuous version of the plant to get output in between
//the discrete point. This will help us ascertain the efficacy of the controller
//at points other than the discrete points
//Note that this is required to be checked because deltaT=1. had it been much
//smaller like 0.01 it would have been a good approx to a continuous system
//thus making this interpolation check redundant

s=%s;
Gp=syslin('c',1/(5*s+1)/(3*s+1));//continuous time version of process
Ts_c=0.01;//sampling time for continuous system
t=Ts_c:Ts_c:length([0 p])*Ts;
p_c=matrix(repmat([0 p],Ts/Ts_c,1),1,Ts/Ts_c*length([0 p]))//hack for zero order hold
//p_c means controller effort which is continous
yc=csim(p_c,t,Gp);
subplot(3,2,5)
plot(t,yc)
plot2d2(ysp)
legend("Vogel Edgar Controller","Set point",position=4)
```

```
xtitle('Fig 17.12 (c)','Time(min)','Output');


mprintf("Note that there is some very slight
    difference between the \n...
curves shown in book and that obtained from scilab\n
    ...
this is simply because of more detailed calculation
    in scilab ")
```

Chapter 19

Real Time Optimization

Scilab code Exa 19.2 Nitration of Decane

```
clear
clc

//Example 19.2
disp('Example 19.2')

function y=f_DNO3(r1)
    D1=0.5;D2=0.5;
    r2=0.4-0.5*r1;
    y=r1*D1/(1+r1)^2/(1+r2)+r2*D2/(1+r1)/(1+r2)^2
endfunction

function [f, g, ind] = costf(x, ind)
    f=-f_DNO3(x);//cost is negative of function to
       be maximised
    g=-derivative(f_DNO3,x);//derivative of the cost
        function
endfunction

[fopt, xopt] = optim(costf,0.5);

```

```
disp(xopt,"Optimum value of r1=")
disp(-fopt,"Max value of DNO3=")

mprintf('Note that the answer in book is not as
    accurate as the one\n...
calculated from scilab')
```

Scilab code Exa 19.3 Refinery blending and production

```
clear
clc

//Example 19.3
disp('Example 19.3')

//function for minimization
c=-[-24.5 -16 36 24 21 10]';
//Equality Constraints
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];
beq=zeros(4,1);
//Inequality Constraints
A=[0 0 1 0 0 0;0 0 0 1 0 0;0 0 0 0 1 0];
b=[24000 2000 6000]';
//Lower bound on x
lb=zeros(6,1);
//Initial guess: such that it satisfies Aeq*x0=beq
x0=zeros(6,1);
x0(1:2)=[5000 3000]';//Initial guess for x1 and x2
x0(3:6)=Aeq(:,3:6)\(beq-Aeq(:,1:2)*x0(1:2));//
    solution of linear equations
//Note that x0 should also satisfy A*x0<b and lb


[xopt,fopt]=karmarkar(Aeq,beq,c,x0,[],[],[],[],A,b,
```

```
    lb)

disp(xopt,"Optimum value of x=")
mprintf("\nMax value of f=$ %f /day\n",-fopt)

mprintf('\n Note that the answer in book is not as
    accurate as the one\n...
 calculated from scilab')
```

Scilab code Exa 19.4 Fuel cost in boiler house

```
clear
clc

//Example 19.4
disp('Example 19.4')

//Here we have Nonlinear programming problem hence
    we use optim function
//Since optim does not have the ability to handle
    constraints
//we use the penalty method for optimization
//ie we make the constraints a part of the cost
    function such that
//cost function increases severly for any violation
    of constraints
//MATLAB users must be familiar with fmincon
    function in MATLAB
//Unfortunately a similar function in Scilab is not
    yet available
//Fmincon toolbox development for scilab is under
    development/testing

x0=[2 4 4 1]'; //Initial guess

```

```
function y=func(x) //x is 4*1 vector
    P1=4.5*x(1)+0.1*x(1)^2+4*x(2)+0.06*x(2)^2;
    P2=4*x(3)+0.05*x(3)^2+3.5*x(4)+0.2*x(4)^2;
    if (P1>30) then
        c1=abs(P1-30)^2;
    elseif P1<18
        c1=abs(P1-18)^2;
        else c1=0;
    end
    if (P2>25) then
        c2=abs(P2-30)^2;
    elseif P2<14
        c2=abs(P2-18)^2;
        else c2=0;
    end
    c3=abs(P1+P2-50)^2;
    c4=abs(x(2)+x(4)-5)^2;
    y=(x(1)+x(3))+100*(c1+c2+c3+c4);
endfunction

function [f, g, ind] = costf(x, ind)
    f=func(x);//cost is negative of function to be maximised
    g=derivative(func,x);//derivative of the cost function
endfunction

[fopt, xopt] = optim(costf,"b",zeros(4,1), 10*ones(4,1),x0);
// "b",zeros(4,1), 10*ones(4,1) stands for lower and upper bounds on x

disp(xopt,"Optimum value of x=")
disp(fopt,"Min value of f=")

mprintf('Note that the answer in book is not as accurate as the one\n...
calculated from scilab')
```

Chapter 20

Model Predictive Control

Scilab code Exa 20.1 Step response coefficients

```
clear
clc

//Example 20.1
disp('Example 20.1')


K=5;
tau=15;//min
theta=2;//min
Ts=1;//Sampling period
k=[0:79]';//samples
N=80;


//From  eqn 20-5
S=zeros(N,1);
S=K*(1-exp(-(k*Ts-theta)/tau));
S(1:(theta+1),1)=0;//delay
```

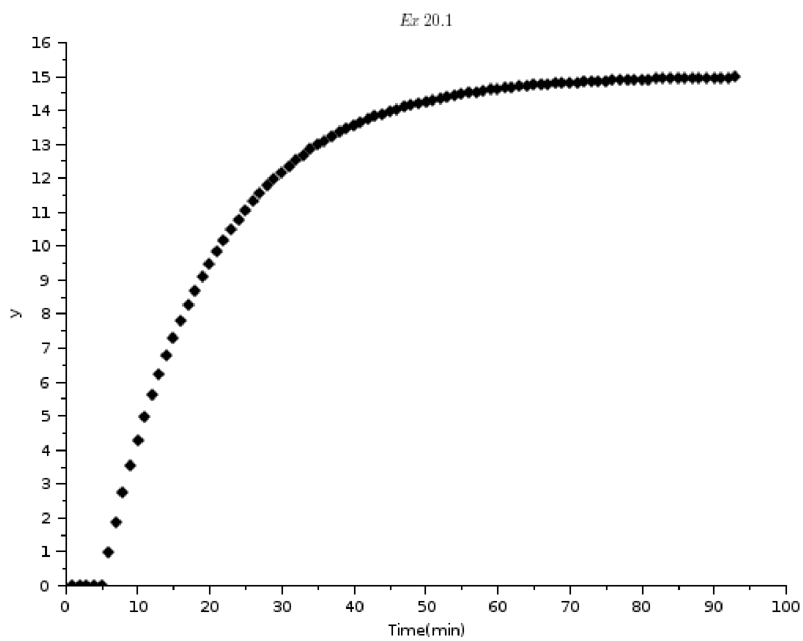


Figure 20.1: Step response coefficients

```


//Step change
M=3;

//Calculating step response from Eqn 20-4
step=3;//step change occurs at t=3 min
i=[(theta+1):90]';
yi=[zeros(theta+step,1);K*M*(1-exp(-(i*Ts-theta)/tau));]

plot2d(yi,style=-4);
xtitle("$Ex\ 20.1$","Time(min)","y")
```

Scilab code Exa 20.3 J step ahead prediction

```
clear
clc

//Example 20.3
disp('Example 20.3')


for J=[3 4 6 8] //Tuning parameter

Ts=5;//Sampling time ie delta_T
N=16;//Model Order

s=%s;
G=syslin('c',1/(5*s+1)^5);//Transfer function
t=0:Ts:N*Ts;
S=csim('step',t,G)';//generating step response model
//plot(t,S);
```

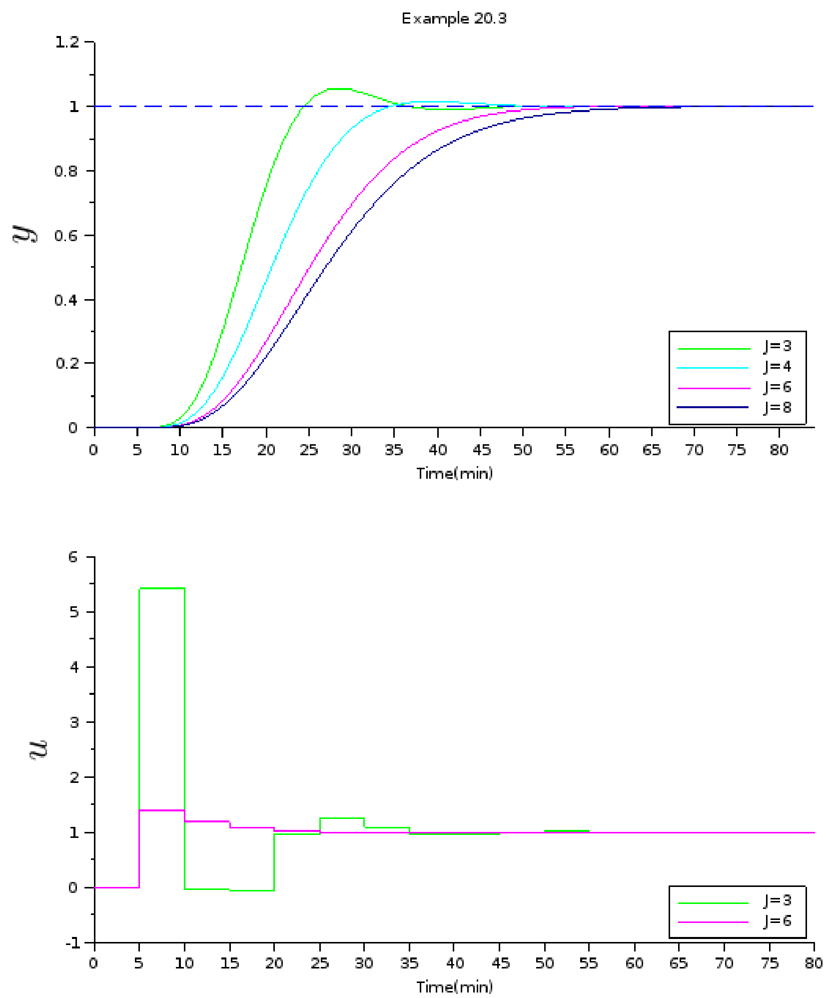


Figure 20.2: J step ahead prediction

