Scilab Textbook Companion for Feedback Control of Dynamic Systems by G. F. Franklin, J. D. Powell and A. Emami-Naeini[1]

Created by
Mrs. Deepti Khimani
Feeback Control System, Control System Design
Instrumentation Engineering
Mumbai University
College Teacher
None
Cross-Checked by
Spandana

June 7, 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: Feedback Control of Dynamic Systems

Author: G. F. Franklin, J. D. Powell and A. Emami-Naeini

Publisher: Pearson Education and Dorling Kindersley

Edition: 5

Year: 2010

ISBN: 978-81-317-2142-1

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

Chapter 2

Dynamic Models

Scilab code Exa 2.1.b step response of Cruise control system

```
//Example 2.1
//(b) step response of Cruise control system

xdel(winsid())//close all graphics Windows
clear;
clc;

//
  ---------------------------------------------------------------------------------------------------------

//Cruise control parameters
m=1000;
b=50;
u=500;

// Transfer function
s=%s; // or
s=poly(0,'s');
sys=syslin('c',(1/m)/(s+b/m))

//step response to u=500;
```

```
t=0:0.5:100;
v=csim('step',t,u*sys);
plot2d(t,v,2)

//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title('Responses of car velocity to a step in u','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('Amplitude','fontsize',2)

//-------------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 2.5.b step response of pendulum

```
//Example 2.5
//(b) step response of pendulum

xdel(winsid())//close all graphics Windows
clear;
clc;

//-------------------------------------------------------------------------------
//Pendulum parameters
m=0.5;
```

```
l=1;
g=9.81;

// Transfer function
s=%s;
sys=syslin('c',(1/(m*l^2))/(s^2+g/l));

//step response to u=500;
t=0:0.02:10;
theta=csim('step',t,sys);
plot(t,theta*57.3);

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script to set figure properties
title('Response of pendulum to a step input in the applied torque',...
'fontsize',3);
xlabel('Time t (sec.)','fontsize',2);
ylabel('Pendulum angle (degree)','fontsize',2);

//______________________________________________________________________
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Chapter 3

Dynamic Responses

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 3.4 Frequency response

```
//Example 3.4
//Frequency response

xdel(winsid())//close all graphics Windows
clear;
clc;
//
    ---------------------------------------------------------------------------------
//(a) Frequency response of 1/(s+k)
k=1;
fmin=1e-2;
fmax=1e2;
// Transfer function
s=poly(0,'s');
sysH=syslin('c',1/(s+k))

```

```
//Frequency response for k=1
//Note that - magnitude plot semilog plot unlike log-log plot in the book.
bode(sysH,fmin,fmax)
title('Frequency response for k=1','fontsize',3)

//--------------------------------------------------------------------------------------------------------
//(b) Response to u=sin(10*t);
t=0:0.02:10;
u=sin(10*t);
y=csim(u,t,sysH);
figure, plot(t,y)

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Complete transient response','fontsize',3)
xlabel('Time (sec.)','fontsize',2)
ylabel('Output','fontsize',2)

//phase lag
figure, plot(t,y)
plot(t,u,'r')
zoom_rect([9 -1 10 1])
exec .\fig_settings.sci; // custom script for setting figure properties
title('Phase lag between output and input','fontsize',3)
xlabel('Time (sec.)','fontsize',2)
ylabel('Output, Input','fontsize',2)
h=legend('y(t)','u(t)')
h.legend_location = "in_upper_right"
h.fill_mode='off'

// time lag
w=find(t>=9.4 & t<=10);
```

```
T=t(w);
Y=y(w);
U=u(w);
wu=find(U==max(U))
wy=find(Y==max(Y))

//Responses
plot2d3(T(wy),Y(wy))
plot2d3(T(wu),U(wu))
delta_t=T(wu)-T(wy); //time lag sec.
xstring(9.64,-0.1,"$\delta t$",0,0)
xarrows([9.58;9.72], [0;0], 0.7, 1)
xarrows([9.72;9.58], [0;0], 0.7, 1)
t=get("hdl")
disp(abs(delta_t), "Time lag of output in sec. is")
disp(abs(delta_t)*10, "Phase lag of output in radians is")

//------------------------------------------------------------------------
```

Scilab code Exa 3.8 Partial fraction expansion for distinct real roots

```
//Example 3.8
//Partial fraction expansion for distinct real roots.

clear;
clc;
//
```

```
    ____________________________________________________________________
//Partial fraction expansion for distinct real roots
// Transfer function
s=%s;
num=(s+2)*(s+4)
p1=s;
p2=(s+1);
p3=(s+3);
sys=syslin('c',num/(p1*p2*p3))
//
    ____________________________________________________________________
//Partial fraction expansion is: sys= r1/p1 + r2/p2
    + r3/p3
//residue calculation
r1=residu(num,p1,(p2*p3))
r2=residu(num,p2,(p1*p3))
r3=residu(num,p3,(p1*p2))

disp([r1 r2 r3]',"Residues of the poles p1, p2 and
    p3 are")
disp([roots(p1), roots(p2), roots(p3)]',"Poles p1,
    p2 and p3 are at")
disp('k=[]')

//
    ____________________________________________________________________
```

Scilab code Exa 3.9 Final value theorem

```
//Example 3.9
//Computing final value (use of final value theorem)
    .
```

```

clear;
clc;

// -----------------------------------------------------------------------------

//Computing final value (use of final value theorem)
// Output of the system
s=poly(0,'s');
num=3*(s+2);
den=s*(s^2+2*s+10);
Ys=syslin('c',num/den);


//final value theorem, lim s-->0 in s*Y(s)

Y_final=horner(s*Ys,0)
disp(Y_final,"The final value of the output y is:")

// -----------------------------------------------------------------------------
```

Scilab code Exa 3.10 Incorrect use of final value theorem

```
//Example 3.10

//Computing final value for unstable system to show the incorrect
// use of final value theorem.
clear;
clc;
//
```

```
------------------------------------------------------------------------
s=poly(0,'s');
num=3;
den=s*(s-2);
Ys=syslin('c',num/den);

//final value theorem, lim s-->0 in s*Y(s)
Y_final=horner(s*Ys,0);
disp(Y_final,"The final value of the output y is:");
disp('The final value computed is incorrect as the system...
 response is unbounded');
//------------------------------------------------------------------------
```

Scilab code Exa 3.11 DC gain of the system

```
//Example 3.11
//Computing DC gain of the system.

clear;
clc;
//------------------------------------------------------------------------
//Transfer Function
s=poly(0,'s');
num=3*(s+2);
den=(s^2+2*s+10);
Ys=syslin('c',num/den);

//The DC gain of the system Y(s) as s-->0 is
DC_Gain=horner(Ys,0)
```

```
disp(DC_Gain,"The DC gain of the system is:")
//
  ------------------------------------------------------------------------------
```

Scilab code Exa 3.14 Partial fraction expansion for distinct real roots

```
//Example 3.14
//Partial fraction expansion for distinct real roots
clear;
clc;
//
  ------------------------------------------------------------------------------
// Transfer function
s=%s;
num=2;
p1=(s+1);
p2=(s+2);
p3=(s+4);
sys=syslin('c',num/(p1*p2*p3))

//Partial fraction expansion is: sys= r1/p1 + r2/p2
   + r3/p3
//residue calculation
r1=residu(num,p1,(p2*p3))
r2=residu(num,p2,(p1*p3))
r3=residu(num,p3,(p1*p2))

disp([r1 r2 r3]',"Residues of the poles p1, p2 and
   p3 are")
disp([roots(p1), roots(p2), roots(p3)]',"Poles p1,
   p2 and p3 are at")
disp('k=[]')
//
```

```
______________________________________________________________________
```

Scilab code Exa 3.15 Cruise Control Transfer Function

```
//Example 3.15 Cruise Control Transfer Function.
//Coefficients of numerator and denominator of the
   transfer function

clear;
clc;
//
   ______________________________________________________________________

// Transfer function coefficients
num=[0.001 0];
den=[0 0.05 1];

// Transfer function
Ns=poly(num,'s','coeff');
Ds=poly(den,'s','coeff');
sys=syslin('c',Ns/Ds);

//gain (K) pole (P) and zeros (Z) of the system
temp=polfact(Ns);
Z=roots(Ns); //locations of zeros
P=roots(Ds); //locations of poles
K=temp(1); //first entry is always gain
disp( K,"Gain", P, "Poles",Z,"Zeros",)

//
   ______________________________________________________________________
```

---

check Appendix AP 1 for dependency:

fig_settings.sci

Scilab code Exa 3.16 DC Motor Transfer Function

```
//Example 3.16 DC Motor Transfer Function.

xdel(winsid())//close all graphics Windows
clear;
clc;
//-----------------------------------------------------------------------------------------------------------
//Coefficients of numerator and denominator of the transfer function
numb=[100];
denb=[0 101 10.1 1];

// Transfer function
Ns=poly(numb,'s','coeff');
Ds=poly(denb,'s','coeff');
sysb=syslin('c',Ns/Ds);

//gain (K) pole (P) and zeros (Z) of the system
temp=polfact(Ns);
Z=roots(Ns); //locations of zeros
P=roots(Ds); //locations of poles
K=temp(1); //first entry is always gain
disp( K,"Gain", P, "Poles",Z,"Zeros",)

//Transient response of DC Motor (consider velocity as output)
s=%s;
t=linspace(0,5,501);
y=csim('step',t,sysb*s)
plot(t,y)
```

```
exec .\fig_settings.sci; //custom script for setting figure properties
title('Transient response of DC Motor','fontsize',3)
xlabel('$Time\,\, t(sec.)$','fontsize',3)
ylabel('$\omega\,\,(rad/sec)$','fontsize',3)
//-------------------------------------------------------------------------------------
```

Scilab code Exa 3.17 Transformations

```
//Example 3.17 Transformations

clear;
clc;
//-------------------------------------------------------------------------------------
//Coefficients of numerator and denominator of the transfer function
numG=[9 3];
denG=[25 6 1];

// Transfer function
Ns=poly(numG,'s','coeff');
Ds=poly(denG,'s','coeff');
sysG=syslin('c',Ns/Ds);

//gain (K) pole (P) and zeros (Z) of the system
temp=polfact(Ns);
Z=roots(Ns); //locations of zeros
P=roots(Ds); //locations of poles
K=temp(1); //first entry is always gain
```

```
disp( K,"Gain", P, "Poles",Z,"Zeros",)
//-----------------------------------------------------------------------------------
```

Scilab code Exa 3.18 Satellite Transfer Function

```
//Example 3.18 Satellite Transfer Function

xdel(winsid())//close all graphics Windows
clear;
clc;
//-----------------------------------------------------------------------------------
//(a)
//Given
d=1 //meters
I=5000 //Kg-meter^2

//Coefficients of numerator and denominator of the transfer function
// of satellite
numG=[d/I 0];
denG=[0 0 1];

// Transfer function
Ns=poly(numG,'s','coeff');
Ds=poly(denG,'s','coeff');
sysG=syslin('c',Ns/Ds);
t=0:0.01:10;
[i j]=size(t);

//-----------------------------------------------------------------------------------
```

```
//(b)
// Thrust input after 5 sec.
u=zeros(1,j);
w=find(t>=5 & t<=5+0.1);
u(w)=25;
plot(t,u);
exec .\fig_settings.sci; //custom script for setting figure properties
title("Transient response of the satellite...
 (a) Thrust input",'fontsize',3);
xlabel('Time t (sec.)','fontsize',2)
ylabel('Fc','fontsize',2)

//Transient response of the satellite to the thrust input as a pulse
sysd=dscr(sysG,0.01); //sample data system model
y=flts(u,sysd);         //impulse response
figure, plot(t,y*180/%pi);
exec .\fig_settings.sci; //custom script for setting figure properties
title("Transient response of the satellite(double-pulse)...
 (b) satellite attitude",'fontsize',3);
xlabel('Time t (sec.)','fontsize',2)
ylabel('$\theta(deg)$','fontsize',2)
//---------------------------------------------------------------------------
// Thrust input double-pulse.
u=zeros(1,j);
w1=find(t>=5 & t<=5+0.1);
u(w1)=25;
w2=find(t>=6.1 & t<=6.1+0.1);
u(w2)=-25;
figure,
plot(t,u);
exec .\fig_settings.sci; //custom script for setting
```

```
    figure properties
title("Transient response of the satellite (double-pulse)...
 (a) Thrust input",'fontsize',3);
xlabel('Time t (sec.)','fontsize',2)
ylabel('Fc','fontsize',2)

//Transient response of the satellite to the thrust input as a pulse
sysd=dscr(sysG,0.01); //sample data system model
y=flts(u,sysd);        //impulse response
figure, plot(t,y*180/%pi);
exec .\fig_settings.sci; //custom script for setting figure properties
title("Transient response of the satellite(double-pulse)...
 (b) satellite attitude",'fontsize',3);
xlabel('Time t (sec.)','fontsize',2)
ylabel('$\theta(deg)$','fontsize',2)

//
-----------------------------------------------------------------------------------------------
```

Scilab code Exa 3.21 Transfer function of a simple system

```
//Example 3.21
//Series, Parallel and Feedback connections of TF blocks
//to get effective TF.

```

```
clear;
clc;
//----------------------------------------------------------------------------------------
//Transfer function block G1
num1=[2];
den1=[1];
Ns=poly(num1,'s','coeff');
Ds=poly(den1,'s','coeff');
sysG1=syslin('c',Ns/Ds);

//Transfer function block G2
num2=[4];
den2=[0 1];
Ns=poly(num2,'s','coeff');
Ds=poly(den2,'s','coeff');
sysG2=syslin('c',Ns/Ds);

//Transfer function block G4
num4=[1];
den4=[0 1];
Ns=poly(num4,'s','coeff');
Ds=poly(den4,'s','coeff');
sysG4=syslin('c',Ns/Ds);

//Transfer function block G6
num6=[1];
den6=[1];
Ns=poly(num6,'s','coeff');
Ds=poly(den6,'s','coeff');
sysG6=syslin('c',Ns/Ds);

//Effective transfer function
// (+) operator for paralle connection,
// (*) operator for series connection
// (/.)operator for feedback connection
sysG=(sysG1 + sysG2) * sysG4 /. sysG6
```

```
disp(sysG, ”The effective transfer function is”)
//
  ------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 3.22 Response Versus Pole Locations Real Roots

```
//Example 3.22 Response Versus Pole Locations , Real
   Roots

xdel(winsid())//close all graphics Windows
clear;
clc;
//
  ------------------------------------------------------------------------

//Transfer function
numH=[1 2];
denH=[2 3 1];
Ns=poly(numH,’s’,’coeff’);
Ds=poly(denH,’s’,’coeff’);
sysH=syslin(’c’,Ns/Ds);

//Pole-zero locations
//Partial fraction method to see the effect of
   sperated poles
temp=polfact(Ds);
p1s=temp(2);
p2s=temp(3);

//residues at poles
r1=residu(Ns,p1s,p2s);
```

```
r2=residu(Ns,p2s,p1s);

//Note that - H1(s)+H2(s)=H(s)
H1s=syslin('c',r1/p1s);
H2s=syslin('c',r2/p2s);

//impulse response of the H1(s), H2(s) and H(s)
t=0:0.02:10;
h1=csim('impuls',t,H1s);
h2=csim('impuls',t,H2s);
h=csim('impuls',t,sysH);
figure,
plot(t,h1,'r--',t,h2,'m-.', t, h, 'b')
plot(t,h2,'m-.')
plot(t,h)

exec .\fig_settings.sci; //custom script for setting figure properties
title(['impulse response of the system and subsystems with...
 independent poles.';'(h1(t) is faster than h2(t))'],'fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('h(t), h1(t), h2(t)','fontsize',2)
h=legend('h1(t) with pole at -2','h2(t) with pole at -1'...
,'h(t)=h1(t)+h2(t)')
h.legend_location = "in_upper_right"
h.fill_mode='off'
//---------------------------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 3.23 Oscillatory Time Response

```
//Example 3.23 Oscillatory Time Response

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------------
//Transfer function of second order underdamped system
numH=[1 2];
denH=[5 2 1];
Ns=poly(numH,'s','coeff');
Ds=poly(denH,'s','coeff');
sysH=syslin('c',Ns/Ds);

//damping factor (xi) and natural frequency (wn)
[wn xi]=damp(sysH);
wn=wn(1);
xi=xi(1);
sigma=xi*wn;
wd=wn*sqrt(1-xi^2);

//denominator in sigma-wn form H(s)=H1(s)+H2(s)
s=%s;
p=(s+sigma)^2+wd^2
temp=polfact(Ns);
k=temp(1),zr=temp(2);
h1=(s+sigma)/p;
h2=-((s+sigma)-temp(2))*wd/p;
H1s=syslin('c',k*h1);
H2s=syslin('c',k*h2/wd);
```

```

// responses with exponential envelope
Env=syslin('c',k/(s+sigma));
t=0:0.02:10;
//impulse response
ht=csim('impuls',t,sysH);
envt=csim('impuls',t,Env);
envt_neg=csim('impuls',t,-Env);

plot(t,ht)
plot(t,envt,'r--')
plot(t,envt_neg,'r--')
exec .\fig_settings.sci; //custom script for setting figure properties
title('Impulse response of the underdamped system','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('h(t)','fontsize',2)
xset("font",1,2)
xstring(1,0.75,"$e^{-\sigma t}$",0,0)
xstring(1,-0.85,"$-e^{-\sigma t}$",0,0)
//------------------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 3.25 Aircraft Response

```
//Example 3.25 Aircraft Response
xdel(winsid())//close all graphics Windows
clear;
```

```
clc;
//--------------------------------------------------------------------------------------------------------
//(a)impulse response of aircraft

//Transfer function of aircraft
numG=[-6 1];
denG=[0 13 4 1];
Ns=30*poly(numG,'s','coeff');
Ds=poly(denG,'s','coeff');
u=-1 //impulsive elevator input of 1 degree
sysG=syslin('c',u*Ns/Ds);

//impulse response
t=0:0.02:10;
gt=csim('impuls',t,sysG);
plot(t,gt)
exec .\fig_settings.sci; //custom script for setting figure properties
title('Response of an airplanes altitude to an impulsive elevator input','fontsize',3)
xlabel('Time (sec.)','fontsize',2)
ylabel('Altitude (ft)','fontsize',2)

//final value theorem, lim s-->0 in s*G(s)
s=%s;
gt_final=horner(s*sysG,0)
disp(gt_final,"The final value of the output altitude is:")
//--------------------------------------------------------------------------------------------------------
//(b)response specifications

//damping factor (xi) and natural frequency (wn)
[wn xi]=damp(sysG);
wn=wn(2);//natural frequency (wn)
```

```
xi=xi(2);//damping factor
disp(wn,xi,"Damping factor and natural frequency (rad)...
 of the response are:")

tr=1.8/wn; //rise time
disp(tr,"Rise time (sec) of the response is:")

sigma=xi*wn
ts=4.6/sigma; //settling time
disp(ts,"Settling time (sec) of the response is:")

Mp=exp(-xi*%pi/sqrt(1-xi^2))
wd=wn*sqrt(1-xi^2);
tp=%pi/wd;
disp(tp, Mp,"Overshoot and time of overshoot (sec)...
 in the response are:")

//----------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 3.29 Stability versus parameter range

```
//Example 3.29
//Stability versus parameter range

xdel(winsid())//close all graphics Windows
clear;
```

```
clc;
//-------------------------------------------------------------------------------
//Stability versus parameter range

numT=[-1];//zeros
denT=[1 0 -6];//poles
Ns=poly(numT,'s','roots');
Ds=poly(denT,'s','roots');
Gfs=syslin('c',Ns/Ds); //forward transfer function block

num=[1];
den=[1 0];
Ns=poly(num,'s','coeff');
Ds=poly(den,'s','coeff');
Hs=syslin('c',Ns/Ds); //feedback transfer function block

//check the step responses with the forward path gain K=7.5, 13, 25
t=0:0.02:12;
i=1;

for K=[7.5, 13, 25]
    sysT= (K * Gfs) /. Hs;
    yt(i,:)=csim('step',t,sysT);
    i=i+1;
end
//Step response
plot(t',yt')
exec .\fig_settings.sci; //custom script for setting figure properties
title("Transient response for different values of K",'fontsize',3);
xlabel('Time t (sec.)','fontsize',2)
ylabel('y(t)','fontsize',2)
```

```
h=legend('K=7.5','K=13', 'K=25')
h.legend_location = "in_upper_right"
h.fill_mode='off'
//
    --------------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 3.30 Stability versus two parameter ranges

```
//Example 3.30
//Stability versus two parameter ranges
xdel(winsid())//close all graphics Windows
clear;
clc;
//
    --------------------------------------------------------------------------------

//Stability versus parameter ranges

num=[1 0];//zeros
den=[-1 -2];//poles
Ns=poly(num,'s','coeff');
Ds=poly(den,'s','roots');
Gfs=syslin('c',Ns/Ds); //forward transfer function block

num=[1];
den=[1 0];
Ns=poly(num,'s','coeff');
Ds=poly(den,'s','coeff');
```

```
Hs=syslin('c',Ns/Ds); //feedback transfer function block

//check the step responses with the forward, path gain K=7.5, 13, 25
t=0:0.02:12;
i=1;
num=[5 10;1 1;0 1];

for i=1:3
    den=[0 1];
    Ns=poly(num(i,:),'s','coeff');
    Ds=poly(den,'s','coeff');
    Gcs=syslin('c',Ns/Ds); //Controller transfer function block
    sysT= Gcs * Gfs /. Hs;
    yt(i,:)=csim('step',t,sysT);
    i=i+1;
end

//Transient response for different values of K and Ki
plot(t',yt')
exec .\fig_settings.sci; //custom script for setting figure properties
title("Transient response for the system",'fontsize',3);
xlabel('Time t (sec.)','fontsize',2)
ylabel('y(t)','fontsize',2)
xset("font",1,1)
xstring(1.4,1.05,'$K=10,K_I=5$');
xstring(3.3,0.8,'$K=1,K_I=1$');
xstring(5.5,0.35,'$K=1,K_I=0$')
//------------------------------------------------------------------------
```

Chapter 4

Basic properties of feedback

Scilab code Exa 4.6 PID Control of DC Motor Speed

```
//Example 4.6
//PID Control of DC Motor Speed.