```
S(1)=[];

T=80;//simulation time
n=T/Ts*2+1;
u=zeros(1,n);
//Input initialization 80 min is the Time for simulation
//We take a few u values in negative time to facilitate usage of step response
//model
delta_u=[0 diff(u)];
yhat_u=zeros(n,1);
ysp=1;
for k=(n-1)/2+1+1:n-J //an additional +1 is because MPC starts after set point change
    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
    disp(yhat_u(k+J))
    delta_u(k)=(ysp-yhat_u(k+J))/S(J);
    u(k)=u(k-1)+delta_u(k);
end
u(n-J+1:$)=u(k)*ones(1,J);//Carry forward the u as constant

t=-(n-1)/2*Ts:Ts:(n-1)*Ts/2;
subplot(2,1,2);
if J==3 | J==6 then
    plot2d2(t((n-1)/2+1:n),u((n-1)/2+1:n),style=J);
end
legend("J=3","J=6",position=4)
xtitle("","Time(min)","$u$");
a=get("current_axes");
c=a.y_label;c.font_size=5;


res=Ts;//resolution
//u_cont=matrix(repmat([0 u],res,1),1,res*length([0 u]));
```

```
u_cont=matrix(repmat([u],res,1),1,res*length([u]));
entries=length(u_cont);
t_cont=linspace(-T,T+Ts-1,entries);
yt=csim(u_cont,t_cont,G);
subplot(2,1,1);
if J=8 then //for color of plot2d
    J=9
end
plot2d(t_cont((entries-Ts)/2+1:$),yt((entries-Ts)/2+1:$),style=J,rect=[0,0,80,1.2]);
end

//Other niceties for plots
subplot(2,1,1);
plot(t_cont((entries-Ts)/2+1:$),ones(length(t_cont((entries-Ts)/2+1:$)),1),'--');
legend("J=3","J=4","J=6","J=8",position=4)
xtitle("Example 20.3","Time(min)","$y$");
a=get("current_axes");
c=a.y_label;c.font_size=5;
```

Scilab code Exa 20.4 Output feedback and bias correction

```
clear
clc

//Example 20.4
disp('Example 20.4')

J=15;
Ts=1;//Sampling time ie delta_T
N=80;//Model Order
s=%s;
G=syslin('c',5/(15*s+1));//Transfer function
t=0:Ts:N*Ts;
```

```
u_sim=ones(1,length(t));
u_sim(1:3)=[0 0 0]; //input delay to account for 2 min delay in G
S=csim(u_sim,t,G)';//generating step response model for real plant
//plot(t,S);
S(1)=[];
T=100;//simulation time

n=T/Ts*2+1; //no. of discrete points in our domain of analysis
//Input initialization T min is the Time for simulation
//We take a few u values in negative time to facilitate
//usage of step response model
u=zeros(n,1);
d=zeros(n,1);
delta_u=zeros(n,1);
delta_u(101+2)=1; //Step change at t=2 min
u=cumsum(delta_u);
delta_d=zeros(n,1);
delta_d(101+8)=0.15;//disturbance t=8 min
d=cumsum(delta_d);

yhat=zeros(n,1); //J step ahead predictions
ytilda=zeros(n,1); //J step ahead predictions corrected
b=zeros(n,1); //corrections

t=-(n-1)/2:Ts:(n-1)/2;

for k=(n-1)/2+1-J:n-J
    yhat(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1),1)+S(N)*u(k+J-N);
    //Predicted y  Eqn 20-10
    y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1),1)+S(N)*u(k+J-N)+...
```

```
            S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1),1)+S(N)*d(k+J-N);
    //Actual values of the process
    b(k+J)=y(k)-yhat(k); //Note that there is a delay in corrections
    //which is opposite of prediction
end
ytilda=b+yhat;//calculation of corrected y
plot(t,y,'-',t,yhat,'-.',t,ytilda,'--');
set(gca(),"data_bounds",[0 100 0 6]); //putting bounds on display
l=legend("y","$\hat y$","$\tilde y$",position=4);
l.font_size=5;
xtitle("Example 20.4","Time(min)","$y$");
a=get("current_axes");
c=a.y_label;c.font_size=5;


//======Part b=====//
G2=syslin('c',4/(20*s+1));//Transfer function
t2=0:Ts:N*Ts;
u_sim=ones(1,length(t2));
u_sim(1:3)=[0 0 0]; //input delay to account for 2 min delay in G
S2=csim(u_sim,t2,G2)';//generating step response model for model
//plot(t2,S);
S2(1)=[];

yhat=zeros(n,1); //J step ahead predictions
ytilda=zeros(n,1); //J step ahead predictions corrected
b=zeros(n,1); //corrections

for k=(n-1)/2+1-J:n-J
    yhat(k+J)=S2(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1),1)+S2(N)*u(k+J-N);
    //Predicted y Eqn 20-10
```

```
    y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1),1)+S(N)*u(k+J-N)+...
            S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1),1)+S(N)*d(k+J-N);
    //Actual values of the process
    b(k+J)=y(k)-yhat(k); //Note that there is a delay in corrections
    //which is opposite of prediction
end
ytilda=b+yhat;//calculation of corrected y
scf();
plot(t,y,'-',t,yhat,'-.',t,ytilda,'--');
set(gca(),"data_bounds",[0 100 0 6]); //putting bounds on display
l=legend("y","$\hat y$","$\tilde y$",position=4);
l.font_size=5;
xtitle("Example 20.4","Time(min)","$y$");
a=get("current_axes");
c=a.y_label;c.font_size=5;
```

Scilab code Exa 20.5 Comparison of MPCs and PID

```
clear
clc

//Example 20.5
disp('Example 20.5')

//==============Part(a)=========//
//==============Part(a)=========//
//==============Part(a)=========//
```

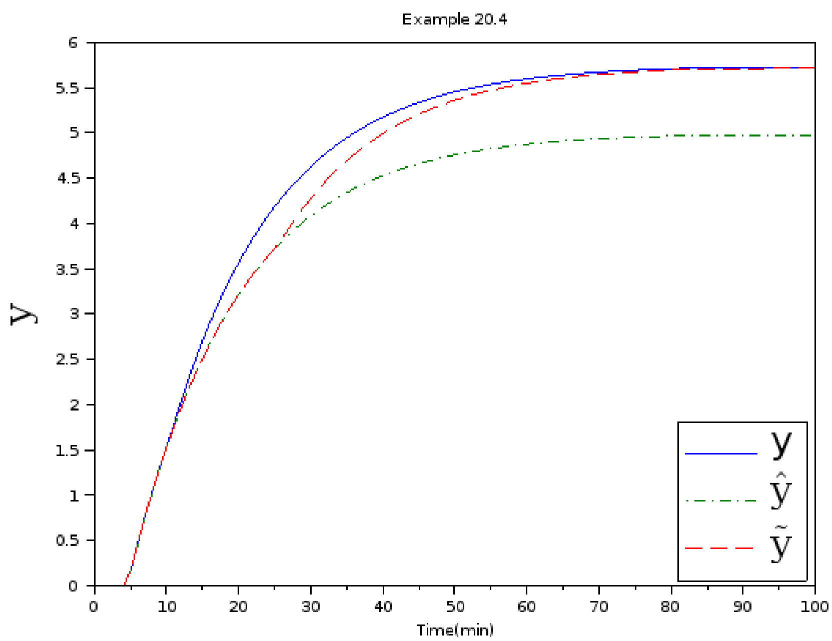


Figure 20.3: Output feedback and bias correction

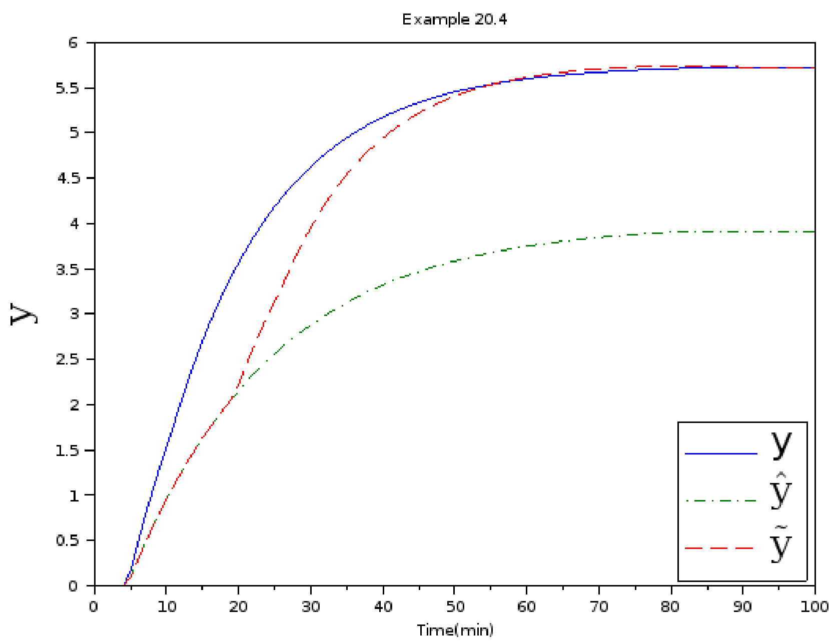


Figure 20.4: Output feedback and bias correction

```
//==============Part(a)========//
//==============Part(a)========//

Ts=1;//Sampling time ie delta_T
N=70;//Model Order
s=%s;
G=syslin('c',1/(5*s+1)/(10*s+1));//Transfer function
t=0:Ts:(N-1)*Ts;
u_sim=ones(1,length(t));
//There is automatically an input delay of one unit in csim function
S=csim(u_sim,t,G)';//generating step response model for real plant
//plot(t,S);
T=80;//simulation time

//Let the three simulations correspond to
//MPC1==> P=3,M=1
//MPC2==> P=4,M=2
///PID==> The PID controller

P1=3;M1=1;
P2=4;M2=2;
S1=S(1:P1);//MPC-1
S2=[S(1:P2) [0;S(1:P2-1)]];//MPC-2

//SISO system
Q=1;
R=0;//No move suppression

Kc1=inv(S1'*Q*S1+R*eye(M1,M1))*S1'*Q;//Eqn20-57 MPC1
Kc2=inv(S2'*Q*S2+R*eye(M2,M2))*S2'*Q;//Eqn20-57 MPC2

mprintf('\nFor P=3 and M=1,    \nKc=\n     [%f %f %f]\n',Kc1)
mprintf('\nFor P=4 and M=2,\nKc=')
```

```
disp(Kc2)

//===============Part(b)=========//
//===============Part(b)=========//
//===============Part(b)=========//
//===============Part(b)=========//
//===============Part(b)=========//
//===============Part(b)=========//

//===============Part(b) MPC-1=========//
n=T/Ts*2+1; //no. of discrete points in our domain of analysis
//Input initialization T is the Time for simulation
//We take a few u values in negative time to facilitate
//usage of step response model
u=zeros(n,1);
d=zeros(n,1);
delta_u=zeros(n,1);
//delta_u(101+2)=1; //Step change at t=2 min
u=cumsum(delta_u);
delta_d=zeros(n,1);
//delta_d(101+8)=0.15;//disturbance t=8 min
d=cumsum(delta_d);

y=zeros(1,n);//Actual values
yhat=zeros(1,n); //predicted value
ydot=zeros(P1,n); //Unforced predictions
ydottilde=zeros(P1,n); //Corrected unforced predictions
yr=ones(P1,n);//reference trajectory(same as setpoint)
edot=zeros(P1,n);//predicted unforced error

t=-(n-1)/2:Ts:(n-1)/2;

for k=(n-1)/2+1:n-P1

```

```
    //Unforced predictions
    for J=1:P1
        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);
    end

    //Actual values of the process
    J=0;
    y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1),1)+S(N)*u(k+J-N)+...
            S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1),1)+S(N)*d(k+J-N);

    //Predicted value of the process
    J=0;
    yhat(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1),1)+S(N)*u(k+J-N);

    //Corrected prediction for unforced case
    ydottilde(:,k+1)=ydot(:,k+1)+ones(P1,1)*(y(k)-yhat(k));

    //Predicted unforced error   Eqn20-52
    edot(:,k+1)=yr(:,k+1)-ydottilde(:,k+1);

    //Control move
    delta_u(k)=Kc1*edot(:,k+1);
    u(k)=u(k-1)+delta_u(k);

end
subplot(1,2,1);
plot(t,y,'black-');
subplot(1,2,2);
plot2d2(t,u);

//==============Part(b) MPC-2========//
n=T/Ts*2+1; //no. of discrete points in our domain of analysis
```

```
//Input initialization T is the Time for simulation
//We take a few u values in negative time to facilitate
//usage of step response model
u=zeros(n,1);
d=zeros(n,1);
delta_u=zeros(n,1);
//delta_u(101+2)=1; //Step change at t=2 min
u=cumsum(delta_u);
delta_d=zeros(n,1);
//delta_d(101+8)=0.15;//disturbance t=8 min
d=cumsum(delta_d);

y=zeros(1,n);//Actual values
yhat=zeros(1,n); //predicted value
ydot=zeros(P2,n); //Unforced predictions
ydottilde=zeros(P2,n); //Corrected unforced predictions
yr=ones(P2,n);//reference trajectory(same as setpoint)
edot=zeros(P2,n);//predicted unforced error

t=-(n-1)/2:Ts:(n-1)/2;

for k=(n-1)/2+1:n-P2

    //Unforced predictions
    for J=1:P2
        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);
    end

    //Actual values of the process
    J=0;
    y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1),1)+S(N)*u(k+J-N)+...
            S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1),1)+S(N)*d(k+J-N);
```

```
    //Predicted value of the process
    J=0;
    yhat(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1),1)+S(N)*u(k+J-N);

    //Corrected prediction for unforced case
    ydottilde(:,k+1)=ydot(:,k+1)+ones(P2,1)*(y(k)-yhat(k));

    //Predicted unforced error   Eqn20-52
    edot(:,k+1)=yr(:,k+1)-ydottilde(:,k+1);

    //Control move
    delta_u(k)=Kc2(1,:)*edot(:,k+1);
    u(k)=u(k-1)+delta_u(k);

end
subplot(1,2,1);
plot(t,y,'-.');
set(gca(),"data_bounds",[0 60 0 1.25]); //putting bounds on display
l=legend("MPC(P=3,M=1)","MPC(P=4,M=2)",position=4);
xtitle("Process Output","Time(min)","$y$");
a=get("current_axes");
c=a.y_label;c.font_size=5;

subplot(1,2,2);
plot2d2(t,u,style=2);



//==============Part(b) PID========//
n=T/Ts*2+1; //no. of discrete points in our domain of analysis
//Input initialization T is the Time for simulation
//We take a few u values in negative time to facilitate
```

```
//usage of step response model
u=zeros(n,1);
d=zeros(n,1);
delta_u=zeros(n,1);
delta_d=zeros(n,1);
//delta_d(101+8)=0.15;//disturbance t=8 min
d=cumsum(delta_d);

y=zeros(n,1);//Actual values
ysp=1;//setpoint
e=zeros(n,1);//error
delta_e=zeros(n,1);//error

t=-(n-1)/2:Ts:(n-1)/2;

//PID settings
Kc=2.27;taui=16.6;tauD=1.49;

for k=(n-1)/2+1:n-1
    //Actual values of the process
    J=0;
    y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1),1)+S(N)*u(k+J-N)+...
            S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1),1)+S(N)*d(k+J-N);
    //error
    e(k)=ysp-y(k);
    delta_e(k)=e(k)-e(k-1);

    //Controller move----Digital PID----Eqn 7-28 Pg 136 (Velocity form)
    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))]);
    delta_u(k)=u(k)-u(k-1);
end
subplot(1,2,1);
plot(t,y,'red--');
set(gca(),"data_bounds",[0 60 0 1.25]); //putting
```

```
bounds on display
l=legend("MPC (P=3,M=1)","MPC (P=4,M=2)","PID controller",position=4);
xtitle("Process Output","Time(min)","$y$");
a=get("current_axes");
c=a.y_label;c.font_size=5;

subplot(1,2,2);
plot2d2(t,u,style=5);
set(gca(),"data_bounds",[0 30 -100 100]); //putting bounds on display
l=legend("MPC (P=3,M=1)","MPC (P=4,M=2)","PID controller",position=4);
xtitle("Controller Output","Time(min)","$u$");
a=get("current_axes");
c=a.y_label;c.font_size=5;


//==================Part(c)==========//
//==================Part(c)==========//
//==================Part(c)==========//
//==================Part(c)==========//
//==================Part(c)==========//
//==================Part(c)==========//
//==================Part(c)==========//

//==================Part(c) MPC-1==========//
n=T/Ts*2+1; //no. of discrete points in our domain of analysis
//Input initialization T is the Time for simulation
//We take a few u values in negative time to facilitate
//usage of step response model
u=zeros(n,1);
d=zeros(n,1);
delta_u=zeros(n,1);
u=cumsum(delta_u);
delta_d=zeros(n,1);
```

```
delta_d((n-1)/2+1)=1;//disturbance t=0 min
d=cumsum(delta_d);

y=zeros(1,n);//Actual values
yhat=zeros(1,n); //predicted value
ydot=zeros(P1,n); //Unforced predictions
ydottilde=zeros(P1,n); //Corrected unforced
   predictions
yr=zeros(P1,n);//reference trajectory(same as
   setpoint)
edot=zeros(P1,n);//predicted unforced error

t=-(n-1)/2:Ts:(n-1)/2;

for k=(n-1)/2+1:n-P1

    //Unforced predictions
    for J=1:P1
        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);
    end

    //Actual values of the process
    J=0;
    y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1)
       ,1)+S(N)*u(k+J-N)+...
            S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1)
               ,1)+S(N)*d(k+J-N);

    //Predicted value of the process
    J=0;
    yhat(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J
       -1),1)+S(N)*u(k+J-N);

    //Corrected prediction for unforced case
    ydottilde(:,k+1)=ydot(:,k+1)+ones(P1,1)*(y(k)-
       yhat(k));

```

```
    //Predicted unforced error   Eqn20-52
    edot(:,k+1)=yr(:,k+1)-ydottilde(:,k+1);

    //Control move
    delta_u(k)=Kc1*edot(:,k+1);
    u(k)=u(k-1)+delta_u(k);

end

scf();
subplot(1,2,1);
plot(t,y,'black-');
subplot(1,2,2);
plot2d2(t,u);

//===============Part(c) MPC-2========//
n=T/Ts*2+1; //no. of discrete points in our domain of analysis
//Input initialization T is the Time for simulation
//We take a few u values in negative time to facilitate
//usage of step response model
u=zeros(n,1);
d=zeros(n,1);
delta_u=zeros(n,1);
u=cumsum(delta_u);
delta_d=zeros(n,1);
delta_d((n-1)/2+1)=1;//disturbance t=0 min
d=cumsum(delta_d);

y=zeros(1,n);//Actual values
yhat=zeros(1,n); //predicted value
ydot=zeros(P2,n); //Unforced predictions
ydottilde=zeros(P2,n); //Corrected unforced predictions
yr=zeros(P2,n);//reference trajectory(same as setpoint)
edot=zeros(P2,n);//predicted unforced error
```