//-------------------------------------------------------------------------------
//NOTE THAT---

//The model as given in matlab program for this example in the book is

//num=Ra*s + La*s^2 ;
//den=Ke*ki + (Ra*Ke*Ke+Ke*kp)*s + (Ra*b+Ke*Ke+Ke*kd)*s^2 + Jm*La*s^3;

//this does not match to the model of DC motor given on page 43.
//Also, if we assume this model, disturbance response given
//in figure 4.13 (a)
//is different from expected.
```

```
//For instance, with P control, output should asymptotically go to 0
//for disturbance step input, because numerator is s(Ra + La*s)
//and system is type 0 (no pole at origin).
//i.e. y(inf)=lim s->0 s*Y(s)= s*[s(Ra + La*s)/den]*1/s=0;

//In following code, we have considered correct model of DC motor as
//given on page 43. Note that, this model must have been used
//by authors of the book for
//step reference tracking as it is correctly shown in figure 4.13 (b)

//--------------------------------------------------------------------------------------------
xdel(winsid())//close all graphics Windows
clear;
clc;

//--------------------------------------------------------------------------------------------
// System parameters
Jm=0.0113; // N-m-s^2/rad
b=0.028;   // N-m-s/rad
La=0.1;    // henry
Ra=0.45;   // ohms
Kt=0.067   // n-m/amp
Ke=0.067;  // V-sec/amp

// Controller parameters
kp=3;
ki=15;    // sec^-1
kd=0.3;   // sec
```

```

// DC Motor Transfer function as given on page 43 of book (edition 5)
//G=Kt/[Jm*La s^2 + (Jm*Ra + La*b)s +(Ra*b +Kt*Ke)]
s=%s;
num=[Kt];
den=[(Ra*b +Kt*Ke) (Jm*Ra + La*b) Jm*La];
Ns=poly(num,'s','coeff');
Ds=poly(den,'s','coeff');
G=syslin('c',Ns/Ds)

//PID controller , Gc=(kd s^2 + kp s + ki)/s
num=[ki kp kd;ki kp 0;0 kp 0]; //numerator parameters of controller)
                                 //(row wise for PID , PI and P)
den=[0 1];                    //denominator parameters of controller
Ds=poly(den,'s','coeff');    //denominator polynomial of controller
t=0:0.005:10;               // Simulation time
//------------------------------------------------------------------
//Step disturbance response with P, PI and PID controller.

for i=1:3
Ns=poly(num(i,:),'s','coeff');//numerator polynomial of controller
sysG=syslin('c',Ns/Ds);
sysD=G/. sysG;
v(i,:)=csim('step',t,sysD);
end
plot(t',v');
//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script to set the figure properties
```

```
title('Responses of P,PI and PID control to step disturbance...
 input','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('Amplitude','fontsize',2)
hl=legend(['PID','PI','P']);

//--------------------------------------------------------------------------------------------
//Reference step response

figure
for i=1:3
Ns=poly(num(i,:),'s','coeff');
Gc=syslin('c',Ns/Ds);
// Step reference response with P, PI and PID controller.
sysR=G*Gc/(1+G*Gc);
v(i,:)=csim('step',t,sysR);
end
plot(t',v')
//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script to set the figure properties
title('Responses of PID control to step reference input','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('Amplitude','fontsize',2)
hl=legend(['PID','PI','P']);

//--------------------------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 4.7 Discrete Equivalent

```
//Example 4.7
//Discrete Equivalent.
//------------------------------------------------------------------------------------------------

xdel(winsid())//close all graphics Windows
clear;
clc;

// Transfer function
s=%s;
num=[1 11];
den=[1 3]
Us=poly(num,'s','coeff');
Es=poly(den,'s','coeff');
Ds=syslin('c',Us/Es);
sysc=tf2ss(Ds)

//Discretize the system using sampling time Ts=1 and Bilinear Transform
Ts=1;
sysd=cls2dls(sysc,Ts);

//Pulse transfer function
Dd=ss2tf(sysd)
```

```
disp(Dd,"Dd=")
disp("Note that, multiply numerator and denomintor each by 7...
 will give the result as in book.")
//------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 4.8 Equivalent discrete controller for DC motor speed control

```
//Example 4.8
//Equivalent discrete controller for DC motor speed control.
//------------------------------------------------------------------------

//NOTE THAT-- The system response (continuous) to sampled control
//input depends on
//the sampling time set for continuous signal in SIMULATION.
//In this example we consider sampling period of 0.009 sec
//to represent continuous time signal.
//------------------------------------------------------------------------

xdel(winsid())//close all graphics Windows
clear;
clc;
```

```
//  ------------------------------------------------------------------------
// Continuous time system and controller
// System transfer function
s=%s;
num=[45 0];
den=[45 14 1]
Nms=poly(num,'s','coeff');
Dns=poly(den,'s','coeff');
Gp=syslin('c',Nms/Dns); //system transfer function

// Controller

numDa=[6 1];
denDa=[0 1]
Nms=poly(numDa,'s','coeff');
Dns=poly(denDa,'s','coeff');
sysD=syslin('c',1.4*Nms/Dns); //controller transfer function

//Closed loop responses

num=[1 0];
den=[1 0];
Nms=poly(num,'s','coeff');
Dns=poly(den,'s','coeff');
H=syslin('c',Nms/Dns)

sysDa=Gp*sysD/.H;

//step response and control input
t=0:0.009:5;
yt=csim('step',t,sysDa); //step response
figure(0)
plot2d(t,yt,1)
Gu=sysD/(1+Gp*sysD);
ut=csim('step',t,Gu); //control input
```

```
figure(1)
plot2d(t,ut,1)
//-------------------------------------------------------------------------------

sys=tf2ss(Gp); //state space model of the system
con=tf2ss(sysD); //controller state space model

// discrete-time time system and controller

//Discretize the system and control with sampling time Ts=0.07
// using Bilinear Transform
Ts=0.07;
sysDd=cls2dls(sys,Ts); // discrete-time system state space model
conDd=cls2dls(con,Ts); // discrete-time controller state space model

//Pulse transfer function of system
Gpz=ss2tf(sysDd);
//Pulse transfer function of controller
Gcz=ss2tf(conDd);
//Closed loop response
Gz=Gpz*Gcz/(1+Gpz*Gcz)
//Control input pulse transfer function
Guz=Gcz/(1+Gpz*Gcz)
T=0:Ts:5;
r=ones(1,length(T));
yd=flts(r,Gz);.............//Discrete respnse to discrete input
ud=flts(r,Guz);            //Discrete Control input
//continuous response for digital input
t=0:0.009:5;
k=0;

for i=1:length(yd)
```

```
    for j=1:8
       if (k+j)>length(t) then
          break
          else
          YD(1,k+j)=yd(i);
       end
       end
    k=k+j;
end

yt=csim(1-YD,t,Gp*sysD);
scf(0)
plot2d(t,yt,5);
scf(1)
plot2d2(T,ud,5);
//-------------------------------------------------------------------
//Discretize the system and control with sampling time Ts=0.035
// using Bilinear Transform
Ts=0.035;
sysDd=cls2dls(sys,Ts); // discrete-time system state space model
conDd=cls2dls(con,Ts); // discrete-time controller state space model

Gpz=ss2tf(sysDd);       //Pulse transfer function of system
Gcz=ss2tf(conDd);      //Pulse transfer function of controller

//Closed loop response
Gz=Gpz*Gcz/(1+Gpz*Gcz)
//Control input pulse transfer function
Guz=Gcz/(1+Gpz*Gcz)
T=0:Ts:5;
r=ones(1,length(T));
```

```
yd=flts(r,Gz);............//Discrete respnse to discrete input
ud=flts(r,Guz);           //Discrete Control input
t=0:0.009:5;
k=0;

for i=1:length(yd)
    for j=1:4
       if (k+j)>length(t) then
          break
          else
          YD(1,k+j)=yd(i);
       end
       end
    k=k+j;
end

yt=csim(1-YD,t,Gp*sysD);
scf(0)
plot2d(t,yt,2);
scf(1)
plot2d2(T,ud,2);

scf(0)
//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script to set the figure properties
title('Comparision plots of Speed-control system with continuous...
 and discrete controllers','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
hl=legend(['Continuous time','Discrete-time, Ts=0.07 s'...
,'Discrete-time, Ts=0.035 s'],4);
scf(1)
//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script to set the figure properties
```

```
title('Comparision plots of Speed-control system with continuous...
 and discrete controllers','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
hl=legend(['Continuous time','Discrete-time, Ts=0.07 s',...
'Discrete-time, Ts=0.035 s']);
//____________________________________________________________________________
```

Chapter 5

The Root Locus Design method

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 5.1 Root locus of a Motor Position Control

```
//Example 5.1
//Root locus of a Motor Position Control.

xdel(winsid())//close all graphics Windows
clear;
clc;

//
  ------------------------------------------------------------------------

//System transfer function and its root locus

s=poly(0,'s');
Ls=1/(s*(s+1));

//Title, labels and grid to the figure
```

```
exec .\fig_settings.sci; //custom script for setting figure properties
evans(Ls)
title(['Root locus for', '$L(s)=1/[s(s+1)]$'],'fontsize',3)
zoom_rect([-2 -1.5 2 1.5])
sgrid([0.5],1,5)
xset("font",1,1.5)
xstring(-1.2,1.1,'$\theta=sin^{-1} \xi$",0,0)
h=legend('');
h.visible = "off"

//------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 5.2 Root locus with respect to a plant open loop pole

```
//Example 5.2
//Root locus with respect to a plant open loop pole.
xdel(winsid())//close all graphics Windows
clear;
clc;

//
   ------------------------------------------------------------------
//System transfer function and its root locus
s=poly(0,'s');
Gs=s/(s*s+1);

```

```
//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
evans(Gs,100)
title(['Root locus vs. damping factor','$c$',...
'for','$1+G(s)=1+1/[s(s+c)]=0$'],'fontsize',3)
zoom_rect([-2 -1.5 2 1.5])
h=legend('');
h.visible = "off"

//
---------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 5.3 Root locus for satellite attitude control with PD control

```
//Example 5.3
//Root locus for satellite attitude control with PD control.

xdel(winsid())//close all graphics Windows
clear;
clc;

//
---------------------------------------------------------------------------

//System transfer function and its root locus

```

```
s=poly(0,'s');
sysS=(s+1)/(s^2);
evans(sysS,100)

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title(['Root locus for','$L(s)=G(s)=(s+1)/s^2$'],'fontsize',3)
zoom_rect([-6 -3 2 3])
h=legend('');
h.visible = "off"

//
--------------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 5.4 Root locus for satellite attitude control with modified PD control or Lead compensator

```
//Example 5.4
//Root locus for satellite attitude control with modified
//PD control or Lead compensator.

xdel(winsid())//close all graphics Windows
clear;
clc;

```

```
//
--------------------------------------------------------------------------------

//System transfer function and its root locus

s=poly(0,'s');
sysL=(s+1)/(s^2*(s+12));
evans(sysL,100)

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title(['Root locus for', '$L(s)=(s+1)/s^2(s+12)$'],'fontsize',3)
zoom_rect([-6 -3 2 3])
h=legend('');
h.visible = "off"

//
--------------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 5.5 Root locus for satellite control with Lead compensator

```
//Example 5.5
//Root locus for satellite control with Lead compensator.

xdel(winsid())//close all graphics Windows
```

```
clear;
clc;

//------------------------------------------------------------------------------------------------------------------
//System transfer function and its root locus

s=poly(0,'s');
sysL=(s+1)/(s^2*(s+4));
evans(sysL)

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title(['Root locus for', '$L(s)=(s+1)/s^2(s+4)$'],'fontsize',3)
zoom_rect([-6 -3 2 3])
h=legend('');
h.visible = "off"

//------------------------------------------------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 5.6 Root locus for satellite attitude control with a Transition value for the pole

```
//Example 5.6
//Root locus for satellite attitude control with a
```

```
//Transition value for the pole.

xdel(winsid())//close all graphics Windows
clear;
clc;

//
-------------------------------------------------------------------------

//System transfer function and its root locus

s=poly(0,'s');
sysL=(s+1)/(s^2*(s+9));
evans(sysL)

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title(['Root locus for', '$L(s)=(s+1)/(s^2(s+9))$'],'fontsize',3)
zoom_rect([-6 -3 2 3])
h=legend('');
h.visible = "off"

//
-------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 5.7 Root locus for satellite control with a Collocated Flexibility

```
//Example 5.7
//Root locus for satellite control with a Collocated Flexibility.

xdel(winsid())//close all graphics Windows
clear;
clc;

//------------------------------------------------------------------------

//System transfer function with controller.

s=poly(0,'s');
NumD=(s+1);
DenD=(s+12);
D=NumD/DenD;

NumG=(s+0.1)^2+36
DenG=s^2*((s+0.1)^2+(6.6)^2)

G=NumG/DenG;

NumL=NumD*NumG;
DenL=DenD*DenG;

L=NumL/DenL;

zr=roots(NumL);
pl=roots(DenL);

//------------------------------------------------------------------------

//Angle of departure.
//Find angle of departure from pole at phi1=-0.1+6.6i
//(real poles don't have angle of departure,
```

```
//they move along real axis only)
//psi1=angle[(Departing pole)- (zero at - 0.1 + 6.6i)]
[Mpsi1, psi1] = polar(pl(2)-zr(1))
psi1=real(psi1)*180/%pi;           //angle in degree

//psi2=angle[(Departing pole)- (zero at - 0.1 - 6.6i)]
[Mpsi2, psi2] = polar(pl(2)-zr(2))
psi2=real(psi2)*180/%pi;           //angle in degree

//psi3=angle[(Departing pole)- (zero at - 1)]
[Mpsi3, psi3] = polar(pl(2)-zr(3))
psi3=real(psi3)*180/%pi;           //angle in degree

//phi2=angle[(Departing pole)- (pole at 0)]
[Mphi2, phi2] = polar(pl(2)-pl(4))
phi2=real(phi2)*180/%pi;           //angle in degree

//phi3 is same as phi2, as pole is repeated at 0.
phi3=phi2;

//phi4=angle[(Departing pole)-(pole at - 0.1 - 6.6i)]
[Mphi4, phi4] = polar(pl(2)-pl(3))
phi4=real(phi4)*180/%pi;           //angle in degree

//phi5=angle[(Departing pole)- (pole at - 12 )]
[Mphi5, phi5] = polar(pl(2)-pl(1))
phi5=real(phi5)*180/%pi;           //angle in degree

//Therefore angle of departure phi1 at - 0.1 + 6.6i is
//phi1 = 180 + sum(angle to zeros) - sum(angle to poles)

phi1 = 180 + sum(psi1+psi2+psi3) - sum(phi2+phi3+phi4+phi5)
```

```

//angle contributions in figure
figure(0)
plzr(L)
xset('font size',1.5)
xarrows([real(pl(1));real(pl(2))],[imag(pl(1));imag(
   pl(2))],0,2)
xarc(-13,1,2,2,0,phi5*64)
xstring(-11,0.05,"$\phi_5$")


xarrows([real(zr(3));real(pl(2))],[imag(zr(3));imag(
   pl(2))],0,4)
xarc(-2,1,2,2,0,psi3*64)
xstring(-0.7,1,"$\psi_3$")

xarrows([real(pl(4));real(pl(2))],[imag(pl(4));imag(
   pl(2))],0,5)
xarc(-1,1,2,2,0,phi2*64)
xstring(0.8,0.5,"$\phi_2,\,\phi_3$")


xarrows([real(pl(3));real(pl(2))],[imag(pl(3));imag(
   pl(2))],0,3)
xarc(-1,-6.6,2,2,0,phi4*64)
xstring(0.8,-7,"$\phi_4$")

xarrows([real(zr(2));real(pl(2))],[imag(zr(2));imag(
   pl(2))],0,6)
xarc(-1,-5,2,2,0,psi2*64)
xstring(0.8,-5.5,"$\psi_2$")

xarrows([real(zr(1));real(pl(2))],[imag(zr(1));imag(
   pl(2))],0,24)
xstring(0.3,5.5,"$\psi_1$")
xstring(0.3,6.5,"$\phi_1$")

exec .\fig_settings.sci; //custom script for setting
```

```
    figure properties
title(['Figure for computing a departure angle for'
   ,...
'$L(s)=\frac{s+1}{s+12}\frac{(s+0.1)^2+6^2}{s^2[(s
   +0.1)^2+6.6^2]}$'],...
'fontsize',3)
zoom_rect([-15 -8 5 8])
h=legend('');
h.visible = "off"

//
   ----------------------------------------------------------------------

//Root locus system transfer function with
   controller.
figure(1)
evans(L)
//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting
    figure properties
title(['Root locus for','$L(s)=\frac{s+1}{s+12}\frac
   {(s+0.1)^2+6^2}...
{s^2[(s+0.1)^2+6.6^2]}$'],'fontsize',3)
zoom_rect([-15 -8 5 8])
h=legend('');
h.visible = "off"