```

t=-(n-1)/2:Ts:(n-1)/2;

for k=(n-1)/2+1:n-P2

    //Unforced predictions
    for J=1:P2
        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);
    end

    //Actual values of the process
    J=0;
    y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1),1)+S(N)*u(k+J-N)+...
            S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1),1)+S(N)*d(k+J-N);

    //Predicted value of the process
    J=0;
    yhat(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1),1)+S(N)*u(k+J-N);

    //Corrected prediction for unforced case
    ydottilde(:,k+1)=ydot(:,k+1)+ones(P2,1)*(y(k)-yhat(k));

    //Predicted unforced error   Eqn20-52
    edot(:,k+1)=yr(:,k+1)-ydottilde(:,k+1);

    //Control move
    delta_u(k)=Kc2(1,:)*edot(:,k+1);
    u(k)=u(k-1)+delta_u(k);

end
subplot(1,2,1);
plot(t,y,'-.');
subplot(1,2,2);
```

```
plot2d2(t,u,style=2);



//==============Part(C) PID========//
n=T/Ts*2+1; //no. of discrete points in our domain of analysis
//Input initialization T is the Time for simulation
//We take a few u values in negative time to facilitate
//usage of step response model

u=zeros(n,1);
d=zeros(n,1);
delta_u=zeros(n,1);
u=cumsum(delta_u);
delta_d=zeros(n,1);
delta_d((n-1)/2+1)=1;//disturbance t=0 min
d=cumsum(delta_d);

y=zeros(n,1);//Actual values
ysp=0;//setpoint
e=zeros(n,1);//error
delta_e=zeros(n,1);//error

t=-(n-1)/2:Ts:(n-1)/2;

//PID settings
Kc=3.52;taui=6.98;tauD=1.73;

for k=(n-1)/2+1:n-1
    //Actual values of the process
    J=0;
    y(k+J)=S(1:N-1)'*flipdim(delta_u(k+J-N+1:k+J-1),1)+S(N)*u(k+J-N)+...
            S(1:N-1)'*flipdim(delta_d(k+J-N+1:k+J-1),1)+S(N)*d(k+J-N);
    //error
```

```
    e(k)=ysp-y(k);
    delta_e(k)=e(k)-e(k-1);

    //Controller move----Digital PID----Eqn 7-28 Pg 136 (Velocity form)
    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))]);
    delta_u(k)=u(k)-u(k-1);
end
subplot(1,2,1);
plot(t,y,'red--');
set(gca(),"data_bounds",[0 60 -0.1 0.3]); //putting bounds on display
l=legend("MPC (P=3,M=1)","MPC (P=4,M=2)","PID controller",position=1);
xtitle("Process Output","Time(min)","$y$");
a=get("current_axes");
c=a.y_label;c.font_size=5;

subplot(1,2,2);
plot2d2(t,u,style=5);
set(gca(),"data_bounds",[0 30 -1.5 0]); //putting bounds on display
l=legend("MPC (P=3,M=1)","MPC (P=4,M=2)","PID controller",position=1);
xtitle("Controller Output","Time(min)","$u$");
a=get("current_axes");
c=a.y_label;c.font_size=5;
```

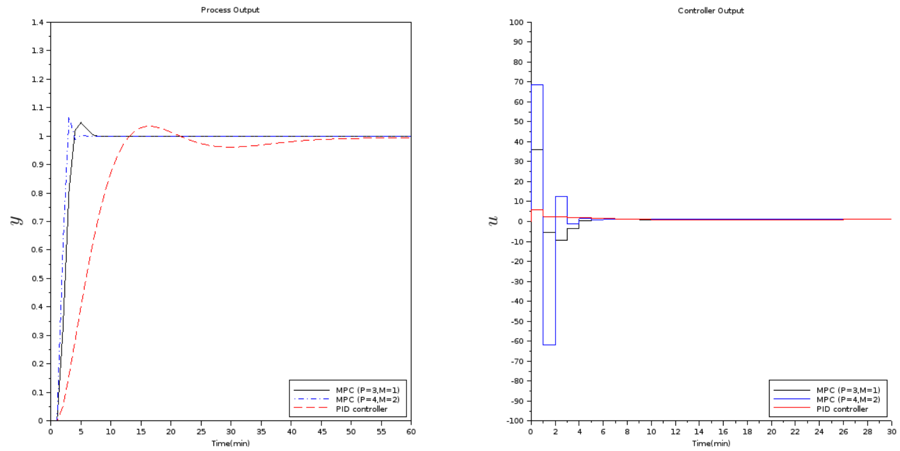


Figure 20.5: Comparison of MPCs and PID

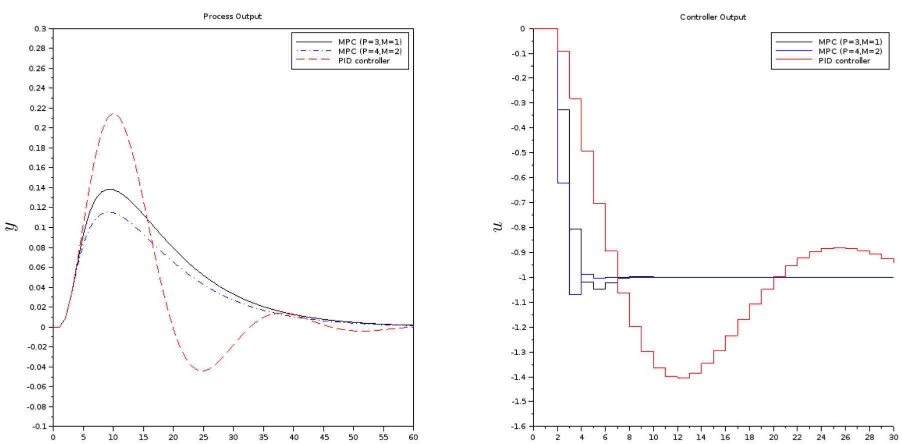


Figure 20.6: Comparison of MPCs and PID

Chapter 21

Process Monitoring

Scilab code Exa 21.2 Semiconductor processing control charts

```
clear
clc

//Example 21.2
disp('Example 21.2')

//data
x=[209.6    207.6    211.1
    183.5    193.1    202.4
    190.1    206.8    201.6
    206.9    189.3    204.1
    260.     209.     212.2
    193.9    178.8    214.5
    206.9    202.8    189.7
    200.2    192.7    202.1
    210.6    192.3    205.9
    186.6    201.5    197.4
    204.8    196.6    225.
    183.7    209.7    208.6
```

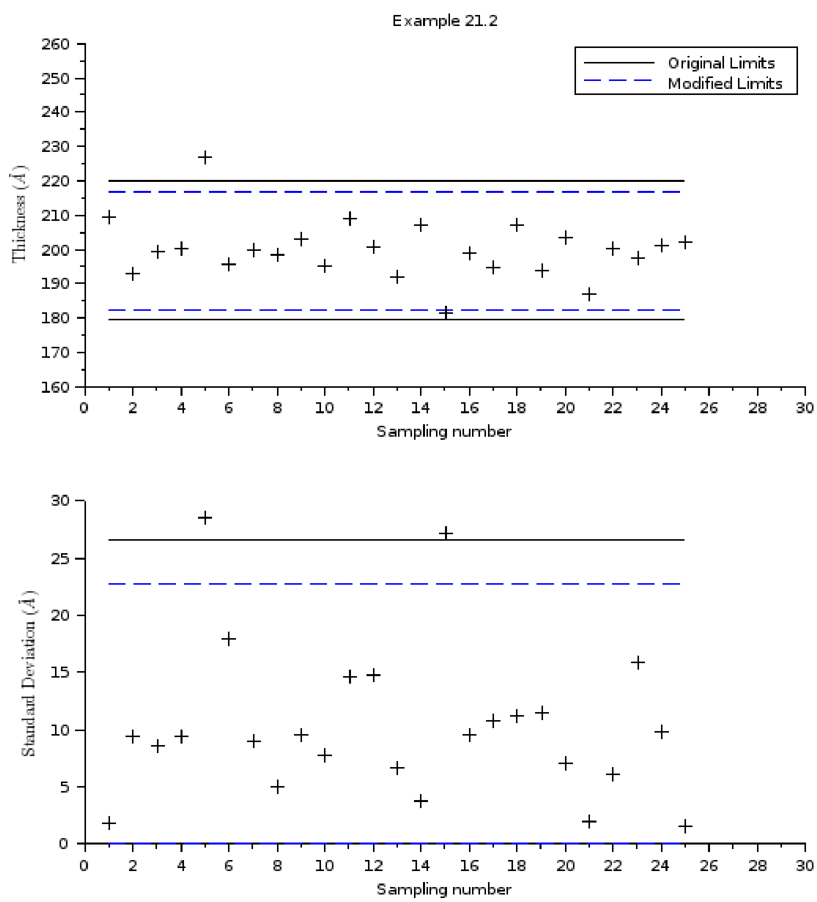


Figure 21.1: Semiconductor processing control charts

```
     185.6     198.9     191.5
     202.9     210.1     208.1
     198.6     195.2     150.
     188.7     200.7     207.6
     197.1     204.      182.9
     194.2     211.2     215.4
     191.      206.2     183.9
     202.5     197.1     211.1
     185.1     186.3     188.9
     203.1     193.1     203.9
     179.7     203.3     209.7
     205.3     190.      208.2
     203.4     202.9     200.4 ]


//Original Limits
n=3;
xbar=sum(x,2)/n; //mean calculation
s=sqrt(1/(n-1)*sum((x-repmat(xbar,1,3)).^2,2)); //
   standard deviation calculation
p=length(xbar);//no. of subgroups
xbarbar=sum(xbar,1)/p;
sbar=sum(s,1)/p;

c4=0.8862;B3=0;B4=2.568;c=3;
sigma=1/c4*sbar/sqrt(n);
//original limits
UCL_x=xbarbar+c*sigma;//Eqn21-9
LCL_x=xbarbar-c*sigma;//Eqn 21-10

UCL_s=B4*sbar;//Eqn21-14
LCL_s=B3*sbar;//Eqn21-15

//Modified Limits
x_mod=x;
x_mod([5 15],:)=[];
n=3;
xbar_mod=sum(x_mod,2)/n; //mean calculation
```