//
   ----------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 5.8 Root locus for noncollocated case

```
//Example 5.8
//Root locus for noncollocated case.

xdel(winsid())//close all graphics Windows
clear;
clc;

//----------------------------------------------------------------------------
//System transfer function with controller

s=poly(0,'s');
NumD=(s+1);
DenD=(s+12);
sysD=NumD/DenD;

NumG=1
DenG=s^2*((s+0.1)^2+(6.6)^2)

sysG=NumG/DenG;

NumL=NumD*NumG;
DenL=DenD*DenG;

sysL=NumL/DenL;

zr=roots(NumL);
pl=roots(DenL);

//----------------------------------------------------------------------------
```

```
//Angle of departure.
//Find angle of departure from pole at phi1= - 0.1 + 6.6i
//(real poles don't have angle of departure,
//they move along real axis only)

//psi1=angle[(Departing pole)- (zero at - 1]
[Mpsi1, psi1] = polar(pl(2)-zr(1))
psi1=real(psi1)*180/%pi;            //angle in degree

//phi2=angle[(Departing pole)- (pole at 0)]
[Mphi2, phi2] = polar(pl(2)-pl(4))
phi2=real(phi2)*180/%pi;            //angle in degree

//phi3 is same as phi2, as pole is repeated at 0.
phi3=phi2;

//phi4=angle[(Departing pole)-(pole at - 0.1 - 6.6i)]
[Mphi4, phi4] = polar(pl(2)-pl(3))
phi4=real(phi4)*180/%pi;            //angle in degree

//phi5=angle[(Departing pole)- (pole at - 12 )]
[Mphi5, phi5] = polar(pl(2)-pl(1))
phi5=real(phi5)*180/%pi;            //angle in degree

//Therefore angle of departure phi1 at - 0.1 + 6.6i is
//phi1 = 180 + sum(angle to zeros) - sum(angle to poles)

phi1 = 180 + sum(psi1) - sum(phi2+phi3+phi4+phi5)

//angle contributions in figure
figure(0)
plzr(sysL)
xset('font size',1.5)
```

```
xarrows([real(pl(1));real(pl(2))],[imag(pl(1));imag(pl(2))],0,2)
xarrows([real(pl(1)); -10],[0;0],0,2)
xarc(-13,1,2,2,0,phi5*64)
xstring(-11,0.05,”$\phi_5$”)

xarrows([real(zr(1));real(pl(2))],[imag(zr(1));imag(pl(2))],0,6)
xarrows([real(zr(1)); -0.3],[0;0],0,6)
xarc(-2,1,2,2,0,psi1*64)
xstring(-0.7,1,”$\psi_1$”)

xarrows([real(pl(4));real(pl(2))],[imag(pl(4));imag(pl(2))],0,5)
xarrows([real(pl(4)); 1],[0;0],0,5)
xarc(-1,1,2,2,0,phi2*64)
xstring(0.8,0.5,”$\phi_2,\,\phi_3$”)

xarrows([real(pl(3));real(pl(2))],[imag(pl(3));imag(pl(2))],0,17)
xarrows([real(pl(3)); 2],[imag(pl(3));imag(pl(3))],0,17)
xarc(-1.1,-5.6,2,2,0,phi4*64)
xstring(0.8,-5.5,”$\phi_4$”)

xstring(0.3,6.5,”$\phi_1$”)

exec .\fig_settings.sci; //custom script for setting figure properties
title([’Figure to compute a departure angle for’,...
’$L(s)=\frac{s+1}{s+12}\frac{1}{s^2[(s+0.1)^2+6.6^2]}$’],...
’fontsize’,3)
zoom_rect([-15 -8 5 8])
h=legend(’’);
h.visible = ”off”

//
```

```
    ----------------------------------------------------------------------------
//Root locus of system transfer function with controller
figure(1)
evans(sysL)
//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title(['Root locus for','$L(s)=\frac{s+1}{s+12}\frac{1}...
{s^2[(s+0.1)^2+6.6^2]}$'],'fontsize',3)
zoom_rect([-15 -8 5 8])
h=legend('');
h.visible = "off"

//
    ----------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 5.9 Root locus for the system having complex multiple roots

```
//Example 5.9
//Root locus for the system having complex multiple roots.

xdel(winsid())//close all graphics Windows
```

```
clear;
clc;
//------------------------------------------------------------------------------
//System transfer function

s=poly(0,'s');

NumL=1;
DenL=s*(s+2)*[(s+1)^2+4];

L=NumL/DenL;

zr=roots(NumL);
pl=roots(DenL);

//------------------------------------------------------------------------------
//Angle of departure.
//Find angle of departure from pole at phi1= - 1 + 2i
//(real poles don't have angle of departure,
// they move along real axis only)

//phi2=angle[(Departing pole)- (pole at 0)]
[Mphi1, phi1] = polar(pl(1)-pl(4))
phi1=real(phi1)*180/%pi;           //angle in degree

//phi2=angle[(Departing pole)- (pole at -2)]
[Mphi2, phi2] = polar(pl(1)-pl(3))
phi2=real(phi2)*180/%pi;           //angle in degree

//phi2=angle[(Departing pole)- (pole at - 1 - 2i)]
[Mphi4, phi4] = polar(pl(1)-pl(2))
phi4=real(phi4)*180/%pi;           //angle in degree

```

```
//Therefore angle of departure phi1 at - 1 + 2i is
//phi3 = 180 + sum(angle to zeros) - sum(angle to poles)

phi3 = 180 - sum(phi1+phi2+phi4)

//angle contributions in figure
figure(0)
plzr(L)
xset('font size',1.5)
xarrows([real(pl(4));real(pl(1))],[imag(pl(4));imag(pl(1))],0,2)
xarrows([real(pl(4)); 1],[0;0],0,2)
xarc(-0.5,0.5,1,1,0,phi1*64)
xstring(0.5,0.25,"$\phi_1$")

xarrows([real(pl(3));real(pl(1))],[imag(pl(3));imag(pl(1))],0,5)
xarrows([real(pl(3)); -1.3],[0;0],0,5)
xarc(-2.5,0.5,1,1,0,phi2*64)
xstring(-1.5,0.25,"$\phi_2$")

xarrows([real(pl(2));real(pl(1))],[imag(pl(2));imag(pl(1))],0,17)
xarrows([real(pl(2)); -0.3],[-2;-2],0,17)
xarc(-1.5,-1.5,1,1,0,phi4*64)
xstring(-0.5,-1.7,"$\phi_4$")

xstring(-0.8,2,"$\phi_1$")

exec .\fig_settings.sci; //custom script for setting figure properties
title(['Figure to computing a departure angle for',...
'$L(s)=\frac{1}{s(s+2)[(s+1)^2+4]}$'],'fontsize',3)
zoom_rect([-4 -3 4 3])
h=legend('');
h.visible = "off"
```

```
//----------------------------------------------------------------------------
//Root locus of system transfer function with controller
figure(1)
evans(L)
//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title(['Root locus for','$L(s)=\frac{1}{s(s+2)[(s+1)^2+4]}$']...
,'fontsize',3)
zoom_rect([-4 -3 4 3])
h=legend('');
h.visible = "off"
//----------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 5.10 Design using Lead compensator

```
//Example 5.10
//Design using Lead compensator.

xdel(winsid())//close all graphics Windows
clear;
clc;
```

```
//----------------------------------------------------------------------
//System transfer function and its root locus

s=poly(0,'s');

NumG=1;
DenG=s*(s+1);
NumD=(s+2);
DenD=(s+10);

G=NumG/DenG;
D=NumD/DenD;

L=G*D; //open loop transfer function

figure(0)
evans(L)
sgrid(0.5,7,6);

xstring(-2,4,"Damping=0.5",0,0)
xstring(-7,4,"w=7",0,0)
//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Root locus for lead design','fontsize',3)
zoom_rect([-14 -8 4 8])
h=legend('');
h.visible = "off"
//----------------------------------------------------------------------
// Unit step response
//closed loop system
K=70;
sysc=K*L/(1+K*L);
sysc=syslin('c',sysc);
```

```
t=linspace(0,10,1000);
y=csim('step',t,sysc);
figure(1)
plot(t,y);
title('Step response for the system with lead compensator','fontsize',3)
xlabel('Time (sec)','fontsize',2)
ylabel('Amplitude','fontsize',2)
set(gca(),"grid",[0.3 0.3])
zoom_rect([0 0 1.8 1.4])
exec .\fig_settings.sci;

scf(0)
pl=roots(DenG*DenD+K*NumG*NumD)     //closed loop poles at K=70;
plot(real(pl),imag(pl),'ro')       //closed loop pole-locations at K=70;
xstring(-5.8,6,"K=70",0,0)
//
  ----------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 5.11 A second Lead compensation Design

```
//Example 5.11
//A second  Lead compensation Design.

xdel(winsid())//close all graphics Windows
```

```
clear;
clc;
//
   ----------------------------------------------------------------------

//System transfer function and its root locus

s=poly(0,'s');

NumG=1;
DenG=s*(s+1);
NumD=(s+5.4);
DenD=(s+20);

Gs=NumG/DenG;
Ds=NumD/DenD;

Ls=Gs*Ds;   //open loop transfer function

zr=roots(NumD*NumG); //open loop system zeros
pl=roots(DenD*DenG); //open loop system poles
pd=-3.5+3.5*sqrt(3)*%i; //desired pole

//Construction for placing a specific point on the root locus.
figure(0)
plzr(Ls)
plot(real(pd),imag(pd),'ro')
xarrows([real(pl(1));real(pd)],[imag(pl(1));imag(pd)],0,2)
xarrows([real(pl(2));real(pd)],[imag(pl(2));imag(pd)],0,2)
xarrows([real(pl(3));real(pd)],[imag(pl(3));imag(pd)],0,2)
xarrows([real(zr);real(pd)],[imag(zr);imag(pd)],0,6)
xarrows([real(zr);-3],[0;0],0,6)
xarc(-6.4,1,2,2,0,72.6*64)
xset('font size',1.5);
```

```
xstring(-4.7,0.5,"$\psi$")
exec .\fig_settings.sci; //custom script for setting figure properties
title('Construction for placing a specific point on the root locus',...
'fontsize',3)
h=legend('');
h.visible = "off"
//----------------------------------------------------------------------------------
//Root locus of system transfer function with controller
figure(1)
evans(Ls)
sgrid(0.5,7,6)
//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title(['Root locus for','$ L(s)=\frac {s+5.4}{s(s+1)(s+20)}$'],...
'fontsize',3)
zoom_rect([-20 -8 5 8])
h=legend('');
h.visible = "off"
//----------------------------------------------------------------------------------
// Unit step response
//closed loop system

K=127;  // from root locus gain  is computed
sysc=K*Ls/(1+K*Ls)
sysc=syslin('c',sysc);
t=linspace(0,10,1000);
y=csim('step',t,sysc);
figure(2)
plot(t,y);
```

```
exec .\fig_settings.sci; //custom script for setting figure properties
title(['Step response for K=127', 'and',...
'$ L(s)=\frac {s+5.4}{s(s+1)(s+20)}$']...
,'fontsize',3)
xlabel('Time (sec)','fontsize',2)
ylabel('Amplitude','fontsize',2)
zoom_rect([0 0 1.8 1.4])

pl=roots(DenG*DenD+K*NumG*NumD) //closed loop poles at K=127;
scf(1)
plot(real(pl),imag(pl),'ro') //closed loop pole-locations at K=127;
//--------------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 5.12 Negative root Locus for an Airplane

```
//Example 5.12
//Negative root Locus for an Airplane.

xdel(winsid())//close all graphics Windows
clear;
clc;


```

```
//----------------------------------------------------------------------------------
//System transfer function and its root locus

s=poly(0,'s');
Ls=-(s-6)/(s*(s^2+4*s+13));
evans(Ls)

//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title(['Negative root locus for','$L(s)=\frac{s-6}{s(s^2+4s+13)}$'],...
'fontsize',3)
zoom_rect([-5 -6 10 6])
h=legend('');
h.visible = "off"

//----------------------------------------------------------------------------------
```

Chapter 6

The Frequency Response Design Method

Scilab code Exa 6.2.b Frequency response characteristics of Lead compensator

```
//Example 6.2
//Frequency response characteristics of Lead
   compensator.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
   ------------------------------------------------------------------------------------------------

//System transfer function and its bode plot
K=1, T=1, alpha=0.1
s=poly(0,'s');
sysD=syslin('c',K*(T*s+1)/(alpha*T*s+1));

//The bode plot of the system

fmin=0.1/2/%pi; //mininmum frq. in Hz for response
```

```
(0.1 rad/sec)
fmax=100/2/%pi; //maximum frq. in Hz for response (100 read/sec)
//------------------------------------------------------------------------

//Bode plot for frequency in Hz (scilab ver. 5.4.1)
//bode(g,fmin,fmax);
//OR
//Bode plot for frequency in rad/sec (scilab ver. 5.5.1)
bode(sysD,fmin,fmax,"rad")

//------------------------------------------------------------------------

title('(a) Magnitude and  (b) phase for the lead compensator',...
'fontsize',3)
exec .\fig_settings.sci; //custom script for setting figure properties
//------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.3 Bode Plot for Real Poles and Zeros

```
//Example 6.3
//Bode Plot for Real Poles and Zeros.

xdel(winsid())//close all graphics Windows
clear;
clc;
//-----------------------------------------------------------------------
//System transfer function and its bode plot
K=2000;
s=poly(0,’s’);
Gs=syslin(’c’,(K*(s+0.5))/(s*(s+10)*(s+50)));

//The bode plot of the system
wmin=0.1;         // mininmum frq. in rad/sec for response
wmax=100;         // maximum frq. in red/sec for response
fmin=wmin/2/%pi   // mininmum frq. in Hz for response
fmax=wmax/2/%pi   // maximum frq. in Hz for response
//-----------------------------------------------------------------------
//Bode plot for frequency in Hz (scilab ver. 5.4.1)
//bode(g,fmin,fmax)
//OR
//(Only for scilab ver. 5.5.1)
//Bode (frequency scale in rad/sec)
// or gainplot or phaseplot plot with asymptotes
figure(0)
gainplot(Gs,fmin,fmax);
bode_asymp(Gs,wmin,wmax);
xstring(0.03,22,”slope=−1(−20db/dec)”,0,0);
xstring(0.2,9,”slope=0”,0,0);
xstring(3,7,”slope=−1(−20db/dec)”,0,0)
xstring(0.9,-8,”slope=−2(−40db/dec)”,0,0)
```

```
title('Composit plots (a) magnitude plot','fontsize',3);
h=legend('');
exec .\fig_settings.sci; //custom script for setting figure properties
h.visible = "off"
//----------------------------------------------------------------------------------

//phase plot for poles and zeros
zr=((s/0.5)+1)/s //infact this is zero and pole at origin.
zr=syslin('c', zr);
pl1=1/((s/10)+1)
pl1=syslin('c', pl1);
pl2=1/((s/50)+1)
pl2=syslin('c', pl2);
figure(1)
phaseplot([Gs;zr;pl1;pl2],fmin,fmax);
xstring(5.5,-14,"$\frac {1}{s/0.5+1}$",0,0);
xstring(2.8,-22,"$\frac{1}{s/50+1}$",0,0);
xstring(2.5,-60,"$\frac{1}{s/10+1}$",0,0);
xstring(1.2,-100,["Composite";"(Actual)"],0,0);
title('Composit plots (b) Phase','fontsize',3);
exec .\fig_settings.sci; //custom script for setting figure properties

//----------------------------------------------------------------------------------
figure(2)
bode(Gs,fmin,fmax,"rad"); //frequency scale n radians
bode_asymp(Gs,wmin,wmax);
exec .\fig_settings.sci; //custom script for setting figure properties
title('(c) magnitude plot and phase plot approximate
```

```
    and actual...
','fontsize',3)
xstring(2.8,-22,"$\frac{1}{s/50+1}$",0,0);
h=legend('');
h.visible = "off"

//
  ------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.4 Bode Plot with Complex Poles

```
//Example 6.4
//Bode Plot with Complex Poles.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
  ------------------------------------------------------------------------

//System transfer function and its bode plot
K=10;
s=poly(0,'s');
Gs=syslin('c',(K)/(s*(s^2+0.4*s+4)));
//The bode plot of the system

```

```
fmin=0.1/2/%pi; //mininmum frq. in Hz for response (0.1 rad/sec)
fmax=10/2/%pi; //maximum frq. in Hz for response (100 read/sec)
//______________________________________________________________________
//Bode plot for frequency in Hz (scilab ver. 5.4.1)
//bode(g,fmin,fmax);
//OR
//Bode plot for frequency in rad/sec (scilab ver. 5.5.1)
bode(Gs,fmin,fmax,0.01,”rad”)

//______________________________________________________________________
title(['Bode plot for a transfer function with complex poles';...
 '(a) magnitude...
 (b) phase'],'fontsize',3)

//______________________________________________________________________
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.6 Bode Plot for Complex Poles and Zeros

```
//Example 6.6
//Bode Plot for Complex Poles and Zeros:
```

```
//Satellite with Flexible appendages.

xdel(winsid())//close all graphics Windows
clear;
clc;
//----------------------------------------------------------------------
//System transfer function and its bode plot
K=0.01;
s=poly(0,'s');
NumG=K*(s^2+0.01*s+1);
DenG=s^2*((s^2/4)+0.02*(s/2)+1)
sysG=syslin('c',NumG/DenG);

fmin=0.09/2/%pi; //mininmum frq. in Hz for response (0.1 rad/sec)
fmax=11/2/%pi; //maximum frq. in Hz for response (100 read/sec)
//----------------------------------------------------------------------
//Bode plot for frequency in Hz (scilab ver. 5.4.1)
//bode(g,fmin,fmax);
//OR
//Bode plot for frequency in rad/sec (scilab ver. 5.5.1)
bode(sysG,fmin,fmax,0.01,"rad")

//----------------------------------------------------------------------
title(["Bode plot for a transfer function with complex...
poles and zeros"; "(a) magnitude (b) phase"],'fontsize',3)
//----------------------------------------------------------------------
```

```

disp('NOTE : Result of the above example can be verified by checking the figures shown in example 6.5')
```

Scilab code Exa 6.7 Computation of Kv

```
//Example 6.7
//Computation of velocity error constant Kv from Bode plot

xdel(winsid())//close all graphics Windows
clear;
clc;
//----------------------------------------------------------------------------------------------------
//System transfer function and its bode plot
K=10;
s=poly(0,'s');
Gs=syslin('c',(K)/(s*(s+1)));
//The bode plot of the system

fmin=0.01/2/%pi; //mininmum frq. in Hz for response (0.1 rad/sec)
fmax=10/2/%pi; //maximum frq. in Hz for response (100 read/sec)
//----------------------------------------------------------------------------------------------------
```

```
//Bode plot for frequency in Hz (scilab ver. 5.4.1)
//bode(g,fmin,fmax);
//OR
//Bode plot for frequency in rad/sec (scilab ver. 5.5.1)
bode(Gs,fmin,fmax,0.01,"rad")
title(['Determination of Kv from the Bode plot for the system',...
'$10/[s(s+1)]$'],'fontsize',3)
//choose frequency (rad) and magnitude from bode plot and calculate Kv
//Here at w=0.01, magngitude in db is M=60
//i.e actual magnitude of the reponse is |A|=10^(M/20)
w=0.01;       // in rad
M=60          // in db
A=10^(M/20)   //actual gain

//Velocity error constant Kv=w*|A(w)|
Kv=w*A;
disp(Kv,"The Velocity error Constant from bode plot is: ")
//------------------------------------------------------------------------

// Computation of the Kv
[frq repf]=repfreq(Gs,fmin,fmax);
//frq in Hz, repf is freq. response in rectangular form.
//From bode plot, Kv=w*|A(w)|
//i.e Kv=2*pi*f*|A(2*pi*f)|

idx=1;//selecting the frequency and response at that frequency from arrays
Kv=2*%pi*frq(idx)*abs(repf(idx))
disp(Kv,"The Velocity error Constant is computed at 0.0015915 Hz (0.01 rad/sec) : ")
//
```

```
//------------------------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.8 Nyquist plot for a second order system

```
//Example 6.8
// Nyquist plot for a second order system.

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------------------------

//System transfer function and its root locus
s=poly(0,'s');
g=1/(s+1)^2;
sysG=syslin('c',g);

evans(sysG);
exec .\fig_settings.sci; //custom script for setting figure properties
f=gca();
f.x_location = "origin"
f.y_location = "origin"
title(['Root locus of','$G(s)=1/(s+1)^2$','with respect to K'],...
'fontsize',3)
zoom_rect([-3,-2,2,3])
h=legend('');
h.visible = "off"
```

```
//--------------------------------------------------------------------

figure(1)
//The bode plot of the system
fmin=0.01/2/%pi; //mininmum frq. in Hz for response (0.1 rad/sec)
fmax=100/2/%pi; //maximum frq. in Hz for response (100 read/sec)

//Bode plot for frequency in Hz (scilab ver. 5.4.1)
//bode(g,fmin,fmax);
//OR
//Bode plot for frequency in rad/sec (scilab ver. 5.5.1)
bode(sysG,fmin,fmax,0.01,"rad")
title(['Open loop bode plot for', '$G(s)=1/(s+1)^2$'],'fontsize',3);
exec .\fig_settings.sci; //custom script for setting figure properties
//--------------------------------------------------------------------


figure(2)
//The nyquist plot of the system
nyquist(sysG);
title('Nyquist plot of the evaluation of K G(s) for s=C1 and K=1'...
,'fontsize',3);
exec .\fig_settings.sci; //custom script for setting figure properties
f=gca();
f.x_location = "origin"
f.y_location = "origin"
xset('color',2)
//--------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.9 Nyquist plot for a third order system

```
//Example 6.9
// Nyquist plot for a third order system.