```
s_mod=sqrt(1/(n-1)*sum((x_mod-repmat(xbar_mod,1,3)).^2,2)); //standard deviation calculation
p_mod=length(xbar_mod);//no. of subgroups
xbarbar_mod=sum(xbar_mod,1)/p_mod;
sbar_mod=sum(s_mod,1)/p_mod;

c4=0.8862;B3=0;B4=2.568;c=3;
sigma_mod=1/c4*sbar_mod/sqrt(n);
//modified limits
UCL_x_mod=xbarbar_mod+c*sigma_mod;//Eqn21-9
LCL_x_mod=xbarbar_mod-c*sigma_mod;//Eqn 21-10

UCL_s_mod=B4*sbar_mod;//Eqn21-14
LCL_s_mod=B3*sbar_mod;//Eqn21-15



mprintf('\n                    Original Limits     Modified Limits')
mprintf('\n xbar Chart Control Limits ')
mprintf('\n UCL                %f               %f',UCL_x,UCL_x_mod)
mprintf('\n LCL                %f               %f',LCL_x,LCL_x_mod)
mprintf('\n s Chart Control Limits ')
mprintf('\n UCL                %f                %f',UCL_s,UCL_s_mod)
mprintf('\n LCL                %f                 %f',LCL_s,LCL_s_mod)

subplot(2,1,1);
plot2d(repmat(UCL_x,1,p));
plot(repmat(UCL_x_mod,1,p),'--');
plot2d(repmat(LCL_x,1,p));
plot(repmat(LCL_x_mod,1,p),'--');
plot2d(xbar,style=-1,rect=[0,160,30,260])
xtitle('Example 21.2','Sampling number','$\text{Thickness}\ (\AA)$')
```

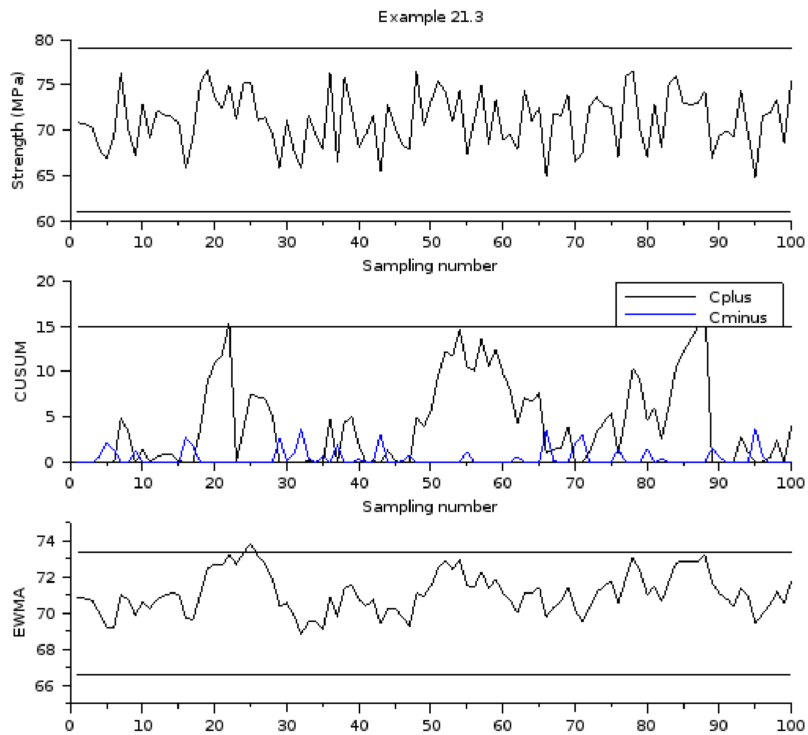


Figure 21.2: Shewart CUSUM EWMA comparison

```
legend('Original Limits','Modified Limits')

subplot(2,1,2);
plot2d(repmat(UCL_s,1,p));
plot2d(repmat(LCL_s,1,p));
plot(repmat(UCL_s_mod,1,p),'--');
plot(repmat(LCL_s_mod,1,p),'--');
plot2d(s,style=-1,rect=[0,0,30,30])
xtitle('','Sampling number','$\text{Standard
    Deviation}\ (\AA)$')
```

Scilab code Exa 21.3 Shewart CUSUM EWMA comparison

```
clear
clc

//Example 21.3
disp('Example 21.3')


T=70;sigma=3;p=100;//p is the no. of samples
x=grand(p,1, "nor", T, sigma);
delta=0.5*sigma;
x(11:$)=x(11:$)+delta;

//Limits for Shewart charts
UCL_1=T+sigma*3;
LCL_1=T-sigma*3;

subplot(3,1,1);
plot2d(repmat(UCL_1,1,p));
plot2d(repmat(LCL_1,1,p));
plot2d(x,style=1,rect=[0,60,100,80])
xtitle('Example 21.3','Sampling number','Strength (
    MPa)')

//CUSUM
Cplus=zeros(100,1);Cminus=zeros(100,1);
K=0.5*sigma;H=5*sigma;
UCL_2=H;

for k=2:100
    Cplus(k)=max(0,x(k)-(T+K)+Cplus(k-1));
    Cminus(k)=max(0,(T-K)-x(k)+Cminus(k-1));
    if Cplus(k-1)>H then
        Cplus(k)=0;
    end
    if Cminus(k-1)>H then
        Cminus(k)=0;
```

```
    end

end


subplot(3,1,2);
plot2d(Cplus,style=1,rect=[0,0,100,20]);
plot2d(Cminus,style=2,rect=[0,0,100,20]);
plot2d(repmat(UCL_2,1,p));
xtitle('','Sampling number','CUSUM')
legend('Cplus','Cminus')

//EWMA
lamda=0.25;
z=x;
for k=2:100
    z(k)=lamda*x(k)+(1-lamda)*z(k-1);
end
UCL_3=T+3*sigma*sqrt(lamda/(2-lamda));
LCL_3=T-3*sigma*sqrt(lamda/(2-lamda));

subplot(3,1,3);
plot2d(repmat(UCL_3,1,p));
plot2d(repmat(LCL_3,1,p));
plot2d(z,style=1,rect=[0,65,100,75])
xtitle('','Sampling number','EWMA')


mprintf('The charts in the example and in the book differ due\n...
a different realization of data everytime the code is run\n...
due to the grand command. If we had the exact data as that given\n...
in the book our charts would have matched.')
```

Scilab code Exa 21.4 Process Capability Indices

```

clear
clc

//Example 21.4
disp('Example 21.4')

xbar=199.5;//Note that this is the correct value and not 199
sbar=8.83;
USL=235;//Note that this is diff from UCL
LSL=185;
c4=0.8862;
n=3;
sigma=5.75;
sigma_x=sbar/c4/sqrt(n);

mprintf('\nValue of sigma_x=%f',sigma_x);

Cp=(USL-LSL)/6/sigma;
Cpk=min(xbar-LSL,USL-xbar)/3/sigma;

mprintf('\nCp=%f and Cpk=%f',Cp,Cpk)
```

Scilab code Exa 21.5 Effluent Stream from wastewater treatment

```
clear
clc

//Example 21.5
```

```
disp('Example 21.5')

//data
x=[ 17.7    1380.
    23.6    1458.
    13.2    1322.
    25.2    1448.
    13.1    1334.
    27.8    1485.
    29.8    1503.
    9.      1540.
    14.3    1341.
    26.     1448.
    23.2    1426.
    22.8    1417.
    20.4    1384.
    17.5    1380.
    18.4    1396.
    16.8    1345.
    13.8    1349.
    19.4    1398.
    24.7    1426.
    16.8    1361.
    14.9    1347.
    27.6    1476.
    26.1    1454.
    20.     1393.
    22.9    1427.
    22.4    1431.
    19.6    1405.
    31.5    1521.
    19.9    1409.
    20.3    1392.];


T=mean(x,'r');
s=sqrt(variance(x,'r'));

```

```
UCL=T+3*s;
LCL=T-3*s;

p=size(x,1)

subplot(2,1,1);
plot2d(repmat(UCL(1),1,p));
plot2d(repmat(LCL(1),1,p));
plot2d(repmat(T(1),1,p));
plot2d(x(:,1),style=-1,rect=[0,0,32,40])
xtitle('Example 21.4','Sampling number','BOD (mg/L)')


subplot(2,1,2);
plot2d(repmat(UCL(2),1,p));
plot2d(repmat(LCL(2),1,p));
plot2d(repmat(T(2),1,p));
plot2d(x(:,2),style=-1,rect=[0,1200,32,1600])
xtitle('','Sampling number','Solids (mg/L)')