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------------------------

//System transfer function
s=poly(0,'s');
g=syslin('c',1/(s*(s+1)^2));

//The bode plot of the system
fmin=0.01/2/%pi;
fmax=100/2/%pi;
//[frq,repf]=repfreq(g1,fmin,fmax,0.01);
bode(g,fmin,fmax,"rad");
frq=[1,10]/2/%pi;
[frq, repf]=repfreq(g,frq);
[db, phi]=dbphi(repf);
plot(frq*2*%pi,db,'ro');
exec .\fig_settings.sci; //custom script for setting
    figure properties
```

```
title(["Bode plot for","$G(s)=1/[s(s+1)^2]$"],'fontsize',3)
//zoom_rect([[0.1 0] -70 [12 -180] 20])
xset("font size", 3);

xstring(1,0,"$C\,\, (\omega=1)$",0,0);
xstring(2,-75,"$E\,\, (\omega=10)$",0,0);
f=gca();

//------------------------------------------------------------------------------------------------------

//The nyquist plot of the system
figure;
nyquist(g,0.8/2/%pi,10/2/%pi,0.02)

exec .\fig_settings.sci; //custom script for setting figure properties
title(["Nyquist plot for","$G(s)=1/[s(s+1)]^2$"],'fontsize',3)
f=gca();
f.x_location = "origin";
f.y_location = "origin";
zoom_rect([-1 -0.2 0.5 0.2]);
xset("clipping", -1.2, 0.2, 1.4,0.4);
xset("font size", 3);
xset("color",2);
xstring(-0.6,0.1,"${\fgcolor{blue}{\omega<0}}$",0,0);
xstring(-0.6,-0.1,"${\fgcolor{blue}{\omega>0}}$",0,0);
xstring(-0.7,0.005,"${\fgcolor{blue}{\omega=\pm 1}}$",0,0);
xstring(-1,-0.2,...
"${\fgcolor{blue}{\text{From \infty at \omega=0^+}}$",0,0);
 xstring(-0.7,0.15,"${\fgcolor{blue}...
 {\text{Towards \infty at \omega=0^-}}$",0,0);
```

```
xstring(-0.525,-0.04,"C",0,0);
xstring(-0.075,0,"E",0,0);
//------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.10 Nyquist plot for an Open loop unstable system

```
//Example 6.10
// Nyquist plot for an Open-loop unstable system.

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------
//System transfer function
s=poly(0,'s');
sysG=(s+1)/(s*(s/10-1));
evans(sysG,50)
exec .\fig_settings.sci; //custom script for setting figure properties
title(["Root Locus for","$G(s)=(s+1)/[s(s/10-1)]$"],'fontsize',3)
zoom_rect([-5 -4 5 4])
f=gca();
f.x_location = "origin"
```

```
f.y_location = "origin"
h=legend('');
h.visible = "off"

g1=syslin('c',(s+1)/(s*(s/10-1)));
//--------------------------------------------------------------------------------
//The bode plot of the system
figure;
bode(g1,0.1/2/%pi,100/2/%pi,"rad")
exec .\fig_settings.sci; //custom script for setting figure properties
title(["Bode plot for","$G(s)=(s+1)/[s(s/10-1)]$"],'fontsize',3)
//bode(g,2*%pi*0.1,2*%pi*100)
//--------------------------------------------------------------------------------
figure;
//The nyquist plot of the system
nyquist(g1,0.5/2/%pi,100/2/%pi,0.05)
exec .\fig_settings.sci; //custom script for setting figure properties
title(["Nyquist plot for","$G(s)=(s+1)/[s(s/10-1)]$"],'fontsize',3)
f=gca();
f.x_location = "origin";
f.y_location = "origin";
zoom_rect([-2 -2 1 2]);
xset("color",2);
xset("font size", 3);
xstring(-1,1.5,"${\fgcolor{blue}{\omega>0}}$",0,0);
xstring(-1,-1.5,"${\fgcolor{blue}{\omega<0}}$",0,0);
xstring(-1.5,0,"${\fgcolor{blue}{\omega=\pm \sqrt{10}}}$",0,0);
xstring(-0.5,0.1,"${\fgcolor{blue}{\omega=\infty}}$",0,0);
```

```
xarrows([-0.2;0],[0.2;0],-1,2)
//------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.11 Stability properties for a conditionally stable system

```
//Example 6.11
// Stability properties for a conditionally stable system.

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------
//System transfer function
s=poly(0,'s');
Gs=(s+10)^2/(s^3);
evans(Gs,100)
exec .\fig_settings.sci; //custom script for setting figure properties
zoom_rect([-40 -20 5 20])
title(["Root locus for","$G(s)=(s+10)^2/s^3$"],'fontsize',3)
h=legend('');
h.visible = "off"
```

```
Gs1=syslin('c',(s+10)^2/(s^3));
//------------------------------------------------------------------------------------------------
//The nyquist plot of the system
figure;
nyquist(7*Gs1,8/2/%pi,100/2/%pi,0.005)
exec .\fig_settings.sci; //custom script for setting figure properties
title(["Nyquist plot for","$G(s)=(s+10)^2/s^3$"],'fontsize',3)
f=gca();
f.x_location = "origin";
f.y_location = "origin";
xset("color",2);
//------------------------------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.12 Nyquist plot for a system with Multiple Crossover frequencies

```
//Example 6.12
// Nyquist plot for a system with Multiple Crossover frequencies

xdel(winsid())//close all graphics Windows
clear;
```

```
clc;
//----------------------------------------------------------------------------
//System transfer function
s=poly(0,'s');
K=85;
g1=K*(s+1)/(s^2*(s^2+2*s+82));
g2=(s^2+2*s+43.25)/(s^2+2*s+101);

Gs=syslin('c',g2*g1);
//----------------------------------------------------------------------------
figure;
//The nyquist plot of the system
nyquist(Gs,0.5/2/%pi,100/2/%pi,0.005)
title(["Nyquist plot for the complex system";...
"$G(s)=85(s+1)(s^2+2s+43.25)/[((s^2+2s+82)(s^2+2s+101)]$"],...
'fontsize',3)
exec .\fig_settings.sci; //custom script for setting figure properties
zoom_rect([-2 -1 0.6 1])
f=gca();
f.x_location = "origin";
f.y_location = "origin";
xset("color",2);
//----------------------------------------------------------------------------
//The bode plot of the system
gm=g_margin(Gs);
pm=p_margin(Gs)
disp(pm,"Phase margin",gm,"Gain margin")
figure(1)
bode(Gs,0.01/2/%pi,100/2/%pi,0.01)
title(["Bode plot for";...
```

```
"$G(s)=85(s+1)(s^2+2s+43.25)/[((s^2+2s+82)(s^2+2s
   +101)]$"],...
'fontsize',3)
exec .\fig_settings.sci; //custom script for setting
    figure properties
//
   ---------------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.13 Use of simple design criterion for spacecraft attitude control

```
//Example 6.13
// Use of simple design criterion for spacecraft
   attitude control.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
   ---------------------------------------------------------------------------------

//System transfer function
s=poly(0,'s');
G=1/s^2;
g1=syslin('c',G);

//The bode plot of the system
```

```
zoom_rect([0.01 -20 100 60])
bode(g1,0.05/2/%pi,2/2/%pi,"rad")
exec .\fig_settings.sci; //custom script for setting figure properties
title('Magnitude of the spacecrafts frequency','fontsize',3)
//----------------------------------------------------------------------

K=1;
Td=20;
Ds=(Td*s+1);
gd1=syslin('c',K*Ds*G);

////The bode plot of compnenstaed open loop system
figure
bode(gd1,0.01/2/%pi,1/2/%pi,"rad")
exec .\fig_settings.sci; //custom script for setting figure properties
title('Bode plot for compensated open-loop transfer function'...
,'fontsize',3)
xstring(0.02,70,"-40db/decade",0,0);
xstring(0.2,40,"-20db/decade",0,0);

//The bode plot of compnenstaed closed loop system
K=0.01;
gc1=K*gd1/(1+K*gd1);
gcl1=syslin('c',gc1);
figure
bode(gcl1,0.01/2/%pi,10/2/%pi,"rad")
title('Closesd loop frequency response','fontsize',3)
exec .\fig_settings.sci; //custom script for setting figure properties

//Bandwidth
```

```
[frq, repf,splitf]=repfreq(gc1,[0.01/2/%pi:0.001:10/2/%pi]);
[db, phi]=dbphi(repf);
w=find(db<=db(1)-3);
wc=w(1);
frqc=frq(wc)*2*%pi;

plot2d3(frqc,db(wc),5)

[r c]=size(frq(1:w(1)));
magn=db(wc)*ones(r,c)
plot(frq(1:w(1))*2*%pi,magn,"b--")
temp_db=db(w);
[r c]=size(db(w));
temp_w=frqc*ones(r,c);
plot(temp_w,temp_db,"b--")
xset("font size", 3);
xstring(0.04,-16,"$\omega_{BW}$");
xstring(frqc,-4,"-3db");
xset("line style",4)
xarrows([0.01;frqc],[-10;-10],-0.2,5)
xarrows([frqc;0.01],[-10;-10],-0.2,5)
//--------------------------------------------------------------------------------------------

//Step response of PD compnensation
figure
t=0:0.5:100;
v=csim('step',t,gcl1);
plot2d(t,v)

//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title('Step response for PD compensation','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('$\theta$','fontsize',2)
```

```
//
    ----------------------------------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.14 Lead compensation for DC motor

```
//Example 6.14
//Lead compensation for DC motor.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
    ----------------------------------------------------------------------------------------------------

//System transfer function
s=poly(0,'s');
g=1/s/(s+1);
K=10; //DC gain
KGs=syslin('c',K*g);

//Lead compensator
numD=s/2+1;
denD=s/10+1;
D=numD/denD;
Ds=syslin('c',D);

KGDs=Ds*KGs; //compensated system
```

```
//------------------------------------------------------------------------
//(a) The bode plot of the system
bode([KGs;KGDs],0.1/2/%pi,100/2/%pi,[’KG(s)’;’D(s)G(s)’],”rad”);
exec .\fig_settings.sci; //custom script for setting figure properties
title(’Frequency response of lead compensation design’,’fontsize’,3)

//root locus
figure(1)
evans(KGDs/K)
xset(”font size”, 3);
xstring(-10,4,”$KD(s)=\frac{s/2+1}{s/10+1}$”,0,0)
xstring(-10,2,”$G(s)=\frac{1}{s(s+1)}$”,0,0)

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title(’Root locus for lead compensation design’,’fontsize’,3)
zoom_rect([-14 -8 4 8])
f=gca();
f.x_location = ”origin”;
f.y_location = ”origin”;
h=legend(’’);
h.visible = ”off”
//------------------------------------------------------------------------
//(b) digital version of lead compensator
//Discretize the system using sampling time Ts=0.05 and Bilinear Transform
Ts=0.05;        //in book its 0.005, which may not give expected responses
D=tf2ss(KGDs/K/g);
```

```
sysD=cls2dls(D,Ts);

//Pulse transfer function
Ddz=ss2tf(sysD)
disp(Ddz,"Ddz=")

//------------------------------------------------------------------------
//(c) Compare step and ramp responses.
//step response switch sw=1 and for ramp response sw=0
//------------------------------------------------------------------------

//step response
sw=1;
importXcosDiagram(".\Ex6_14_model.xcos")

xcos_simulate(scs_m,4);
scs_m.props.context
figure,
a1=newaxes();
a1.axes_bounds=[0,0,1.0,0.5];
plot(time_resp.time,time_resp.values)

xlabel('time');
ylabel('y');
title(["Lead-compensation design (a) step Response ...
 (b) ramp response"],'fontsize',3)
exec .\fig_settings.sci; //custom script for setting figure properties
legend("continuous controller","digital controller",4)
//------------------------------------------------------------------------
```

```
//ramp response
sw=0;
importXcosDiagram(".\Ex6_14_model.xcos")

xcos_simulate(scs_m,4);
scs_m.props.context

a2=newaxes();
a2.axes_bounds=[0,0.5,1.0,0.5];
plot(time_resp.time,time_resp.values)

xlabel('time');
ylabel('y');
title("(b)",'fontsize',3)
exec .\fig_settings.sci; //custom script for setting figure properties
legend("continuous controller","digital controller",4)
//_______________________________________________________________________________
```

This code can be downloaded from the website wwww.scilab.in check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.15 Lead compensation for Temperature Control System

```
//Example 6.15
//Lead compensation for Temperature Control System.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
----------------------------------------------------------------------

//System transfer function
s=poly(0,'s');
numG=1;
denG=(s/0.5+1)*(s+1)*(s/2+1);
sysG=numG/denG;
//Dc gain
K=9;

KGs=syslin('c',K*sysG);

//Lead compensator 1
numD=s+1;
denD=s/3+1;
D1=numD/denD;
D1s=syslin('c',D1);

KGD1s=D1s*KGs; //compensated system

//Lead compensator 2
numD=s/1.5+1;
denD=s/15+1;
D2=numD/denD;
D2s=syslin('c',D2);

KGD2s=D2s*KGs; //compensated system

//The bode plot of the system with K
bode([KGs;KGD1s;KGD2s],0.1/2/%pi,10/2/%pi,['KG';'KGD1';'KGD2'],"rad");
```

```
exec .\fig_settings.sci; // custom script for setting figure properties
title('Bode plot for lead compensation design','fontsize',3)
//--------------------------------------------------------------------------------
//Margins of uncompensated and compensated systems
[gm1,wcg1]=g_margin(KGs);
[pm1,wcp1]=p_margin(KGs);
disp(wcp1*2*%pi,"Wcp",wcg1*2*%pi,"Wcg",pm1,...
"Phase margin",gm1,"Gain margin",...
"Uncompensated system :")

[gm2,wcg2]=g_margin(KGD1s);
[pm2,wcp2]=p_margin(KGD1s);
disp(wcp2*2*%pi,"Wcp",wcg2*2*%pi,"Wcg",pm2,...
"Phase margin",gm2,"Gain margin",...
"System with D1 compensator :")

[gm3,wcg3]=g_margin(KGD2s);
[pm3,wcp3]=p_margin(KGD2s);
disp(wcp3*2*%pi,"Wcp",wcg3*2*%pi,"Wcg",pm3,...
"Phase margin",gm3,"Gain margin",...
"System with D2 compensator :")
//--------------------------------------------------------------------------------
//step response comparison
//closed loop system
Gc1=KGD1s/(KGD1s+1);
Gc2=KGD2s/(KGD2s+1);
figure;
t=0:0.05:20;
v1=csim('step',t,Gc1);
v2=csim('step',t,Gc2);
plot2d([t',t'],[v1',v2'])

```

```
//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title('Step response for lead compensation design','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('y','fontsize',2)

xset("font size", 3);
xarrows([2.5;1.5],[1.3;1.2],-1,1)
xstring(2.5,1.3,"D2",0,0)
xstring(4,1.2,"D1",0,0)
//------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.16 Lead compensation for Servomechanism System

```
//Example 6.16
//Lead compensation for Servomechanism System.

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------

//System transfer function
```

```
s=poly(0,'s');
numG=10;
denG=s*(s/2.5+1)*(s/6+1);
G=numG/denG;
//Dc gain
K=1;

KGs=syslin('c',K*G);

//Lead compensator 1
numD=s/2+1;
denD=s/20+1;
D1=numD/denD;
D1s=syslin('c',D1);

KGD1s=D1s*KGs; //compensated system

//Lead compensator 2
numD=s/4+1;
denD=s/40+1;
D2=D1*numD/denD; //double compensator
D2s=syslin('c',D2);


KGD2s=D2s*KGs; //compensated system

//The bode plot of the system with K
bode([KGs;KGD1s;KGD2s],0.1/2/%pi,100/2/%pi,['KG';'KGD1';'KGD2'],"rad");
exec .\fig_settings.sci; //custom script for setting figure properties
title('Bode plot for lead compensation design','fontsize',3)
//------------------------------------------------------------------------
//Margins of uncompensated and compensated systems
[gm1,wcg1]=g_margin(KGs);
```

```
[pm1,wcp1]=p_margin(KGs);
disp(wcp1*2*%pi,"Wcp",wcg1*2*%pi,"Wcg",pm1,...
"Phase margin",gm1,"Gain margin","Uncompensated
    system :")

[gm2,wcg2]=g_margin(KGD1s);
[pm2,wcp2]=p_margin(KGD1s);
disp(wcp2*2*%pi,"Wcp",wcg2*2*%pi,"Wcg",pm2,...
"Phase margin",gm2,"Gain margin","System with D1
    compensator :")

[gm3,wcg3]=g_margin(KGD2s);
[pm3,wcp3]=p_margin(KGD2s);
disp(wcp3*2*%pi,"Wcp",wcg3*2*%pi,"Wcg",pm3,...
"Phase margin",gm3,"Gain margin","System with D2
    compensator :")
//
    ------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.17 Lag compensation for Temperature Control System

```
//Example 6.17
//Lag compensation for Temperature Control System.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
    ------------------------------------------------------------------
```

```
//System transfer function
s=poly(0,'s');
numG=1;
denG=(s/0.5+1)*(s+1)*(s/2+1);
G=numG/denG;
//Dc gain
K=3; //to set phase requirement

KGs=syslin('c',K*G);

//Lag compensator
numD=5*s+1;
denD=15*s+1;
D=3*numD/denD;
Ds=syslin('c',D);

KGDs=Ds*KGs; //compensated system

//The bode plot of the system with K
bode([KGs;KGDs],0.01/2/%pi,10/2/%pi,['KG';'KGD'],"rad");
exec .\fig_settings.sci; //custom script for setting figure properties
title('Frequency response of lag-compensation design','fontsize',3)

//------------------------------------------------------------------------------------------
//Margins of uncompensated and compensated systems
[gm1,wcg1]=g_margin(KGs);
[pm1,wcp1]=p_margin(KGs);
disp(wcp1*2*%pi,"Wcp",wcg1*2*%pi,"Wcg",pm1,"Phase margin",...
gm1,"Gain margin","Uncompensated system :")

[gm2,wcg2]=g_margin(KGDs);
[pm2,wcp2]=p_margin(KGDs);
```

```
disp(wcp2*2*%pi,"Wcp",wcg2*2*%pi,"Wcg",pm2,"Phase margin",...
gm2,"Gain margin","Compensated system :")

//-------------------------------------------------------------------------------------

//step response
//closed loop system
Gc=KGDs/(KGDs+1);
figure;
t=0:0.05:20;
v=csim('step',t,Gc);
plot2d(t,v)

//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title('Step response for lag compensation design','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('y','fontsize',2)
//-------------------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.18 Lag compensation for DC motor

```
//Example 6.18
//Lag compensation for DC motor.

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------------------------

//System transfer function
s=poly(0,'s');
g=1/s/(s+1);
K=10; //DC gain
KGs=syslin('c',K*g);

//Lag compensator
numD=10*s+1; //0.1
denD=100*s+1; //0.01
D=numD/denD;
Ds=syslin('c',D);

KGDs=Ds*KGs; //compensated system

//The bode plot of the system
bode([KGs;KGDs],0.001/2/%pi,10/2/%pi,['KG(s)';'D(s)G(s)'],"rad");
exec .\fig_settings.sci; // custom script for setting figure properties
title('Frequency response of lag-compensation design ...
 of DC motor','fontsize',3)
//------------------------------------------------------------------------------------------

//step response
//closed loop system
Gc=KGDs/(KGDs+1);
figure;
```

```
t=0:0.05:50;
v=csim('step',t,Gc);
plot(t,v,2)

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for
   setting figure properties
title('Step response for Lag-compensation design...
 of DC motor','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('y','fontsize',2)
//
   -------------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 6.19 PID compensation design for spacecraft attitude control

```
//Example 6.19
//PID compensation design for spacecraft attitude
   control.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
   -------------------------------------------------------------------------------
```

```
//System transfer function
s=poly(0,'s');
G1=(0.9/s^2);
G2=(2/(s+2));
G=G1*G2;
Gs=syslin('c',G);

// PID controller parameters
Td_inv=0.1;  //  Td_inv=1/Td=0.1
Kd=1/Td_inv; //Kd=Td=Td_inv (derivative gain)

Ti_inv=0.005;  //  Ti_inv=1/Ti=0.005
Ki=Ti_inv; //Ki=Ti_inv (integral gain)

Kp=0.05 //Kp (Proportional gain)

D=Kp*(Kd*s+1)*(Ki/s+1); //PID Compensator

Dsc=syslin('c',D);

Ds=syslin('c',D/Kp); //PID Compensator with Kp=1
// Compensated system with Kp=1
GDs=Gs*Ds;
//PID compensated system Kp=0.05;
GDsc=Gs*Dsc;
//------------------------------------------------------------------------------------------

//The bode plots
bode([Gs;GDs;GDsc],0.01/2/%pi,100/2/%pi,...
['G(s)';'D(s)G(s) with (Kp=1)';'D(s)G(s) with (Kp=0.05)'],"rad");
exec .\fig_settings.sci; //custom script for setting figure properties
title('Compensation for PID design','fontsize',3)

//Phase margin of pid compensated system with Kp=0.05;
```

```
[pm wcp]=p_margin(GDsc);

//--------------------------------------------------------------------------------

//closed loop system
//step response
Gc=GDsc/(GDsc+1);
figure;
t=0:0.05:40;
y=csim('step',t,Gc);
plot(t,y,2)

//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title('Step response for PID compensation of spacecraft'...
,'fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('$theta$','fontsize',2)
//--------------------------------------------------------------------------------

//step disturbance response
Gc=G1/((G1*G2*D)+1);
Gcs=syslin('c',Gc);
figure;
t=0:0.5:1000;
u=0.1*ones(1,length(t));
y=csim(u,t,Gcs)
plot(t,y,2)

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Step disturbance response for PID compensation...
```

```
 of spacecraft','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('$theta$','fontsize',2)
//
  ------------------------------------------------------------------
```

Chapter 7

State Space Design

Scilab code Exa 7.2.b Cruise control system step response

```
//Example 7.2
//Cruise control system step response.

xdel(winsid())//close all graphics Windows
clear;
clc;
clc;
//
   ____________________________________________________________________________

//Cruise control system parameters
m=1000;
b=50;
u=500;

// Transfer function
s=%s; // or
s=poly(0,'s');
sys1=syslin('c',(1/m)/(s+b/m));
disp(sys1)
//
```

```
----------------------------------------------------------------------------
F=[0 1; 0 -b/m];
G=[0;1/m];
H=[0 1];
J=0;
sys=syslin('c',F,G,H,J);
//----------------------------------------------------------------------------
//step response to u=500;
t=0:0.5:100;
v=csim('step',t,u*sys);
plot(t,v,2)

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Responses of car velocity to a step in u','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('Amplitude','fontsize',2)
//----------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.7 Analog computer Implementation

```
//Example 7.7
//Analog computer Implementation.
```

```

xdel(winsid())//close all graphics Windows
clear;
clc;
//--------------------------------------------------------------------------------
// State space model of the given system
F=[-6 -11 -6; 1 0 0; 0 1 0];
G=[6; 0; 0];
H=[0 0 1];
J=0;
sys_ss=syslin('c',F,G,H,J)
disp(sys_ss)
//--------------------------------------------------------------------------------
//Transfer function form
[d,Ns,Ds]=ss2tf(sys_ss)
Ns=clean(Ns);
G=syslin('c',Ns/Ds);
disp(G)
//--------------------------------------------------------------------------------
// convert numerator - denominator to pole - zero form
//gain (K) pole (P) and zeros (Z) of the system
temp=polfact(Ns);
Z=roots(Ns); //locations of zeros
P=roots(Ds); //locations of poles
K=temp(1); //first entry is always gain
disp( K,"Gain", P, "Poles",Z,"Zeros",)
//--------------------------------------------------------------------------------
```

Scilab code Exa 7.8 Time scaling an oscillator

```
//Example 7.8
//Time scaling an oscillator.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
   ----------------------------------------------------------------------------

// State space model of an oscillator
wn=15000 // rad/sec
F=[0 1;wn^2 0];
G=[0;10^6];
disp(G,"G",F,"F","Given system");

//
   ----------------------------------------------------------------------------

// State space model of the time-scaled system for
// a millisecond scale w0=1e3;
w0=1e3; //rad/sec
F1=F/w0;
G1=G/w0;
disp(G1,"G1",F1,"F1","Time scaled system in mm");
//
   ----------------------------------------------------------------------------
```

Scilab code Exa 7.9 State Equations in Modal Canonical Form

```
//Example 7.9
//State Equations in Modal Canonical Form.

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------------

//System transfer function
s=poly(0,'s');
g1=1/s^2;
g2=-1/(s^2+2*s+4);
Gs=g1+g2;
//------------------------------------------------------------------------------

// State space representation in modal canonical form
sys1=tf2ss(g1);
sys2=tf2ss(g2);
[F1,G1,T1]=canon(sys1.A, sys1.B)
H1=sys1.C*T1;

[F2,G2,T2]=canon(sys2.A, sys2.B)
H2=sys2.C*T2;

F=[F1 zeros(2,2);zeros(2,2) F2];
G=[G1;G2];
H=[H1,H2];
J=0;
disp(J,"J",H,"H",G,"G",F,"F","System in modal canonical form")
//------------------------------------------------------------------------------

 //As Y=G*U;  consatnts k1 and k2  are taken out from G1 and G2 will be
```

```
 //multiplied to H1 and H2

// So alternately , it can be reprsented as
k1=-1;k2=-2;
F=[F1 zeros(2,2);zeros(2,2) F2];
G=[G1/k1;G2/k2];
H=[H1*k1,H2*k2];
J=0;
disp(J,”J”,H,”H”,G,”G”,F,”F”,”System in modal canonical form”)
//------------------------------------------------------------------------------------------------
```

Scilab code Exa 7.10 Transformation of Thermal System from Control to Modal Form

```
//Example 7.10
//Transformation of Thermal System from Control to Modal Form

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------------------------------
// State space matrices of the given system
Ac=[-7 1; -12 0];
Bc=[1;2];
Cc=[1 0];
Dc=0;
//------------------------------------------------------------------------------------------------
```

```
// State space representation in modal canonical form
T=[4 -3;-1 1]
Am=T\Ac*T;
Bm=T\Bc;
Cm=Cc*T;
Dm=Dc;
disp(Dm,”Dm”,Cm,”Cm”, Bm,”Bm”,Am,”Am”,”Thermal System in modal canonical form”)
//
----------------------------------------------------------------------------
```

Scilab code Exa 7.11 Poles and Zeros of Tape Drive System

```
//Example 7.11
//Poles and Zeros of Tape Drive System.
//Also, Transform the system into modal form

xdel(winsid())//close all graphics Windows
clear;
clc;
//
----------------------------------------------------------------------------

// State space matrices of Tape Drive System

F=[0 2 0 0 0;
-0.1 -0.35 0.1 0.1 0.75;
0 0 0 2 0;
0.4 0.4 -0.4 -1.4 0;
0 -0.03 0 0 -1];
G=[0 0 0 0 1]’;
H2=[0 0 1 0 0];
H3=[0.5 0 0.5 0 0];
```

```
Ht=[-0.2 -0.2 0.2 0.2 0];
//-----------------------------------------------------------------------------
// Poles (eigen values) of the system
p=clean(spec(F));
disp(p,"Poles of Tape Drive System are")

//It requires complete state-space model.
sys=syslin('c',F,G,[Ht;H2;H3],[0;0;0])

// zeros of the system
[tr]=trzeros(sys)
disp(tr,"Transmission zeros of Tape Drive System are ")
//-----------------------------------------------------------------------------
// State space representation in modal canonical form with H3 output only.

[m Am1]=spec(F)
T1=[1/2 -%i/2;1/2 %i/2];
//transformation for a complex pair of eigen values.
temp=eye(5,5);
T=[T1 zeros(2,3);zeros(3,2) eye(3,3)];
temp(1,1)=-1; temp(2,2)=-1; //for change in input output signs as desired
M=m*T*temp //real Modal transformation

Am=clean(M\F*M);
Bm=clean(M\G);
Cm=clean(H3*M);
Dm=0;

disp(Dm,"Dm",Cm,"Cm", Bm,"Bm",Am,"Am","Tape Drive System in modal canonical form")
//
```

```
    ----------------------------------------------------------------------
```

Scilab code Exa 7.12 Transformation of Thermal System from state description

```
//Example 7.12
//Transformation of Thermal System from state
   description

xdel(winsid())//close all graphics Windows
clear;
clc;
//
   ----------------------------------------------------------------------

// State space model of Thermal System
s=%s;
F=[-7 -12; 1 0];
G=[1;0];
H=[1 2];
J=0;
sys=syslin('c',F,G,H,J)
//
   ----------------------------------------------------------------------

//Transfer function model of Thermal System
[ch num den]=ss2tf(sys);
disp(num/den, "G=","Transfer function model of
   Thermal System")
//
   ----------------------------------------------------------------------
```

Scilab code Exa 7.13 Zeros for the Thermal System from a State Description

```
//Example 7.13
//Zeros for the Thermal System from a State Description

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------------
// State space model of the given system
F=[-7 -12; 1 0];
G=[1;0];
H=[1 2];
J=0;
sysG=syslin('c',F,G,H,J)
//------------------------------------------------------------------------------
//Transfer function model
[d num den]=ss2tf(sysG);
zr=roots(num);
disp(zr,'zr=');
//Alternately, it can be obtained as
zr=trzeros(sysG);
disp(zr,'zr=');
//------------------------------------------------------------------------------
```

Scilab code Exa 7.14 Analysis of state equations of Tape Drive

```
//Example 7.14
//Analysis of state equations of Tape Drive.
//compute the poles, zeros  and transfer function of Tape Drive System.

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------

// State space matrices of Tape Drive System

F=[0 2 0 0 0;
-0.1 -0.35 0.1 0.1 0.75;
0 0 0 2 0;
0.4 0.4 -0.4 -1.4 0;
0 -0.03 0 0 -1];
G=[0 0 0 0 1]';
H2=[0 0 1 0 0];
H3=[0.5 0 0.5 0 0];
Ht=[-0.2 -0.2 0.2 0.2 0];
//------------------------------------------------------------------

//Poles (eigen values) of the system
p=clean(spec(F));

disp(p,"P","Poles of Tape Drive System are (for any output)")
disp("
************************************************************
```

```
")


disp("pole and zero polynomials and transfer function...
 for a system with output H2")
sys2=syslin('c',F,G,H2,0);
[d2 num2 den2]=ss2tf(sys2);
N2=coeff(num2);
D2=coeff(den2);
disp(D2,"D2",N2,"N2")
// zeros of the system with output H2
[zer2]=trzeros(sys2)
disp(zer2,"ZER2","zeros are")
// transfer function of the system with output H2
G2=clean(num2/den2);
disp(G2,"G2(s)=N2(s)/D2(s)=")
disp("************************************************************")

disp("pole and zero polynomials and transfer function for a...
 system with output H3")
sys3=syslin('c',F,G,H3,0);
[d3 num3 den3]=ss2tf(sys3);
N3=coeff(num3);
D3=coeff(den3);
disp(D3,"D3",N3,"N3")
// zeros of the system with output H3
[zer3]=trzeros(sys3)
disp(zer3,"ZER3","zeros are")
// transfer function of the system with output H3
G3=clean(num2/den2);
disp(G3,"G3(s)=N3(s)/D3(s)=")
disp("************************************************************")
```

```


disp("pole and zero polynomials and transfer function for a...
 system with output Ht")
syst=syslin('c',F,G,Ht,0);
[dt numt dent]=ss2tf(syst);
Nt=coeff(numt);
Dt=coeff(dent);
disp(Dt,"Dt",Nt,"Nt","zeros are")
// zeros of the system with output Ht
[zert]=trzeros(syst)
disp(zert,"ZERT")
// transfer function of the system with output Ht
Gt=clean(numt/dent);
disp(Gt,"G(s)=Nt(s)/Dt(s)=")
disp("************************************************************")
//------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.15 Control law for a pendulum

```
//Example 7.15
//Control law for a pendulum.

xdel(winsid())//close all graphics Windows
clear;
clc;
```

```
//--------------------------------------------------------------------------------------------------------------------------------
//Pendulum state model;
w0=1;

F=[0 1;-w0^2 0];
G=[0 1]';
H=eye(2,2); //representing x1 and x2 states as outputs
J=[0 0]';

sys=syslin('c',F,G,H,J); //open loop system

x0=[1 0]' //initial condition
t=0:0.2:7;
y=csim('impulse',t,sys); //open loop response
//--------------------------------------------------------------------------------------------------------------------------------

//simulation for closed loop system
x0=[1 0]' //initial condition

//control law u=-Kx;
K=[3*w0^2 4*w0];
syscl=syslin('c',(F-G*K),G,H,J); //closed loop system


t=0:0.1:7;
u=zeros(1,length(t));
[x z]=csim(u,t,syscl,x0); //closed loop response
plot(t',x');

u=-K*x;
plot(t',u'/4,'r--'); //control law u plot (scaled to 1/4 in figure);
```

```
legend("x1","x2","u/4")

//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title('Impulse response of undamped oscillator with full-state...
 feedback(w0=1)','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('Amplitude','fontsize',2)
//------------------------------------------------------------------------
```

---

check Appendix AP 2 for dependency:

`acker_dk.sci`

Scilab code Exa 7.16 Ackermanns formula for undamped oscillator

```
//Example 7.16
//Ackermann's formula for undamped oscillator.

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------

//undamped oscillator (Pendulum) state model;
w0=1;

F=[0 1;-w0^2 0];
G=[0 1]';
```

```
H=eye(2,2); //representing x1 and x2 states as outputs
J=[0 0]';
//-------------------------------------------------------------------------------------
//Ackermann's formula for feedback gain computation

pc=[-2 -2];    //desired poles
exec('./acker_dk.sci', -1);
[K,eig]=acker_dk(F,G,pc)
disp(K,"Feedback gain K=")
disp(eig,"Closed loop eigen values are ")
//-------------------------------------------------------------------------------------
```

---

check Appendix AP 2 for dependency:

`acker_dk.sci`

Scilab code Exa 7.17 How zero location affect control law

```
//Example 7.17 How zero location affect control law
// Obtain state feedback gain matrix for the given system

xdel(winsid())//close all graphics Windows
clear;
clc;
//-------------------------------------------------------------------------------------
//(a) state feedback gain matrix for zero at 2.
//Location of system Zero
z0=2;
```

```

// State space representation
Ao=[-7 1;-12 0];
Bo=[1 -z0]';
Co=[1 0];
Do=0;

// Desired poles
Pd=[1 2 4];
Pc=roots(Pd);


// State feedback gain matrix for system zero at -2.0
K=ppol(Ao,Bo,Pc)
disp(K,"K=","State feeback gain for a system with zero at 2" )
// ------------------------------------------------------------------------

//Location of system Zero
z0=-2.99
B=[1 -z0]';
// State feedback gain matrix for system zero...
// at -2.99 (by ackermann's formula)
exec('./acker_dk.sci', -1);
K1=acker_dk(Ao,B,Pc)
disp(K1,"K1","State feeback gain for a system with zero at -2.99")
// ------------------------------------------------------------------------
```

---

check Appendix AP 2 for dependency:

`acker_dk.sci`

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.18 Introducing the reference input

```
//Example 7.18
//Introducing the reference input.

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------------------------
//Pendulum state model;
w0=1;

F=[0 1;-w0^2 0];
G=[0 1]';
H=[1 0]; //representing x1 and x2 states as outputs
J=0;
n=sqrt(length(F));

//computing state feedback matrix to place poles at [-2 -2]
exec('./acker_dk.sci', -1);
K=acker_dk(F,G,[-2, -2]);
//------------------------------------------------------------------------------------------
//augmented matrix for tracking the reference
A=[F G;H J];
N=A\[zeros(1,n) 1]';
Nx=N(1:n);
Nu=N(n+1);

//feedforward gain (input weight)
```

```
Ntilde=Nu+K*Nx;

//--------------------------------------
//Alternately, it can be computed as /
Ntilde1=-inv(H*inv(F-G*K)*G); //      /
//--------------------------------------

//Closed loop system and step response
syscl=syslin('c',(F-G*K),G*Ntilde,H,J); //closed loop system

t=0:0.1:7;
[y x]=csim('step',t,syscl); //closed loop response
plot(t',x');

u=-K*x+Ntilde;
plot(t',u'/4,'r--'); //control law u plot (scaled to 1/4 in figure);
legend("x1","x2","u/4");
xset('font size',3);
xstring(5,0.93,"$x_{ss}$")
xstring(5,0.25,"$u_{ss}$")

//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title('Step response of undamped oscillator to reference input',...
'fontsize',3);
xlabel('Time t (sec.)','fontsize',2);
ylabel('Amplitude','fontsize',2);

//------------------------------------------------------------------------------------------
```

---

check Appendix AP 2 for dependency:

`acker_dk.sci`

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.19 Reference input to Type1 control system DC Motor

```
//Example 7.19
//Reference input to Type-1 control system: DC Motor

xdel(winsid())//close all graphics Windows
clear;
clc;
//
  ----------------------------------------------------------------------

//Location of system Zero
z0=2;

// State space representation
F=[0 1;0 -1];
G=[0 1]';
H=[1 0];
J=0;
n=sqrt(length(F)); //order of the system

//computing state feedback matrix to place poles at
  assumed location [-1 -2]
exec('./acker_dk.sci', -1);
K=acker_dk(F,G,[-1, -2]); //assume pd=[-1 -2]
```

```
//--------------------------------------------------------------------------------------
//augmented matrix for tracking the reference
A=[F G;H J];
N=A\[zeros(1,n) 1]';
Nx=N(1:n);
Nu=N(n+1);
disp(Nx,"Nx",Nu,"Nu")

//feedforward gain (input weight)
Ntilde=Nu+K*Nx;
disp(Ntilde,"N_tilde","Input gain: N_tilde =Nu+K Nx")
//--------------------------------------------------------------------------------------
// Verify if ||y-r|| -> 0;

syscl=syslin('c',(F-G*K),G*Ntilde,H,J); //closed loop system

t=0:0.1:10;
r=ones(1,length(t));//reference input
[y x]=csim('step',t,syscl); //closed loop response

e=sqrt((r-y).^2) //norm of error
plot(t,y);
plot(t,r,'m:'); //reference input
plot(t,e,'r-.'); //norm of error
xset('font size',3);
xstring(3,0.83,"y")
xstring(2,1,"r")
xstring(3,0.1,"$\|e\|$")
//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Step response of undamped oscillator to
```

```
    reference input','fontsize',3);
xlabel('Time t (sec.)','fontsize',2);
ylabel('Amplitude','fontsize',2);
zoom_rect([0 -0.1 10 1.1])
//
    ------------------------------------------------------------------
```