//subplot(3,1,3);
scf()
plot2d(x(8,1),x(8,2),style=-3,rect=[0,1200,40,1600])
plot2d(x(:,1),x(:,2),style=-1,rect=[0,1200,40,1600])
legend("Sample #8",position=4)
xtitle('','BOD (mg/L)','Solids (mg/L)')

mprintf('\nThe confidence interval for third case is not drawn\n...
    because it is beyond the scope of this book')
```

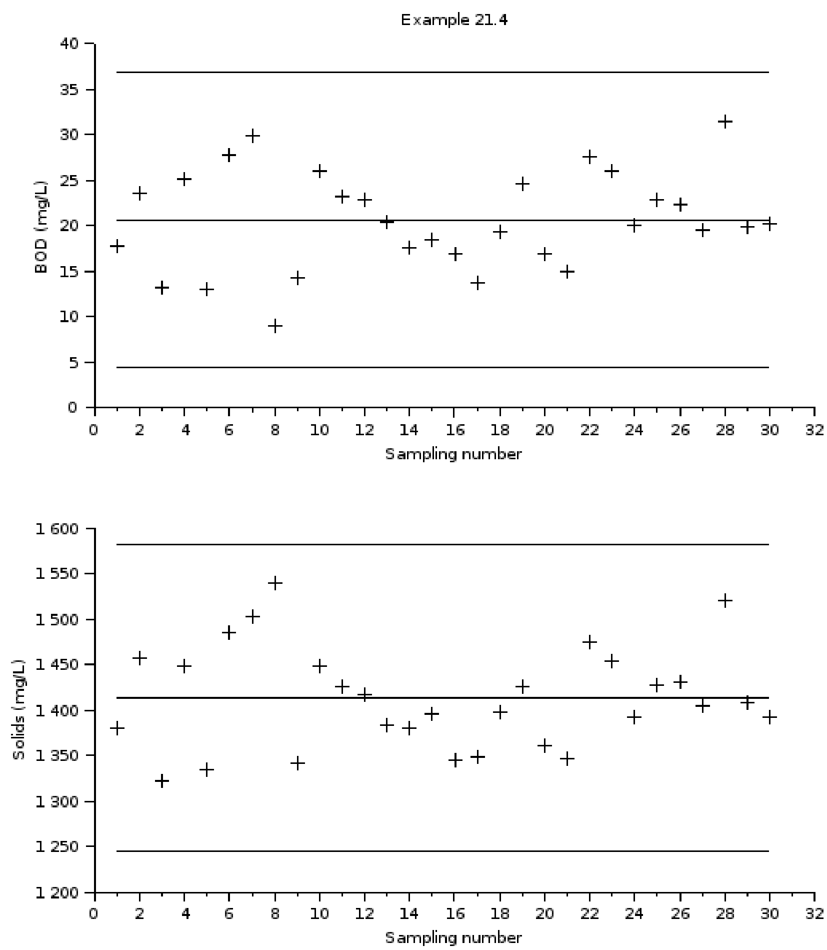


Figure 21.3: Effluent Stream from wastewater treatment

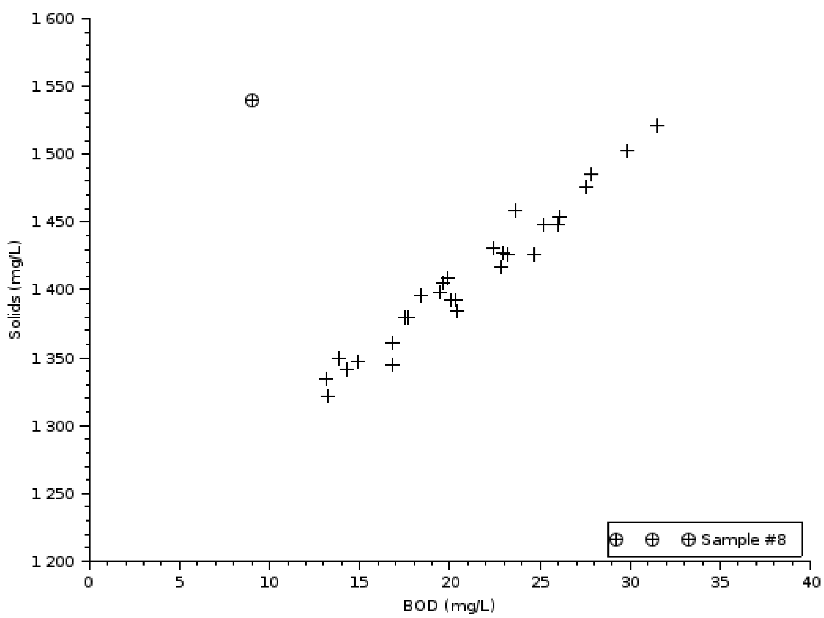


Figure 21.4: Effluent Stream from wastewater treatment

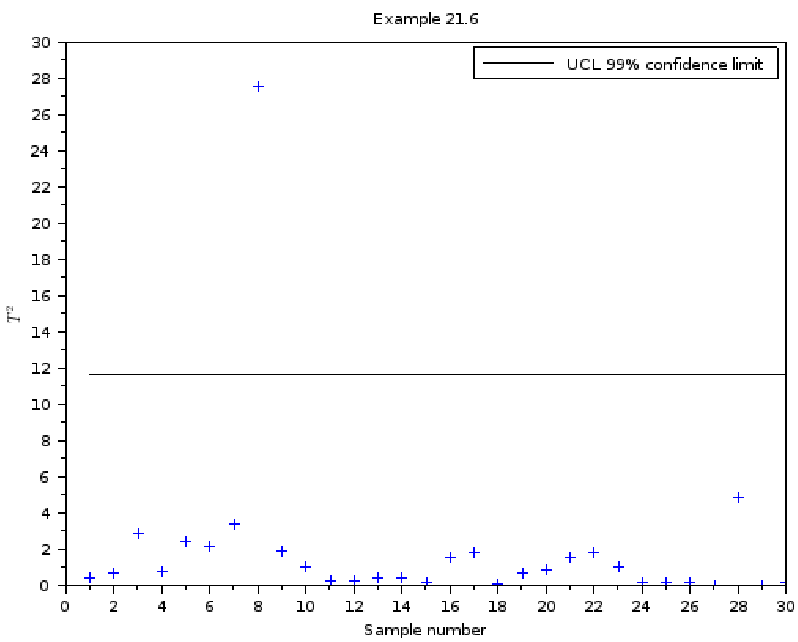


Figure 21.5: Hotellings T square statistic

Scilab code Exa 21.6 Hotellings T square statistic

```
clear
clc

//Example 21.6
disp('Example 21.6')

//data
x=[ 17.7    1380.
    23.6    1458.
    13.2    1322.
    25.2    1448.
    13.1    1334.
    27.8    1485.
    29.8    1503.
    9.      1540.
    14.3    1341.
    26.     1448.
    23.2    1426.
    22.8    1417.
    20.4    1384.
    17.5    1380.
    18.4    1396.
    16.8    1345.
    13.8    1349.
    19.4    1398.
    24.7    1426.
    16.8    1361.
    14.9    1347.
    27.6    1476.
    26.1    1454.
    20.     1393.
    22.9    1427.
```

```
    22.4    1431.
    19.6    1405.
    31.5    1521.
    19.9    1409.
    20.3    1392.];


n=1;
N=size(x,1);
T=mean(x,'r');
//For our example n=1 because each measurement is a subgroup
S=mvvacov(x);
//Note that mvvacov calculates covariance with denominator N, while
//variance caluclates with denominator N-1, hence diagonal entry of mvvacov does not
//match with variance calculated manually for each vector
//As per wikipedia the book is wrong and for covariance matrix we should
//use N-1 but here we follow the book
Tsquare=zeros(N,1);
for k=1:N
    Tsquare(k)=n*(x(k,:)-T)*inv(S)*(x(k,:)-T)';
end

UCL=11.63;

plot(repmat(UCL,1,N),color='black');
plot(Tsquare,'+')
legend("UCL 99% confidence limit")
xtitle("Example 21.6","Sample number","$T^2$")
```

Chapter 22

Biosystems Control Design

Scilab code Exa 22.1 Fermentor

```
clear
clc

//Example 22.1
disp('Example 22.1')

//Parameters
Yxs=0.4;B=0.2;Pm=50;Ki=22;
a=2.2;mu_m=0.48;Km=1.2;Sf=20;


//ODE model
function ydot=model(t,y,D)
    X=y(1);S=y(2);P=y(3);

    Xdot=-D*X+mu(S,P)*X;
    Sdot=D*(Sf-S)-1/Yxs*mu(S,P)*X;
    Pdot=-D*P+[a*mu(S,P)+B]*X

```

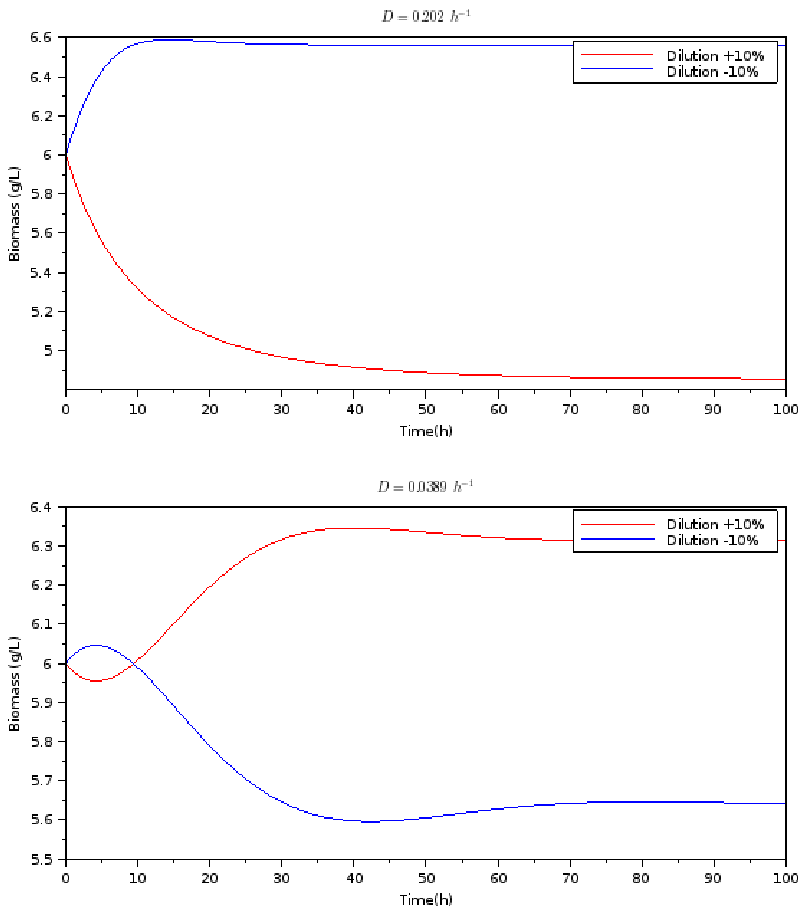


Figure 22.1: Fermentor