---

check Appendix AP 2 for dependency:

`acker_dk.sci`

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.20 Pole Placement as a Dominant Second Order System

```
//Example 7.20
// Pole Placement as a Dominant Second-Order System

xdel(winsid())//close all graphics Windows
clear;
clc;
//
    ------------------------------------------------------------------

clc;
clear all;

// State space representation
F=[0 2 0 0 0;-0.10 -0.35 0.1 0.1 0.75; 0 0 0 2 0;...
 0.4 0.4 -0.4 -1.4 0; 0 -0.03 0 0 -1];
```

```
G=[0 0 0 0 1]';
H=[0.5 0 0.5 0 0]; //Tape position at the head
Ht=[-0.2 -0.2 0.2 0.2 0]; //Tension output
J=0;
n=sqrt(length(F))
// Desired poles
Pc=[-0.707+0.707*%i -0.707-0.707*%i -4 -4 -4]/1.5;
//-----------------------------------------------------------------------
// State feedback gain matrix via LQR (riccati equation)
Q = eye(5,5);
R =1
// Riccati equation
P=riccati(F, G*inv(R)*G', Q, 'c')
K1=inv(R)*G'*P
//-----------------------------------------------------------------------
// State feedback gain matrix via pole-placement
exec('./acker_dk.sci', -1);
K2=acker_dk(F,G,Pc);
disp(K2,'K2=',"Gain by ackermans formula" );
//-----------------------------------------------------------------------
Ntilde1=-inv(H*inv(F-G*K1)*G); //input gain for LQR feedback gain.
Ntilde2=-inv(H*inv(F-G*K2)*G); //input gain for Ackerman's feedback gain.

syscl1=syslin('c',(F-G*K1),G*Ntilde1,H,J); //closed loop system with K1
syscl2=syslin('c',(F-G*K2),G*Ntilde2,H,J); //closed loop system with K2

t=0:0.1:12;
```

```
[y1 x1]=csim('step',t,syscl1); //response of position head with K1
[y2 x2]=csim('step',t,syscl2); //response of position head with K2

//plot of a position of read write head
plot(t,y1,"m-."); //Design via LQR
plot(t,y2,2); //Design via Ackerman's Formula

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Step response of tape servomotor designs','fontsize',3);
xlabel('Time t (sec.)','fontsize',2);
ylabel('Tape Posotion','fontsize',2);

xstring(2.5,1.1,"LQR")
xarrows([3;4],[1.1;0.95],-1,1)
xstring(5,0.7,["Dominant";"second order"])
xarrows([5;4.2],[0.8;0.9],-1.5,1)
//--------------------------------------------------------------------------------


//response as a tape tension
yt1=Ht*x1;
yt2=Ht*x2;

figure(1)
plot(t,yt1,"m-."); //Design via LQR
plot(t,yt2,2); //Design via Ackerman's Formula

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Tension plots for tape servomotor step responses','fontsize',3);
```

```
xlabel('Time t (sec.)','fontsize',2);
ylabel('Tape Tension','fontsize',2);

xstring(3.5,0,"LQR")
xarrows([3.7;4.7],[0;0],-1)
xstring(6.1,-0.015,["Dominant";"second order"])
xarrows([6;6],[-0.013;-0.002],-1)
//--------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.21 Symmetric root locus for servo speed control

```
//Example 7.21
// Symmetric root locus (SRL) for servo speed control

xdel(winsid())//close all graphics Windows
clear;
clc;
//--------------------------------------------------------------------------

//Transfer function model of the given system
a=1.5;//assume
s=poly(0,'s');
nums=1;
dens=s+a;
```

```
num_s=1;
den_s=-s+a;
G0s=syslin('c',nums/dens); //G0(s)
G0_s=syslin('c',num_s/den_s); //G0(−s)

evans(G0s)
evans(G0_s)
zoom_rect([-3 -0.1 3 0.1])
f=gca();
f.x_location = ”origin”
f.y_location = ”origin”
xset(”color”,2);
h=legend('');
h.visible = ”off”

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Symmetric root locus for a first order system','fontsize',3);
//-----------------------------------------------------------------------------------------------
//Root locus design
//rho>0; choose rho=2
rho=2;
//optimal pole p=−sqrt(aˆ2+rho)
p=-sqrt(a^2+rho)
sig=real(p);
omega=imag(p);
plot(sig,omega,'ro')
xstring(-2.5,0.02,[”pole location at”;”$\rho=2$”])
xarrows([-2.2;-2.07],[0.02;0.002],-1.5,1)
//-----------------------------------------------------------------------------------------------
```

check Appendix AP 2 for dependency:

`acker_dk.sci`

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.22 SRL design for satellite attitude control

```
//Example 7.22
// SRL design for satellite attitude control

xdel(winsid())//close all graphics Windows
clear;
clc;
//-----------------------------------------------------------------------------------------------
//Transfer function for satellite attitude control system
s=poly(0,'s');
nums=1;
dens=s^2;
num_s=1;
den_s=(-s)^2;
G0s=syslin('c',nums/dens); //G0(s)
G0_s=syslin('c',num_s/den_s); //G0(-s)
//evans(G0s*G0_s)
evans(1/s^4)
zoom_rect([-3 -3 3 3])
f=gca();
f.x_location = "origin"
f.y_location = "origin"
```

```
xset("color",2);
h=legend('');
h.visible = "off"

//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title('Symmetric root locus for the satellite','fontsize',3);
//--------------------------------------------------------------------------

//Root locus design
//choose rho=4.07 that places pole at -1+-j
rho=4.07;
chr_eqn=(1+rho*G0s*G0_s)
p=[-1+%i, -1-%i];
sig=real(p);
omega=imag(p);
plot(sig,omega,'ro')
xstring(-2.2,0.5,["pole locations at";"$\rho=4.07$"])
//--------------------------------------------------------------------------

//pole-placement design;
sys=tf2ss(G0s);
exec('./acker_dk.sci', -1);
K=acker_dk(sys.A,sys.B,p);
syscl=syslin('c',(sys.A-sys.B*K),sys.B, sys.C, sys.D)
disp(spec(syscl.A),"Closed loop eigen values");
//--------------------------------------------------------------------------
```

check Appendix AP 2 for dependency:

`acker_dk.sci`

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.23 SRL design for an inverted pendulum

```
//Example 7.23
// SRL Design for an Inverted Pendulum

xdel(winsid())//close all graphics Windows
clear;
clc;
//
   ----------------------------------------------------------------------

//Transfer function model of Inverted Pendulum.
s=poly(0,'s');

nums=-(s+2);
dens=(s^2-1)
num_s=-(-s+2);
den_s=((-s)^2-1)
G0s=syslin('c',nums/dens); //G0(s)
G0_s=syslin('c',num_s/den_s); //G0(-s)
sysGG=G0s*G0_s;
evans(sysGG)
title('Symmetric root locus for Inverted Pendulum')
zoom_rect([-3 -2 3 2])
f=gca();
```

```
f.x_location = ”origin”
f.y_location = ”origin”
xset(”color”,2);
h=legend('');
h.visible = ”off”

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Symmetric root locus for the inverted pendulum','fontsize',3);
//------------------------------------------------------------------------------
//Root locus design
//choose rho=1 that places pole at -1.36+-j0.606
rho=1;
p=[-1.36+0.606*%i, -1.36-0.606*%i];
sig=real(p);
omega=imag(p);
plot(sig,omega,'ro')
xstring(-1.25,0.5,[”pole locations at”;”$\rho=1$”])
//------------------------------------------------------------------------------
//pole-placement design;
Ac=[0 1;1 0];Bc=[0 -1]'; Cc=[2 1];Dc=0;
exec('./acker_dk.sci', -1);
K=acker_dk(Ac,Bc,p);
disp(K,”K=”,spec(Ac-Bc*K),”Closed loop eigen values”);

//input gain calculation
n=sqrt(length(Ac));
A=[Ac Bc;Cc Dc];
N=A\[zeros(1,n) 1]';
Nx=N(1:n);
Nu=N(n+1);
```

```

//feedforward gain (input gain)
Ntilde=Nu+K*Nx;

//Step respose
t=0:0.1:4.5;
syscl=syslin('c',(Ac-Bc*K),Bc*Ntilde, Cc, Dc)
[y x]=csim('step',t,syscl); //closed loop response
figure,
plot(t,y);

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Step response for inverted pendulum','fontsize',3);
xlabel('Time t (sec.)','fontsize',2);
ylabel(["Position","$x_1$"],'fontsize',2);
//-------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.24 LQR Design for a Tape Drive

```
//Example 7.24
// LQR Design for a Tape Drive

xdel(winsid())//close all graphics Windows
```

```
clear;
clc;
//------------------------------------------------------------------
// State space model for a Tape Drive
F=[0 2 0 0 0;-0.10 -0.35 0.1 0.1 0.75; 0 0 0 2 0; 0.4 0.4 -0.4 -1.4 0; 0 -0.03 0 0 -1];
G=[0 0 0 0 1]';
H3=[0.5 0 0.5 0 0];
//------------------------------------------------------------------
// State feedback gain matrix via LQR (riccati equation)
// (a) Continuous LQR for rho=1
rho=1;
R1=1;
Q1=rho*H3'*H3;
// Riccati equation
P1=riccati(F, G*inv(R1)*G', Q1, 'c')
K1=inv(R1)*G'*P1
disp(K1,'K1')
//------------------------------------------------------------------
// State feedback gain matrix via LQR (riccati equation)
// (a) Comparision in step response with rho=0.1,1,10.
rho=0.1;
R2=1;
Q2=rho*H3'*H3;
// Riccati equation
P2=riccati(F, G*inv(R2)*G', Q2, 'c')
K2=inv(R2)*G'*P2

rho=10;
```

```
R3=1;
Q3=rho*H3'*H3;
// Riccati equation
P3=riccati(F, G*inv(R3)*G', Q3, 'c')
K3=inv(R3)*G'*P3
//------------------------------------------------------------------------------------------
//input gains for step reference with rho=0.1,1,10.
Ntilde1=-inv(H3*inv(F-G*K1)*G);
Ntilde2=-inv(H3*inv(F-G*K2)*G);
Ntilde3=-inv(H3*inv(F-G*K3)*G);

//Closed loop system with rho=0.1,1,10.
syscl1=syslin('c',(F-G*K1),G*Ntilde1,H3,0);
syscl2=syslin('c',(F-G*K2),G*Ntilde2,H3,0);
syscl3=syslin('c',(F-G*K3),G*Ntilde3,H3,0);

//step response with rho=0.1,1,10.
t=0:0.1:12;
[y1 x1]=csim('step',t,syscl1); //closed loop response
[y2 x2]=csim('step',t,syscl2); //closed loop response
[y3 x3]=csim('step',t,syscl3); //closed loop response

figure,
a1=newaxes();
a1.axes_bounds=[0,0,1.0,0.5];
plot(t,y1);
plot(t,y2,'r-.');
plot(t,y3,'m:');

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('(a)Step response of step servo motor for LQR
```

```
Design','fontsize',3);
xlabel('Time t (sec.)','fontsize',2);
ylabel(["Tape Position","$x_3$"],'fontsize',2);

xstring(4.1,0.85,"$\rho=1$")
xstring(5.5,0.75,"$\rho=0.1$")
xstring(2.1,1.05,"$\rho=10$")
//----------------------------------------------------------------------------

//Tensions for the Tape
//For tape output is Ht=[-0.2 -0.2 0.2 0.2 0];
Ht=[-0.2 -0.2 0.2 0.2 0];
H3=Ht;
//input gains can not be computed because of singularity. so set it 1;
Ntilde1=1;
Ntilde2=1;
Ntilde3=1;

//Closed loop system with rho=0.1,1,10.
syscl1=syslin('c',(F-G*K1),G*Ntilde1,H3,0);
syscl2=syslin('c',(F-G*K2),G*Ntilde2,H3,0);
syscl3=syslin('c',(F-G*K3),G*Ntilde3,H3,0);

//step response with rho=0.1,1,10.
t=0:0.1:12;
[y1 x1]=csim('step',t,syscl1); //closed loop response
[y2 x2]=csim('step',t,syscl2); //closed loop response
[y3 x3]=csim('step',t,syscl3); //closed loop response

a2=newaxes();
a2.axes_bounds=[0,0.5,1.0,0.5];
plot(t,y1);
plot(t,y2,'r-.');
```

```
plot(t,y3,'m:');

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('(b)Corresponding tension for Tape servomotor step response','fontsize',3);
xlabel('Time t (sec.)','fontsize',2);
ylabel(["Tape Tension","T"],'fontsize',2);


xstring(4.3,-0.05,"$\rho=1$")
xstring(6,-0.1,"$\rho=0.1$")
xstring(1.5,-0.03,"$\rho=10$")
//--------------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.25 An estimator design for a simple pendulum

```
//Example 7.25
// An estimator design for a simple pendulum

xdel(winsid())//close all graphics Windows
clear;
clc;
//--------------------------------------------------------------------------------

// State space representation
```

```
w0=1;
F=[0 1; -w0^2 0];
G=[0 1]';
H=[1 0];
J=0;
n=sqrt(length(F));//order of the system
// Desired estimator poles
Pe=[-10*w0 -10*w0];
// Observer gain matrix for system
Lt=ppol(F',H',Pe);
L=Lt';
disp(L,"L=");
//
  ------------------------------------------------------------------

//simulation for closed loop system
x0=[1 0]' //initial condition

//State feedback control law u=-Kx; (from Ex7_15)
K=[3*w0^2 4*w0];
//
  ------------------------------------------------------------------

//Augmented plant and observer
Faug=[F-G*K, zeros(n,n); L*H, F-L*H];
Gaug=[0 0 0 0]';
Haug=[H -H];
Jaug=0;

sys_aug=syslin('c',Faug,Gaug,Haug,Jaug);
t=0:0.1:4;
u=zeros(1,length(t));
x0=[1 0 0 0]';
[x z]=csim(u,t,sys_aug,x0); //closed loop response
plot(t,z(1,:));
plot(t,z(2,:),'m');
plot(t,z(3,:),'b:');
plot(t,z(4,:),'m:');
```

```

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title(['Initial condition response of oscillator showing',...
'$\mathbf{x}$','and','$\hat{\mathbf{x}}$'],'fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('Amplitude','fontsize',2)
legend('$x_1$','$x_2$','$\hat{x}_1$','$\hat{x}_2$')
xset('font size',2)
//-----------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.26 A reduced order estimator design for pendulum

```
//Example 7.26
// A reduced order estimator design for pendulum

xdel(winsid())//close all graphics Windows
clear;
clc;
//-----------------------------------------------------------------------------

// State space representation
w0=1;
F=[0 1; -w0^2 0];
```

```
G=[0 1]';
H=[1 0];
J=0;
n=sqrt(length(F));//order of the system

//partioned system
Faa=F(1,1); Fab=F(1,2);
Fba=F(2,1); Fbb=F(2,2);

// Desired estimator poles
Pe=[-10];
// Observer gain matrix for system
L=ppol(Fbb',Fab',Pe);
L=L';
disp(L,"L=" );
//--------------------------------------------------------------------------------
//simulation for closed loop system
x0=[1 0 10]' //initial condition

//State feedback control law u=-Kx; (from Ex7_15)
K=[3*w0^2 4*w0];
//--------------------------------------------------------------------------------
//Augmented plant and observer
Faug=[F-G*K, zeros(n,1); Fab, L*Fab, Fbb-L*Fab];
Gaug=[0 0 0]';
Haug=[H 0];
J=0;

sys_aug=syslin('c',Faug,Gaug,Haug,J);
t=0:0.1:4;
u=zeros(1,length(t));
[x z]=csim(u,t,sys_aug,x0); //closed loop response
plot(t,z(1,:),'b');
plot(t,z(2,:),'r');
```

```
plot(t,z(3,:),'r--');


//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Initial condition response of the reduced order estimator','fontsize',3)
xlabel('Time t (sec.)','fontsize',2)
ylabel('Amplitude','fontsize',2)
legend('$x_1$','$x_2$','$\hat{x}_2$')
xset('font size',2)
//--------------------------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.27 SRL estimator design for a simple pendulum

```
//Example 7.27
// SRL estimator design for a simple pendulum

xdel(winsid())//close all graphics Windows
clear;
clc;
//--------------------------------------------------------------------------------------------

// State space representation
F=[0 1; -1 0];
G=[0 1]';
```

```
H=[1 0];
J=0;

//Transfer function
sys=syslin('c',F,G,H,J)
sysGG=ss2tf(sys)

//Symmetric root locus for the inverted pendulum estimator design
//----------------------------------------------------------------------
//Root locus design
evans(sysGG*sysGG)
zoom_rect([-5 -5 5 5])
f=gca();
f.x_location = "origin"
f.y_location = "origin"
xset("color",2);
h=legend('');
h.visible = "off"
//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Symmetric root locus for inverted the pendulum estimator design',...
'fontsize',3);
//----------------------------------------------------------------------
//pole locations for q=365; p=-3+-j3.18
p=[-3+3.18*%i -3-3.18*%i]
sig=real(p);
omega=imag(p);
plot(sig,omega,'ro')
xstring(-4,1,["pole location at";"q=365"])
xarrows([-3.5;-3.05],[2;3.1],-1.5,1)
//
```

```
________________________________________________________________________________
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

check Appendix AP 3 for dependency:

`zpk_dk.sci`

Scilab code Exa 7.28 Full order compensator design for satellite attitude control

```
//Example 7.28
// Full order compensator design for satellite attitude control.

xdel(winsid())//close all graphics Windows
clear;
clc;
//________________________________________________________________________________

// State space representation
A=[0 1; 0 0];
B=[0 1]';
C=[1 0];
D=0;
n=sqrt(length(A));
//Desired poles for the satellite attitude control system.
Pc=[-0.707+0.707*%i -0.707-0.707*%i ]

```

```
// State feedback gain
K=ppol(A,B,Pc)
disp(K,'K=',"State feedback gain")

//Estimator - error roots are at
Pe=[-2.5+4.3*%i -2.5-4.3*%i]
L=ppol(A',C',Pe);
L=L';
disp(L,'L=',"Observer gain")
//
  ------------------------------------------------------------------------------

//Compensator Design
sys1=syslin('c',A,B,C,D);
G=ss2tf(sys1);
s=poly(0,'s');

Ds=-K*inv(s*eye(n,n)-A+B*K+L*C)*L;

exec('./zpk_dk.sci', -1);
[pl,zr Kp]=zpk_dk(Ds);
D=poly(zr,'s','roots')/poly(pl,'s','roots')

evans(G*D)
zoom_rect([-8 -6 8 6])

f=gca();
f.x_location = "origin"
f.y_location = "origin"
xset("color",2);
h=legend('');
h.visible = "off"

//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting
    figure properties
title('Root locus for combined control and estimator
  ,...
```

```
 with process gain as the parameter','fontsize',3);
//------------------------------------------------------------------------------------
//Frequnecy response for 1/s^2 and compensated

figure,
bode([-Ds*G;G],0.01/2/%pi,100/2/%pi,"rad");
title(["Frequency response for","$G(s)=1/s^2$"],'fontsize',3)
legend('Compensated','Uncompensated')
exec .\fig_settings.sci; //custom script for setting figure properties
//------------------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

check Appendix AP 3 for dependency:

`zpk_dk.sci`

Scilab code Exa 7.29 A reduced order compensator design for a satellite attitude control

```
//Example 7.29
// A reduced order compensator design for a satellite attitude control

xdel(winsid())//close all graphics Windows
```

```
clear;
clc;
//------------------------------------------------------------------------------
// State space representation
F=[0 1;0 0];
G=[0 1]';
H=[1 0];
J=0;
n=sqrt(length(F));//order of the system

//partioned system
Faa=F(1,1); Fab=F(1,2);
Fba=F(2,1); Fbb=F(2,2);
Ga=G(1);Gb=G(2);

// Desired estimator poles
Pe=[-5];
// Observer gain matrix for system
L=ppol(Fbb',Fab',Pe);
L=L';
disp(L,"L=" );
//------------------------------------------------------------------------------
//State feedback control law u=-Kx=-(K+[L*k2 0])[y xc]';
k1=1; k2=sqrt(2);
K=[k1 k2];
Kc=K+[L*k2 0];
//------------------------------------------------------------------------------
//compensator differential equation
//xc_dot=(Fbb-L*Fab)*xb_hat + (Fba - L*Faa)*y + (Gb - L*Ga)*u
//xc_dot=((Fbb-L*Fab)-k2)*xc + [(Fba - L*Faa)-(Gb -
```

```
L*Ga)*(k1+L*k2)+L*(Fbb-L*Fab)]*y
Fc=(Fbb-L*Fab)-Gb*k2
Fy=(Fba - L*Faa)-(Gb - L*Ga)*(k1+k2*L)+(Fbb-L*Fab)*L
//compensator transfer function
s=poly(0,'s');
Gest=syslin('c',Fy/(s-Fc))//estimator transfer function
Dcr=-[k1+L*k2+k2*Gest]
disp(Dcr,'Dcr','compensator transfer function')
//-------------------------------------------------------------------------

//Root locus with reduced order compensator
G=1/s^2;
G=syslin('c',G);
exec('./zpk_dk.sci', -1);
[pl,zr Kp]=zpk_dk(Dcr);

Dcr=poly(zr,'s','roots')/poly(pl,'s','roots')
Dcr=syslin('c',Dcr);
evans(G*Dcr)
zoom_rect([-8 -4  2 4])

f=gca();
f.x_location = "origin"
f.y_location = "origin"
xset("color",2);
h=legend('');
h.visible = "off"

//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title(['Root locus of a reduced order controller and ',"$1/s^2$",...
 "process"],'fontsize',3);
//-------------------------------------------------------------------------
```

```
//Frequnecy response for 1/s^2 and compensated

figure,
bode([-Kp*G*Dcr;G],0.01/2/%pi,100/2/%pi,"rad");
title(["Frequency response","$G(s)=1/s^2$", "with a reduced...
 order estimator"],'fontsize',3)
exec .\fig_settings.sci; //custom script for setting figure properties
legend('Compensated','Uncompensated')
//-----------------------------------------------------------------------------
```

check Appendix AP 1 for dependency:

`fig_settings.sci`

check Appendix AP 3 for dependency:

`zpk_dk.sci`

Scilab code Exa 7.30 Full Order Compensator Design for DC Servo

```
//Example 7.30
// Full-Order Compensator Design for DC Servo.

xdel(winsid())//close all graphics Windows
clear;
clc;
//-----------------------------------------------------------------------------
```

```

// State space representation
//Transfer function model for DC Servo
s=poly(0,'s');
num=10;
den=s*(s+2)*(s+8);
Gs=syslin('c',num/den);

// State space representation
F=[-10 1 0;-16 0 1;0 0 0];
G=[0 0 10]';
H=[1 0 0];
J=0;
n=sqrt(length(F));
//Desired poles for the DC Servo system.
Pc=[-1.42 -1.04+2.14*%i -1.04-2.14*%i ]


// State feedback gain
K=ppol(F,G,Pc)
disp(K,'K=',"State feedback gain")

//Estimator - error roots are at
Pe=[-4.25 -3.13+6.41*%i -3.13-6.41*%i]
L=ppol(F',H',Pe);
L=L';
disp(L,'L=',"Observer gain")
//
  ------------------------------------------------------------------------------------------------
//Compensator Design
DK=-K*inv(s*eye(n,n)-F+G*K+L*H)*L;

exec('./zpk_dk.sci', -1);
[p,z]=zpk_dk(DK);
D=poly(z,'s','roots')/poly(p,'s','roots')

evans(Gs*D)
```

```
zoom_rect([-8 -9 3 9])

f=gca();
f.x_location = "origin"
f.y_location = "origin"
xset("color",2);
h=legend('');
h.visible = "off"

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Root locus for DC servo pole assignment','fontsize',3);
//-----------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.31 Reduced Order Estimatortor Design for DC Servo

```
//Example 7.31
// Reduced-Order Estimator Design for DC Servo.

xdel(winsid())//close all graphics Windows
clear;
clc;
//-----------------------------------------------------------------------------

```

```
// State space representation
//Transfer function model for DC Servo
s=poly(0,'s');
num=10;
den=s*(s+2)*(s+8);
Gs=syslin('c',num/den);
// State space representation
F=[-10 1 0;-16 0 1;0 0 0]
G=[0 0 10]';
H=[1 0 0];
J=0;
n=sqrt(length(F));
//Desired poles for the DC Servo system.
Pc=[-1.42 -1.04+2.14*%i -1.04-2.14*%i ]
// State feedback gain
K=ppol(F,G,Pc)
disp(K,'K=',"State feedback gain")

//
  ----------------------------------------------------------------------

//Estimator - error roots are at
//partioned system
Faa=F(1,1); Fab=F(1,2:3);
Fba=F(3,1); Fbb=F(2:3,2:3);
Ga=G(1);Gb=G(2:3);

Pe=[-4.24+4.24*%i, -4.24-4.24*%i]
// Observer gain matrix for system
L=ppol(Fbb',Fab',Pe);
L=L';
disp(L,"L=" );
//
  ----------------------------------------------------------------------


//State feedback control law u=-Kx=-(K+[L*k2 0])[y
   xc]';
```

```
k1=K(1); k2=K(2:3);

//--------------------------------------------------------------------------------------------------------------

//compensator transfer function
s=poly(0,'s');
num=(-0.735+s)*(1.871+s);
den=poly([-0.990 + 6.12* %i, -0.990 - 6.12* %i] ,'s','roots')
Dcr=syslin('c',num/den);
disp(Dcr,'Dcr','compensator transfer function')
//--------------------------------------------------------------------------------------------------------------

//Root locus with reduced order compensator
evans(-Dcr*Gs)
zoom_rect([-8 -9 3 9])

f=gca();
f.x_location = "origin"
f.y_location = "origin"
xset("color",2);
h=legend('');
h.visible = "off"

//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Root locus for DC servo reduced order controller','fontsize',3);
//--------------------------------------------------------------------------------------------------------------
```

---

check Appendix AP 2 for dependency:

`acker_dk.sci`

check Appendix AP 1 for dependency:

`fig_settings.sci`

check Appendix AP 3 for dependency:

`zpk_dk.sci`

Scilab code Exa 7.32 Redesign of the Dc servo compensator using SRL

```
//Example 7.32
// Redesign of the Dc servo compensator using SRL

xdel(winsid())//close all graphics Windows
clear;
clc;
//
   ---------------------------------------------------------------------

// State space representation
//Transfer function model for DC Servo
s=poly(0,'s');
num=10;
den=s*(s+2)*(s+8);
Gs=syslin('c',num/den);

// State space representation
F=[-10 1 0;-16 0 1;0 0 0]
G=[0 0 10]';
H=[1 0 0];
J=0;
n=sqrt(length(F));
//Desired poles for the DC Servo system.
Pc=[-2+1.56*%i -2-1.56*%i -8.04]
```

```


// State feedback gain
K=ppol(F,G,Pc)
disp(K,'K=',"State feedback gain")

//Estimator - error roots are at
Pe=[-4+4.49*%i -4-4.49*%i -9.169]
exec .\acker_dk.sci;
Lt=ppol(F',H',Pe);
L=clean(Lt');
disp(L,'L=',"Observer gain")
//Error in book, Gain values are different in book.
//-----------------------------------------------------------------------------
//Compensator Design
DK=-K*inv(s*eye(n,n)-F+G*K+L*H)*L;
DK=syslin('c',DK)
exec('./zpk_dk.sci', -1);
[pl,zr,Kp]=zpk_dk(DK);
Dc=poly(zr,'s','roots')/poly(pl,'s','roots')
//-----------------------------------------------------------------------------
//symmetric root locus
G_s=horner(Gs,-s);
evans(Gs*G_s)
zoom_rect([-10 -5 10 5])
f=gca();
f.x_location = "origin"
f.y_location = "origin"
xset("color",2);
h=legend('');
h.visible = "off"
//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
```

```
title('Symmetric root locus','fontsize',3);
//------------------------------------------------------------------------------------------

//root locus
figure,
evans(Gs*Dc) //Correct root locus
zoom_rect([-11 -6 1 6])
f=gca();
f.x_location = "origin"
f.y_location = "origin"
xset("color",2);
h=legend('');
h.visible = "off"
//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for setting figure properties
title('Root locus for pole assignment from the SRL','fontsize',3);
//------------------------------------------------------------------------------------------

//Discrete-time controller
nc=94.5*conv([7.98 1],[2.52 1])
dc=conv([59.5348 8.56 1],[10.6 1])
sysDc=poly(nc,'s','coeff')/poly(dc,'s','coeff');
sysDc_ss=syslin('c',tf2ss(sysDc));
ts=0.1;
sysDd=dscr(sysDc_ss,ts)
Gdz=ss2tf(sysDd);

disp(sysDc,"Continuous-time compensator")
disp(Gdz,"Discrete-time compensator")
//------------------------------------------------------------------------------------------

//step responses
importXcosDiagram(".\Ex7_32_model.xcos")
```

```

xcos_simulate(scs_m,4);
scs_m.props.context
figure,
plot(yt.time,yt.values(:,1),2)
plot(yt.time,yt.values(:,2),'r--')
xlabel('Time (sec)');
ylabel('y');
title("Comaprison of step responses for continuous and discrete...
 controllers",'fontsize',3)
exec .\fig_settings.sci; //custom script for setting figure properties
legend("continuous controller","digital controller",4)

//Control inputs
figure,
plot(ut.time,ut.values(:,1),2)
plot(ut.time,ut.values(:,2),'r--')
xlabel('Time (sec)');
ylabel('u');
title("Comaprison of control signals for continuous and discrete...
 controllers",'fontsize',3)
exec .\fig_settings.sci; //custom script for setting figure properties
legend("continuous controller","digital controller")
//--------------------------------------------------------------------------------
```

This code can be downloaded from the website wwww.scilab.in check Appendix AP 2 for dependency:

`acker_dk.sci`

check Appendix AP 3 for dependency:

`zpk_dk.sci`

Scilab code Exa 7.33 DC servo system redesign with modified with dominant second order pole locations

```
//Example 7.33
// DC servo system redesign with modified with
   dominant second
// order pole locations.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
   ------------------------------------------------------------------------------------


// State space representation
//Transfer function model for DC Servo
s=poly(0,'s');
num=10;
den=s*(s+2)*(s+8);
Gs=syslin('c',num/den);
```

```

// State space representation
F=[-10 1 0;-16 0 1;0 0 0]
G=[0 0 10]';
H=[1 0 0];
J=0;
n=sqrt(length(F));
//Desired poles for the DC Servo system.
Pc=[-1.41+1.41*%i -1.41-1.41*%i -8]


// State feedback gain
K=ppol(F,G,Pc)
disp(K,'K=',"State feedback gain")

//Estimator - error roots are at
Pe=[-4.24+4.24*%i -4.24-4.24*%i -8]
exec .\acker_dk.sci;
Lt=ppol(F',H',Pe);
L=clean(Lt');
disp(L,'L=',"Observer gain")
//Error in book, Gain values are different in book.
//---------------------------------------------------------------------------
//Compensator Design
DK=-K*inv(s*eye(n,n)-F+G*K+L*H)*L;
DK=syslin('c',DK)
exec('./zpk_dk.sci', -1);
[pl,zr,Kp]=zpk_dk(DK*10);
disp(zr,"zeros",pl,"Poles",Kp*10,"Gain(includung system gain)")
Dcs=poly(zr,'s','roots')/poly(pl,'s','roots')
disp(Dcs,'Dcs=',"Compensator transfer function")
//---------------------------------------------------------------------------
```

check Appendix AP 2 for dependency:

`acker_dk.sci`

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.34 Servomechanism increasing the velocity constant through zero assignment

```
//Example 7.34
// Servomechanism, increasing the velocity constant through
// zero assignment.

xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------------

// State space representation
//Transfer function model for DC Servo
s=poly(0,'s');
num=1;
den=s*(s+1);
Gs=syslin('c',num/den);

// State space representation
F=[0 1;0 -1]
G=[0 1]';
H=[1 0];
J=0;
n=sqrt(length(F));
```

```
//Desired poles for the DC Servo system.
Pc=[-2 -2]

// State feedback gain
exec .\acker_dk.sci;
K=acker_dk(F,G,Pc)//Gain computed in book is incorrect.
disp(K,'K=',"State feedback gain")
//------------------------------------------------------------------------------

//Overall transfer function with reduced order estimator.
Gred=8.32*(0.096+s)/(0.1 +s)/(8 + 4*s+s^2)
Gred=syslin('c',Gred)
disp(Gred,'Ys/Rs',"Overall transfer function with reduced...
 order estimator")

//Compensator
D=(0.096+s)*(s+1)/(4.08 +s)/(0.0196+s)
Ds=syslin('c',D*8.32)
disp(Ds,'Ds=',"Compensator transfer function")
//------------------------------------------------------------------------------

//root locus
figure(0)
evans(D*Gs,100) //Correct root locus
zoom_rect([-0.2 -0.1 0.1 0.1])
f=gca();
f.x_location = "origin"
f.y_location = "origin"
xset("color",2);
h=legend('');
h.visible = "off"
//Title, labels and grid to the figure
exec .\fig_settings.sci; // custom script for
```

```
setting figure properties
title('Root locus of lag-lead compensation','fontsize',3);
//-----------------------------------------------------------------------
//Bode plot
figure(1)
bode(Ds*Gs,0.01/2/%pi,100/2/%pi,"rad") //Correct root locus

f=gca();
h=legend('');
h.visible = "off"
//Title, labels and grid to the figure
exec .\fig_settings.sci; //custom script for setting figure properties
title('Frequency response of lag-lead compensation','fontsize',3);
//-----------------------------------------------------------------------
//step response of the system with lag compensation
t=0:0.1:5;
ylag=csim('step',t,8.32*Gs*D/(1+8.32*Gs*D));
figure
plot(t,ylag,2);
xlabel('Time (sec)');
ylabel('y');
title("Step response of the system with lag compensation",'fontsize',3)
exec .\fig_settings.sci; //custom script for setting figure properties
//-----------------------------------------------------------------------
//Discrete-time controller
sysDc_ss=syslin('c',tf2ss(Ds));
```

```
ts=0.1;
sysDd=dscr(sysDc_ss,ts)
Gdz=ss2tf(sysDd)

disp(Gdz,"Discrete-time compensator")
//----------------------------------------------------------------------------------------------------------

//step responses comparision
importXcosDiagram(".\Ex7_34_model.xcos")

xcos_simulate(scs_m,4);
scs_m.props.context
figure,
plot(yt.time,yt.values(:,1),2)
plot(yt.time,yt.values(:,2),'r--')
xlabel('Time (sec)');
ylabel('y');
title("Comaprison of step responses for continuous and discrete...
 controllers",'fontsize',3)
exec .\fig_settings.sci; //custom script for setting figure properties
legend("continuous controller","digital controller",4)

//Control inputs
figure,
plot(ut.time,ut.values(:,1),2)
plot(ut.time,ut.values(:,2),'r--')
xlabel('Time (sec)');
ylabel('u');
title("Comaprison of control signals for continuous and discrete...
 controllers",'fontsize',3)
exec .\fig_settings.sci; //custom script for setting figure properties
legend("continuous controller","digital controller")
```

```
//
  ------------------------------------------------------------------------------
```

This code can be downloaded from the website wwww.scilab.in check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 7.35 Integral Control of a Motor Speed System

```
//Example 7.35
// Integral Control of a Motor Speed System

xdel(winsid())//close all graphics Windows
clear;
clc;
//
  ------------------------------------------------------------------------------


//Transfer function model
num=1;
s=poly(0,'s');
den=(s+3);
G=syslin('c',num/den);
sys=tf2ss(G)

// State space representation of augmented system
F=[0 1; 0 -3];
```

```
G=[0 1]'
H=[1 0];
J=0;

//Desired poles for augmented system
Pc=[-5 -5]

// State feedback gain is
K=ppol(F,G,Pc)
disp(K,'K=')

//Estimator
Pe=[-10]
L=ppol(sys.A',sys.C',Pe)
disp(L','L=')

//
    ------------------------------------------------------------------------------

//(c) Compare step reference and disturbance
    response.
//step reference response switch r=1 and w=0;
r=1;w=0;
importXcosDiagram(".\Ex7_35_model.xcos")
 //The diagram data structure
xcos_simulate(scs_m,4);
scs_m.props.context
figure(0)
plot(yt.time,yt.values)
xlabel('time');
ylabel('y');

figure(1)
plot(ut.time,ut.values)
xlabel('time');
ylabel('y');
//
    ------------------------------------------------------------------------------
```

```
// Step disturbance response switch r=0 and w=1;
w=1;r=0;
importXcosDiagram(”.\Ex7_35_model.xcos”)
 //The diagram data structure
xcos_simulate(scs_m,4);
scs_m.props.context

scf(0)
plot(yt.time,yt.values,’r—’)
xlabel(’time’);
ylabel(’y’);
title(”step Response”,’fontsize’,3)
exec .\fig_settings.sci; // custom script for
   setting figure properties
legend(”y1”,”y2”)
xset(’font size’,3);
xstring(0.9,0.9,”$y_1$”);
xstring(0.25,0.12,”$y_2$”);


scf(1)
plot(ut.time,ut.values,’r—’)
xlabel(’time’);
ylabel(’y’);
title(”Control efforts”,’fontsize’,3)
exec .\fig_settings.sci; // custom script for
   setting figure properties
legend(”u1”,”u2”)
xset(’font size’,3);
xstring(0.25,2.5,”$u_1$”);
xstring(1,-1,”$u_2$”);
//
  ——————————————————————————————————————————————————
```

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

Chapter 8

Digital Control

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 8.1 Digital Controller using tustin approximation

```
///Example 8.1
// Digital Controller using tustin approximation.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
   ---------------------------------------------------------------------------------------------

//Cntroller
s=poly(0,'s');
numD=s/2+1;
denD=s/10+1;
D=10*numD/denD;
Ds=syslin('c',D);
//sampling freq. = 25 times bandwidth
Wbw=10;
```

```
Ws=25*Wbw;
fs=Ws/2/%pi;
T=1/fs; //sampling time
a=1;b=-1;
c=1;d=1;
//Digital controller
z=poly(0,'z');
Dz=horner(Ds,2/T*(a*z+b)/(c*z+d));
disp(Dz,'Digital Controller : ')

//-------------------------------------------------------------------------------------
//step response and control efforts.
figure(0);
importXcosDiagram(".\Ex8_1_model.xcos")
 //The diagram data structure
xcos_simulate(scs_m,4);
scs_m.props.context
plot(yt.time,yt.values(:,1),'r--')
plot(yt.time,yt.values(:,2),2)

xlabel('Time (sec.)');
ylabel('Position, y');
title(["Comparison between digital and continuous controller step...
 response";"with a sample rate 25 times bandwidth";" (a) Position "],...
 'fontsize',3);
exec .\fig_settings.sci; // custom script for setting figure properties

//control effort

figure(1);
plot(ut.time,ut.values(:,1),'r--')
plot2d2(ut.time,ut.values(:,2),2)

```

```
xlabel('Time (sec.)');
ylabel('Control, u');
title(["Comparison between digital and continuous controller step...
 response";"with a sample rate 25 times bandwidth";"(b) Control "],...
 'fontsize',3);
exec .\fig_settings.sci; // custom script for setting figure properties
//---------------------------------------------------------------------------------
```