```
    ydot=[Xdot Sdot Pdot]';
endfunction

//Rate law
function mu=mu(S,P)
    mu=mu_m*(1-P/Pm)*S/(Km+S+S^2/Ki);
endfunction

t=0:0.1:100;t0=0;
y0=[6 5 19.14]';//Initial stable condition

D=0.202*1.1;//10% increase
y_up = ode(y0, t0, t, list(model,D))
D=0.202*0.9;//10% decrease
y_down = ode(y0, t0, t, list(model,D))

subplot(2,1,1);
plot(t,y_up(1,:),color='red');
plot(t,y_down(1,:));
xtitle("$D=0.202\ h^{-1}$","Time(h)","Biomass (g/L)")
legend("Dilution +10%","Dilution -10%")

subplot(2,1,2);
t=0:0.1:100;t0=0;
y0=[6 5 44.05]';//Initial stable condition
D=0.0389*1.1;//10% increase
y_up = ode(y0, t0, t, list(model,D))
D=0.0389*0.9;//10% decrease
y_down = ode(y0, t0, t, list(model,D))

plot(t,y_up(1,:),color='red');
plot(t,y_down(1,:))
xtitle("$D=0.0389\ h^{-1}$","Time(h)","Biomass (g/L)");
legend("Dilution +10%","Dilution -10%")
```

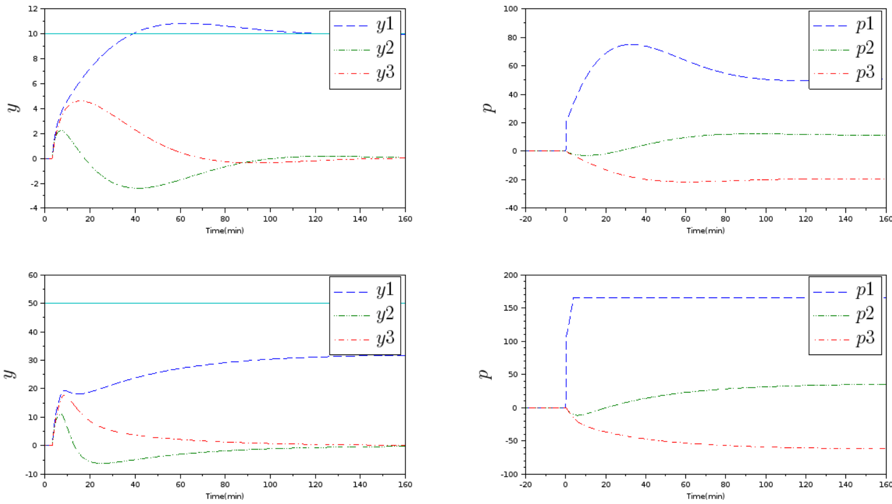


Figure 22.2: Granulation process control

Scilab code Exa 22.2 Granulation process control

```
clear
clc

//Example 22.2
disp('Example 22.2')
//Author: Dhruv Gupta....Aug 4, 2013
//<dgupta6@wisc.edu>

K=[0.2 0.58 0.35;0.25 1.10 1.3;0.3 0.7 1.2];
tau=[2 2 2;3 3 3;4 4 4];
s=%s;

G=K./(1+tau*s);
```

```

RGA=K.*inv(K');
disp(RGA,"RGA=")

//IMC based tuning
tauC=5;
Kc=diag(tau/tauC./K);
mprintf("\n\nThe tauI given in book are wrong\n...
refer to Table 11.1 for correct formula\n\n")
tauI=diag(tau)+1;
mprintf('\nWe still however use the ones given in book\n');


disp(Kc,"Kc=")
disp(tauI,"tauI=")
//Refer to Eqns 15-23 and 15-24
Gc=Kc.*(1+(1)./tauI/s);
//For the sake of brevity we write Gstar as G
//We will account for delays in the for loop that we will write
//Refer to Figure 15.9 Page 295 for details of Smith Predictor


//====Making step response models of the continuos transfer functions====//
Ts=0.1;//Sampling time ie delta_T
delay=3/Ts;
N=150/Ts;//Model Order
s=%s;
G=syslin('c',diag(matrix(G,1,9)));//Transfer function
t=0:Ts:N*Ts;
u_sim=ones(9,length(t));
//u_sim(:,1:4)=zeros(9,4); //input delay to account for 3 min delay in G
S=csim(u_sim,t,G)';//generating step response model
```

```
for real plant
//plot(t,S);
S(1,:)=[];
//Now we have these step response models for each of the transfer functions
//[S1 S4 S7
//S2 S5 S8
//S3 S6 S9]

T=150+delay;//Simulation Run Time in minutes(we add delay because our for loop runs till n-delay)
n=T/Ts*2+1; //no. of discrete points in our domain of analysis
//Input initialization T is the Time for simulation

//==========Set point as 10==============//
//p is the controller output
p=zeros(n,3);
delta_p=zeros(n,3);
ytilde=zeros(n,3); //Prediction of Smith Fig 15.9
e=zeros(n,3); //corrections
edash=zeros(n,3);
delta_edash=zeros(n,3);
ysp=zeros(n,3);
ysp((n-1)/2+1:n,1)=10*ones(n-((n-1)/2+1)+1,1);

t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
y=zeros(n,3);

for k=(n-1)/2+1:n-delay

    //Error  e
    e(k,:)=ysp(k-1,:)-y(k-1,:);

    //Error edash
    edash(k,:)=e(k-1,:)-ytilde(k-1,:)+ytilde(k-1-delay,:);
    //Edash=E-(Y1-Y2)...where Y2 is delayed Y1
```

```
    delta_edash(k,:)=edash(k,:)-edash(k-1,:);

    //Controller calculation----Digital PID-----Eqn
       7-28 Pg 136 (Velocity form)
    p(k,:)=p(k-1,:)+[delta_edash(k,:)+edash(k,:)*
       diag(Ts./tauI)]*diag(Kc);

    //Limits on manipulated variables
    p(k,:)=min([(345-180)*ones(1,3);p(k,:)],'r');
    p(k,:)=max([(105-180)*ones(1,3);p(k,:)],'r');

    delta_p(k,:)=p(k,:)-p(k-1,:);


    //Prediction
    ytilde(k,1)=[S(1:N-1,1);S(1:N-1,4);S(1:N-1,7)
       ]'...
        *[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)]...
        +[S(N,1) S(N,4) S(N,7)]*[p(k-N,1);p(k-N,2);p
           (k-N,3)];
    ytilde(k,2)=[S(1:N-1,2);S(1:N-1,5);S(1:N-1,8)
       ]'...
        *[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)]...
        +[S(N,2) S(N,5) S(N,8)]*[p(k-N,1);p(k-N,2);p
           (k-N,3)];
    ytilde(k,3)=[S(1:N-1,3);S(1:N-1,6);S(1:N-1,9)
       ]'...
        *[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)]...
        +[S(N,3) S(N,6) S(N,9)]*[p(k-N,1);p(k-N,2);p
           (k-N,3)];

    //Output
```

```
    y(k+delay,1)=[S(1:N-1,1);S(1:N-1,4);S(1:N-1,7)]'...
        *[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)]...
        +[S(N,1) S(N,4) S(N,7)]*[p(k-N,1);p(k-N,2);p(k-N,3)];
    y(k+delay,2)=[S(1:N-1,2);S(1:N-1,5);S(1:N-1,8)]'...
        *[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)]...
        +[S(N,2) S(N,5) S(N,8)]*[p(k-N,1);p(k-N,2);p(k-N,3)];
    y(k+delay,3)=[S(1:N-1,3);S(1:N-1,6);S(1:N-1,9)]'...
        *[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)]...
        +[S(N,3) S(N,6) S(N,9)]*[p(k-N,1);p(k-N,2);p(k-N,3)];
end


subplot(2,2,1);
plot(t',y(:,1),'--',t',y(:,2),':',t',y(:,3),'-.',t',ysp(:,1),'-');
set(gca(),"data_bounds",[0 150 -4 12]); //putting bounds on display
l=legend("$y1$","$y2$","$y3$",position=1);
l.font_size=5;
xtitle("","Time(min)","$y$");
a=get("current_axes");
c=a.y_label;c.font_size=5;


subplot(2,2,2);
plot(t',p(:,1),'--',t',p(:,2),':',t',p(:,3),'-.');
```

```
set(gca(),"data_bounds",[-1 150 -40 100]); //putting bounds on display
l=legend("$p1$","$p2$","$p3$",position=1);
l.font_size=5;
xtitle("","Time(min)","$p$");
a=get("current_axes");
c=a.y_label;c.font_size=5;

mprintf("Note that there is no overshoot around time=25 mins \n...
which is in contrast to what is shown in book")


//==========Now for set point as 50==============//

//p is the controller output
p=zeros(n,3);
delta_p=zeros(n,3);
ytilde=zeros(n,3); //Prediction of Smith Fig 15.9
e=zeros(n,3); //corrections
edash=zeros(n,3);
delta_edash=zeros(n,3);
ysp=zeros(n,3);
ysp((n-1)/2+1:n,1)=50*ones(n-((n-1)/2+1)+1,1);

t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
y=zeros(n,3);

for k=(n-1)/2+1:n-delay

    //Error  e
    e(k,:)=ysp(k-1,:)-y(k-1,:);

    //Error edash
    edash(k,:)=e(k-1,:)-ytilde(k-1,:)+ytilde(k-1-delay,:);
    //Edash=E-(Y1-Y2)...where Y2 is delayed Y1
    delta_edash(k,:)=edash(k,:)-edash(k-1,:);
```

```
    //Controller calculation———Digital PID———Eqn 7−28 Pg 136 (Velocity form)
    p(k,:)=p(k-1,:)+[delta_edash(k,:)+edash(k,:)*diag(Ts./tauI)]*diag(Kc);

    //Limits on manipulated variables
    p(k,:)=min([(345-180)*ones(1,3);p(k,:)],'r');
    p(k,:)=max([(105-180)*ones(1,3);p(k,:)],'r');

    delta_p(k,:)=p(k,:)-p(k-1,:);


    //Prediction
    ytilde(k,1)=[S(1:N-1,1);S(1:N-1,4);S(1:N-1,7)]'...
        *[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)]...
        +[S(N,1) S(N,4) S(N,7)]*[p(k-N,1);p(k-N,2);p(k-N,3)];
    ytilde(k,2)=[S(1:N-1,2);S(1:N-1,5);S(1:N-1,8)]'...
        *[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)]...
        +[S(N,2) S(N,5) S(N,8)]*[p(k-N,1);p(k-N,2);p(k-N,3)];
    ytilde(k,3)=[S(1:N-1,3);S(1:N-1,6);S(1:N-1,9)]'...
        *[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)]...
        +[S(N,3) S(N,6) S(N,9)]*[p(k-N,1);p(k-N,2);p(k-N,3)];

    //Output
    y(k+delay,1)=[S(1:N-1,1);S(1:N-1,4);S(1:N-1,7)
```

```
]'...
        *[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)]...
        +[S(N,1) S(N,4) S(N,7)]*[p(k-N,1);p(k-N,2);p(k-N,3)];
    y(k+delay,2)=[S(1:N-1,2);S(1:N-1,5);S(1:N-1,8)]'...
        *[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)]...
        +[S(N,2) S(N,5) S(N,8)]*[p(k-N,1);p(k-N,2);p(k-N,3)];
    y(k+delay,3)=[S(1:N-1,3);S(1:N-1,6);S(1:N-1,9)]'...
        *[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)]...
        +[S(N,3) S(N,6) S(N,9)]*[p(k-N,1);p(k-N,2);p(k-N,3)];
end


subplot(2,2,3);
plot(t',y(:,1),'--',t',y(:,2),':',t',y(:,3),'-.',t',ysp(:,1),'-');
set(gca(),"data_bounds",[0 150 -10 60]); //putting bounds on display
l=legend("$y1$","$y2$","$y3$",position=1);
l.font_size=5;
xtitle("","Time(min)","$y$");
a=get("current_axes");
c=a.y_label;c.font_size=5;


subplot(2,2,4);
plot(t',p(:,1),'--',t',p(:,2),':',t',p(:,3),'-.');
set(gca(),"data_bounds",[-1 150 -100 200]); //
```

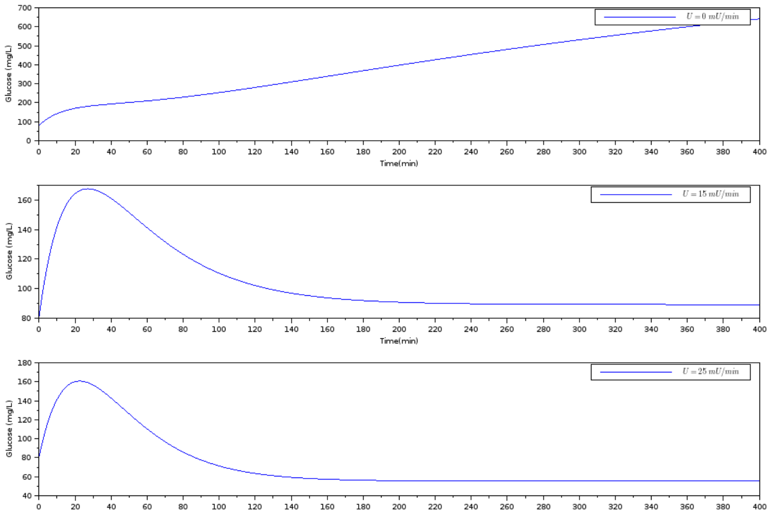


Figure 22.3: Type 1 Diabetes

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

Scilab code Exa 22.3 Type 1 Diabetes

```

clear
clc

//Example 22.3
disp('Example 22.3')

```

```
//Parameters
p1=0.028735;p2=0.028344;p3=5.035E-5;V1=12;n=0.0926;
Ib=15;//basal
Gb=81;

//Diet function
function D=D(t)
    D=9*exp(-0.05*t);
endfunction


//ODE model
function ydot=model(t,y,U)
    G=y(1);X=y(2);I=y(3);
    Gdot=-p1*G-X*(G+Gb)+D(t);
    Xdot=-p2*X+p3*I;
    Idot=-n*(I+Ib)+U/V1;
    ydot=[Gdot Xdot Idot]';
endfunction


t=0:0.1:400;t0=0;
y0=[0 0 0]';//G,X,I are deviation variables

U=0;
y = Gb+ode(y0, t0, t, list(model,U))
subplot(3,1,1);
plot(t,y(1,:));
xtitle("","Time(min)","Glucose (mg/L)")
legend("$U=0\ mU/min$")

U=15;
y =Gb+ ode(y0, t0, t, list(model,U))
subplot(3,1,2);
plot(t,y(1,:));
xtitle("","Time(min)","Glucose (mg/L)")
legend("$U=15\ mU/min$")

```

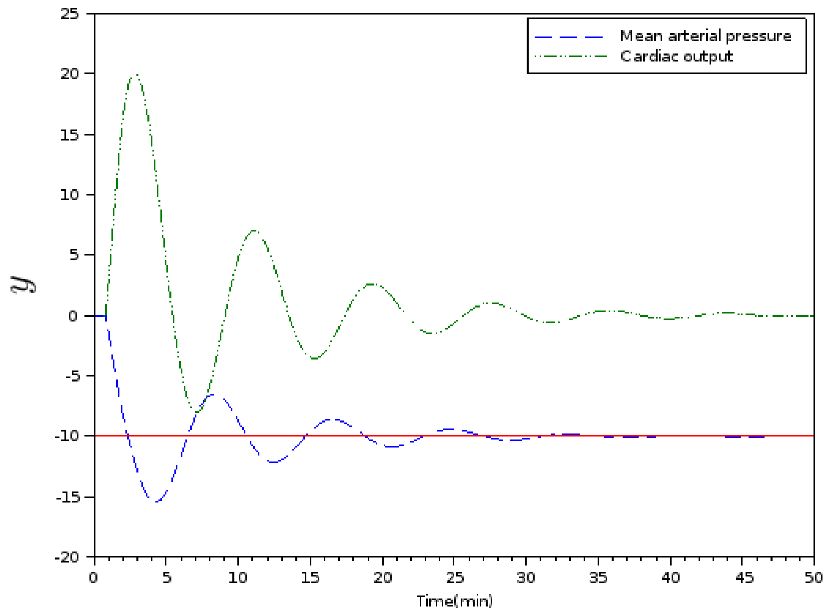


Figure 22.4: Influence of drugs

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

Scilab code Exa 22.4 Influence of drugs

```
clear
clc
```

```

//Example 22.4
disp('Example 22.4')
K=[-6 3;12 5];
tau=[0.67 2;0.67 5];
s=%s;
G=K./(1+tau*s);
delay75=0.75;
delay1=1;
RGA=K.*inv(K');
disp(RGA,"RGA=")

//IMC based tuning
tauC=[tau(1,1) tau(2,2)];
Kc=diag(tau./(repmat(tauC,2,1)+[delay75 delay1; delay75 delay1])./K);
tauI=diag(tau);
disp(Kc,"Kc=")
disp(tauI,"tauI=")

//====Making step response models of the continuos transfer functions====//
Ts=0.05;//Sampling time ie delta_T
delay75=0.75/Ts;
delay1=1/Ts;
N=30/Ts;//Model Order
s=%s;
G=syslin('c',diag(matrix(G,1,4)));//Transfer function
t=0:Ts:N*Ts;
u_sim=ones(4,length(t));
u_sim([1 2],1:(delay75))=zeros(2,delay75); //input delay to account for delay in SNP
u_sim([3 4],1:(delay1))=zeros(2,delay1); //input delay to account for delay in DPM
S=csim(u_sim,t,G)';//generating step response model for real plant
//plot(t,S);
```

```
S(1,:)=[];
//Now we have these step response models for each of the transfer functions
//[S1 S3
//S2 S4




T=50;//Simulation Run Time in minutes
n=T/Ts*2+1; //no. of discrete points in our domain of analysis


//==========Set point as -10==============//
//p is the controller output
p=zeros(n,2);
delta_p=zeros(n,2);
e=zeros(n,2); //errors=(ysp-y) on which PI acts
ysp=zeros(n,2);
ysp((n-1)/2+1:n,1)=-10*ones(n-((n-1)/2+1)+1,1);

t=-(n-1)/2*Ts:Ts:(n-1)/2*Ts;
y=zeros(n,2);


for k=(n-1)/2+1:n

    //Error  e
    e(k,:)=ysp(k-1,:)-y(k-1,:);
    delta_e(k,:)=e(k,:)-e(k-1,:);

    //Controller calculation----Digital PID----Eqn 7-28 Pg 136 (Velocity form)
//     p(k,:)=p(k-1,:)+flipdim([delta_e(k,:)+e(k,:)*diag(Ts./tauI)]*diag(Kc),2);
    p(k,:)=p(k-1,:)+([delta_e(k,:)+e(k,:)*diag(Ts./tauI)]*diag(Kc));
```

```
    //1-1/2-2 pairing

    delta_p(k,:)=p(k,:)-p(k-1,:);

    //Output
    y(k,1)=[S(1:N-1,1);S(1:N-1,3)]'...
        *[flipdim(delta_p(k-N+1:k-1,1),1);flipdim(delta_p(k-N+1:k-1,2),1)]...
        +[S(N,1) S(N,3)]*[p(k-N,1);p(k-N,2)];
    y(k,2)=[S(1:N-1,2);S(1:N-1,4)]'...
        *[flipdim(delta_p(k-N+1:k-1,1),1);flipdim(delta_p(k-N+1:k-1,2),1);]...
        +[S(N,2) S(N,4)]*[p(k-N,1);p(k-N,2)];
end


plot(t',y(:,1),'--',t',y(:,2),':',t',ysp(:,1),'-');
set(gca(),"data_bounds",[0 50 -20 25]); //putting bounds on display
l=legend("Mean arterial pressure","Cardiac output",position=1);
//l.font_size=5;
xtitle("","Time(min)","$y$");
a=get("current_axes");
c=a.y_label;c.font_size=5;

mprintf('\nThere is more interaction in the variables\n...
than the book claims, hence a mismatch between the result\n...
and the book\n')
```