This code can be downloaded from the website wwww.scilab.in check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 8.2 Design of a Space Station Attitude Digital Controller using Discrete Equivalents

```
//Example 8.2
// Design of a Space Station Attitude Digital Controller using
// Discrete Equivalents

xdel(winsid())//close all graphics Windows
clear;
clc;
```

```
//----------------------------------------------------------------------------------------
// State space representation of continuous time system
s=poly(0,'s');
num=1;
den=(s^2);
Gs=syslin('c',num/den);
Ds=0.81*(s+0.2)/(s+2);
Ds=syslin('c',Ds);
sysc=Gs*Ds;

//Root locus
evans(sysc)
zoom_rect([-2 -0.4 0.5 0.4])
f=gca();
f.x_location = "origin"
f.y_location = "origin"
h=legend('');
h.visible = "off"
exec .\fig_settings.sci; //custom script for setting figure properties
title('s-plane locus with respect to K','fontsize',3)
//----------------------------------------------------------------------------------------
//Contonuous time response of the system
figure,
tc=0:0.1:30;
syscl=sysc/(1+sysc)
yc=csim("step",tc,syscl);
plot(tc,yc,'b')
//----------------------------------------------------------------------------------------
// Discretization of the system at
```

```
z=poly(0,'z')
// sampling time Ts=1 sec
Ts=1;
Dz1=horner(Ds,2/Ts*(z-1)/(z+1))
disp(Dz1,"Dz1=","Discrete-time controller with Ts=1 sec.")

// sampling time Ts=0.5 sec
Ts2=0.5;
Dz2=horner(Ds,2/Ts2*(z-1)/(z+1))
disp(Dz2,"Dz2=","Discrete-time controller with Ts=0.5 sec.")

//discrete-time response of the system.

importXcosDiagram(".\Ex8_2_model.xcos")
 //The diagram data structure
xcos_simulate(scs_m,4);
//scs_m.props.context
plot(yt1.time,yt1.values,'m-.') //with Ts=1sec.
plot(yt2.time,yt2.values,'r--') //with Ts=0.5 sec.
//--------------------------------------------------------------------------

title('step responses of continous and digital implementations','fontsize',3)

exec .\fig_settings.sci; // custom script for setting figure properties
xlabel('Time (sec)','fontsize',2)
ylabel('Plant output','fontsize',2)
legend("Continuous design","Discrete equivalent design, T=1 sec."...
,"Discrete equivalent design, T=0.5 sec.",4)
//--------------------------------------------------------------------------
```

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

Chapter 9

Nonlinear Systems

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 9.5 Changing Overshoot and Saturation nonlinearity

```
//Example 9.5
//Changing Overshoot and Saturation nonlinearity.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
  ---------------------------------------------------------------------------------

//System transfer function and its root locus

s=poly(0,'s');
num=(s+1)
den=(s^2);
Gs=syslin('c',num/den)

//Root locus
```

```
evans(Gs,5)
title(["Root locus of", "$(s+1)/(s^2)$","with saturation removed"],...
'fontsize',3);
f=gca();
f.x_location = "origin"
f.y_location = "origin"
h=legend('');
h.visible = "off"
exec .\fig_settings.sci; //custom script for setting figure properties
//-------------------------------------------------------------------------------------

// Step response
K=1;
i=[2 4 6 8 10 12];
figure(1);
importXcosDiagram(".\Ex9_5_model.xcos")

for r=i
xcos_simulate(scs_m,4);
scs_m.props.context
plot(yt.time,yt.values)
end

xlabel('time');
ylabel('y');
title("Step response of the system for various input sizes",'fontsize',3);
exec .\fig_settings.sci; //custom script for setting figure properties

xset('font size',3);
xstring(4,2.5,"$r=2$");
xstring(6,5.5,"$4$");
xstring(8,8.7,"$6$");
xstring(10,12.2,"$8$");
```

```
xstring(12,15.4,"$10$");
xstring(14,18.4,"$12$");
//------------------------------------------------------------------------------
```

This code can be downloaded from the website wwww.scilab.in check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 9.6 Stability of conditionally stable system using root locus

```
//Example 9.6
//Stability of conditionally stable system using root locus.
xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------------

//System transfer function and its root locus

s=poly(0,'s');
num=(s+1)^2
den=(s^3);
Gs=syslin('c',num/den)
//Root locus
```

```
evans(Gs,7)
title(["Root locus for", "$(s+1)^2/(s^3)$","for system"],...
'fontsize',3);
f=gca();
f.x_location = "origin"
f.y_location = "origin"
h=legend('');
h.visible = "off"
exec .\fig_settings.sci; //custom script for setting figure properties
//-----------------------------------------------------------------------

//Response of the system
K=2;
i=[1 2 3 3.475];
figure(1);

importXcosDiagram(".\Ex9_6_model.xcos")

for r=i
xcos_simulate(scs_m,4);
scs_m.props.context
plot(yt.time,yt.values)
end

xlabel('Time (sec.)');
ylabel('Amplitude');
title("Step response of the system",'fontsize',3);

exec .\fig_settings.sci; //custom script for setting figure properties
xset('font size',3);
xstring(3,6.5,"$r=3.475$");
xstring(2.5,5.2,"$3$");
xstring(2,3,"$2$");
xstring(1,1.4,"$1$");
```

```
//------------------------------------------------------------------------------
```

This code can be downloaded from the website wwww.scilab.in check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 9.7 Analysis and design of the system with limit cycle using the root locus

```
//Example 9.7
//Analysis and design of the system with limit cycle using the root locus.
xdel(winsid())//close all graphics Windows
clear;
clc;
//------------------------------------------------------------------------------
//System transfer function and its root locus

s=poly(0,'s');
num=0.1;
den=(s^2+0.2*s+1)*(s);
Gs=syslin('c',num/den);

//Root locus
evans(Gs,40)
```

```
title(["Root locus of", "$(0.1/s(s^2+0.2*s+1)$"],'fontsize',3);
f=gca();
f.x_location = "origin"
f.y_location = "origin"
h=legend('');
h.visible = "off"
exec .\fig_settings.sci; // custom script for setting figure properties
//------------------------------------------------------------------------------

//Response of the system
figure;
//Response of the system
K=0.5;
i=[1 4 8];
importXcosDiagram(".\Ex9_7_model.xcos")

for r=i
xcos_simulate(scs_m,4);
scs_m.props.context
plot(yt.time,yt.values)
end

xlabel('Time (sec.)');
ylabel('Amplitude');
title("Step response of the system",'fontsize',3);
exec .\fig_settings.sci; // custom script for setting figure properties
zoom_rect([0 0 150 9])

xset('font size',3);
xstring(80,1.6,"$r=1$");
xstring(80,4.6,"$r=4$");
xstring(80,8.2,"$r=8$");
//------------------------------------------------------------------------------
```

```
//System with notch compensation
D=123*(s^2+0.18*s+0.81)/(s+10)^2;

//Root locus
figure,
evans(Gs*D,40)
title(["Root locus including notch compensation"],'fontsize',3);
f=gca();
f.x_location = "origin"
f.y_location = "origin"
h=legend('');
h.visible = "off"
exec .\fig_settings.sci; //custom script for setting figure properties
zoom_rect([-14 -2 2 2])
//
-----------------------------------------------------------------------

//Response of the system witth notch filter
figure;
K=0.5;
i=[2 4];
importXcosDiagram(".\Ex9_7_model_notch.xcos")

for r=i
xcos_simulate(scs_m,4);
scs_m.props.context
plot(yt.time,yt.values)
end

xlabel('Time (sec.)');
ylabel('Amplitude');
title("Step response of the system with notch filter",'fontsize',3);
exec .\fig_settings.sci; //custom script for setting figure properties
```

```
xset('font size',3);
xstring(30,2.2,"$r=2$");
xstring(34,3.75,"$r=4$");
//
    ---------------------------------------------------------------------------
```

This code can be downloaded from the website wwww.scilab.in This code can be downloaded from the website wwww.scilab.in check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 9.8 Antiwindup compensation for a PI controller

```
//Example 9.8
//Antiwindup compensation for a PI controller.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
    ---------------------------------------------------------------------------

//System Model

//Response of the system
kp=2;
```

```
ki=4;

//Without antiwindup
ka=0;
importXcosDiagram(".\Ex9_8_model.xcos")
xcos_simulate(scs_m,4);
scs_m.props.context
figure(0)
plot(yt.time,yt.values,'m-.')
figure(1)
plot(ut.time,ut.values,'m-.')

//With antiwindup
ka=10;
xcos_simulate(scs_m,4);
scf(0)
plot(yt.time,yt.values)
exec .\fig_settings.sci; // custom script for setting figure properties
xlabel('Time (sec.)');
ylabel('Output');
title("Integrator antiwindup (a) step response.",'fontsize',3);


scf(1)
plot(ut.time,ut.values);
exec .\fig_settings.sci; // custom script for setting figure properties
xlabel('Time (sec.)');
ylabel('Control');
title("Integrator antiwindup (b) Control effort.",'fontsize',3);
zoom_rect([0 -1.2 10 1.2])

//
-----------------------------------------------------------------------
```

This code can be downloaded from the website wwww.scilab.in check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 9.9 Describing Function for a saturation nonlinearity

```
//Example 9.9
//Describing Function for a saturation nonlinearity.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
    -----------------------------------------------------------------------
//Response of the saturation nonlinearity to sinusoidal input
figure;
importXcosDiagram(".\Ex9_9_model.xcos")
xcos_simulate(scs_m,4);
scs_m.props.context
plot(yt.time,yt.values(:,1),'r--')
plot(yt.time,yt.values(:,2),'b')

xlabel('Time (sec.)');
ylabel('Amplitude');
title("Saturation nonlinearity output to sinusoidal
    input",...
```

```
'fontsize',3);
exec .\fig_settings.sci; //custom script for setting figure properties
//------------------------------------------------------------------------------------------------
//Describing Functin for saturation nonlinearity.
k=1;
N=1;
i=1;
Keq=[];

for a=0:0.2:10
    if k*a/N > 1 then
    Keq(i,1)=2/%pi*(k*asin(N/a/k)+N/a*sqrt(1-(N/k/a)^2))
    else
    Keq(i,1)=k
    end
    i=i+1;
end

a=0:0.2:10;
a=a';
figure,
plot(a,Keq)
xlabel('$a$');
ylabel('$K_{eq}}$');

xset('font size',3);
title("Describing Function for a saturation nonlinearity...
 with k=N=1",'fontsize',3);
exec .\fig_settings.sci; //custom script for setting figure properties
zoom_rect([0 0 10 1.1])
//------------------------------------------------------------------------------------------------
```

This code can be downloaded from the website wwww.scilab.in check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 9.11 Describing Function for a relay with hysteresis non linearity

```
//Example 9.11
//Describing Function for a relay with hysteresis
   nonlinearity.

xdel(winsid())//close all graphics Windows
clear;
clc;
//
  --------------------------------------------------------------------------------

//Response of the saturation noninearity to
   sinusoidal input
figure;
importXcosDiagram(".\Ex9_11_model.xcos")
xcos_simulate(scs_m,4);
scs_m.props.context
plot(yt.time,yt.values(:,1),'r--')
plot(yt.time,yt.values(:,2),'b')

xlabel('Time (sec.)');
```

```
ylabel('Amplitude');
title("Relay with hysteresis nonlinearity output to sinusoidal...
 input",'fontsize',3);
exec .\fig_settings.sci; //custom script for setting figure properties
zoom_rect([0 -1.2 5 1.2])
//----------------------------------------------------------------------------
////Describing Functin for relay with hysteresis nonlinearity.
h=0.1;
N=1;
i=1;

for a=0.1:0.025:1
    if a<h then
        Keq(i,1)=0;
        ro(i,1)=0;
        theta(i,1)=0
    else
        Keq(i,1)=4*N/(%pi*a)*(sqrt(1-(h/a)^2)-%i*h/a)
        [r th]=polar(Keq(i,1));
        ro(i,1)=r; //magnitude
        theta(i,1)=clean(th); //angle in radians
    end
    i=i+1;
end

a=0.1:0.025:1
a=a';
figure,

subplot(2,1,1), plot(a,ro)
xlabel('$a$');
ylabel(['Magnitude', '$|K_{eq}|$']);
```

```

xset('font size',3);
exec .\fig_settings.sci; //custom script for setting figure properties
title("Describing Functin for relay with hysteresis nonlinearity...
 with h=0.1 and N=1",'fontsize',3);

subplot(2,1,2), plot(a,theta*180/%pi)
xlabel('$a$');
ylabel(['Phase', '$ \angle K_{eq}$','deg.']);
xset('font size',3);
exec .\fig_settings.sci; //custom script for setting figure properties
//--------------------------------------------------------------------------
```

This code can be downloaded from the website wwww.scilab.in check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 9.12 Conditionally stable system

```
//Example 9.12
//Conditionally stable system.
xdel(winsid())//close all graphics Windows
clear;
clc;
```

```
//------------------------------------------------------------------------------------------
//System transfer function and its root locus

s=poly(0,'s');
num=0.1;
den=(s^2+0.2*s+1)*(s);
Gs=syslin('c',num/den)

//Nyquist plot of the system
nyquist(Gs,0.035,10)
title("Nyquist plot and describing function to determine limit...
 cycle",'fontsize',3);

f=gca();
f.x_location = "origin"
f.y_location = "origin"
h=legend('');
h.visible = "off"
xset("color",2);

// Nyquist Plot of Describing Function for saturation nonlinearity.
omegat=0.05:0.05:%pi;
a=sin(omegat);
N=0.1;
k=1;

Keq=2/%pi*(k*asin(N ./a /k)+N ./a .* sqrt(1-(N/k ./a) .^2));
DF_nyq=-1 ./Keq;

plot(DF_nyq,zeros(1,length(DF_nyq)),'m-.')
exec .\fig_settings.sci; //custom script for setting figure properties
zoom_rect([-0.8 -0.5 0.2 0.5])
```

```

//limit cycle points
plot(-0.5,0,'bo');

xset('font size',3)
xstring(-0.78,0.08,"limit cycle point");
xarrows([-0.6;-0.52],[0.1;0.02],-1)
xstring(-0.62,-0.22,"$-\frac{1}{K_{eq}$");
xarrows([-0.55;-0.55],[-0.1;0],-1)
//------------------------------------------------------------------------

//Describing Functin for saturation nonlinearity.
Keq=[]
i=1;

for a=0:0.2:10
    if k*a/N > 1 then
    Keq(i,1)=2/%pi*(k*asin(N/a/k)+N/a*sqrt(1-(N/k/a)^2))
    else
    Keq(i,1)=k
    end
    i=i+1;
end

a=0:0.2:10;
a=a';

figure,
plot(a,Keq)
xlabel('$a$');
ylabel('$K_{eq}$');

xset('font size',3);
title("Describing Function for a saturation nonlinearity...
 with N=0.1 and k=1",'fontsize',3);
```

```
exec .\fig_settings.sci; //custom script for setting figure properties
zoom_rect([0 0 10 1.1])
//----------------------------------------------------------------------------
```

---

check Appendix AP 1 for dependency:

`fig_settings.sci`

Scilab code Exa 9.13 Determination of stability with a hysteresis nonlinearity

```
//Example 9.13
//Determination of stability with a hysteresis nonlinearity.

xdel(winsid())//close all graphics Windows
clear;
clc;
//----------------------------------------------------------------------------

//System Model
s=poly(0,'s');
num=1;
den=(s^2+s);
Gs=syslin('c',num/den);
//----------------------------------------------------------------------------
```

```
//Nyquist Plot of the system
nyquist(Gs,0.25,3)

// Nyquist Plot of Describing Function for hysteresis nonlinearity
N=1;
h=0.1;
i=1;

for omegat=0:0.05:%pi-0.1;
    a=sin(omegat);
    DF_nyq(i,1)=-%pi/4/N*(sqrt(a^2-h^2) + h * %i)
    i=i+1;
end

plot(real(DF_nyq),imag(DF_nyq),'m-.')
exec .\fig_settings.sci; // custom script for setting figure properties
zoom_rect([-0.3  -0.3 0 0.3])
title('Nyquist plot of system and describing function to...
 determine limit cycle','fontsize',3)

//limit cycle points
plot(-0.1714,-0.0785,'ro');
xstring(-0.25,0,"limit cycle point");
xarrows([-0.2;-0.172],[0;-0.077],-1);

//-----------------------------------------------------------------------------------

//Response of the system
K=2;
r=1
figure(1);
importXcosDiagram(".\Ex9_13_model.xcos")
xcos_simulate(scs_m,4);
scs_m.props.context
```

```
plot(yt.time,yt.values)

xlabel('Time (sec.)');
ylabel('Output, y');
title("Step response displaying limit cycle oscillations",'fontsize',3);
exec .\fig_settings.sci; //custom script for setting figure properties
//-------------------------------------------------------------------------------------------
```

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

Appendix

Scilab code AP 1 figure setting file

```
//
  ---------------------------------------------------------------------------

//figure handel settings
f=get(”current_figure”); //Current figure handle
f.background=8; //make the figure window background
   white
l=f.children(1);
l.background=8 ;//make the text background white
id=color(’grey’);
xgrid(id);
//
  ---------------------------------------------------------------------------
```

Scilab code AP 2 State feedback gain matrix computation

```
//
  ---------------------------------------------------------------------------

//
  ---------------------------------------------------------------------------

//A function written by Deepti Khimani.
//Usage:-
//[K, lambda]=acker_dk(a, b, pl)
```

```
//K=acker_dk(a, b, pl)
//a:- System matrix.
//b:- input matrix.
//p:- Desired poles.
//K:-State feedback gain for the control law u=-Kx.
//lambda:- Eigen values of (a-b*k)
//
    ------------------------------------------------------------------------

//
    ------------------------------------------------------------------------


function [K, lambda]=acker_dk(a, b, pl)
        [lhs,rhs]=argn(0)

    if rhs == 0 then
        disp(["K=acker_dk(a, b, pl)";"[K, lambda]=acker_dk(a, b, pl)"]);
        disp(["a:- System matrix";"b:- input matrix";"p:- Desired poles"]);
        disp(["K:-State feedback gain for the control law u=-Kx";...
 "lambda:- Eigen values of (a-b*k)"]);
        return;
    end
[ra ca]=size(a);
[rb cb]=size(b);
l=length(pl);

CO=cont_mat(a,b);

if ra~=l then
    error(["Dimension error:";"number of desired poles must equal...
 to order of the system"]);
elseif ra~=ca then
    error(["Dimension error:";"system matrix should
```

```
        be...
 a sqaure matrix"]);
elseif rb~=ra then
    error (["Dimension error:","Input matrix should
        have...
 as many rows as a system matrix."]);
elseif rank(CO)<ra then
    error("system is not controllable");
end
//
  ------------------------------------------------------------------

//controllable canonical form
[Ac,Bc,T,ind]=canon(a,b);

//CO=zeros(ra,cb);
for i=1:ra
    CO(:,ra+1-i)=Ac^(i-1)*Bc;
end
//
  ------------------------------------------------------------------

chr_eq=poly(pl,'s');
des_chr_coeff=coeff(chr_eq);

des_chr_coeff=des_chr_coeff(1:ra);
alpha_c=Ac^ra;

for k=1:ra
    alpha_c=alpha_c + des_chr_coeff(k)*Ac^(k-1)
end
//
  ------------------------------------------------------------------

//State feedback gain
temp=zeros(1,ra);
temp(1)=1;
K=temp*inv(CO)*alpha_c;
```

```
K=K/T;
lambda=spec(a-b*K);
endfunction
//----------------------------------------------------------------------
```

Scilab code AP 3 ZPK computation

```
//----------------------------------------------------------------------
//----------------------------------------------------------------------
//A function written by Deepti Khimani.
//Usage:-
//p=zpk_dk(sl)
//[p, z]=zpk_dk(sl)
//[p, z, k]=zpk_dk(sl)
//p:- Poles of the system
//z:- zeros of the system
//k:- DC gain of the system
//----------------------------------------------------------------------
//----------------------------------------------------------------------

function[pl,zr,k]=zpk_dk(sysmodel)
    [lhs,rhs]=argn(0)

    if rhs == 0 then
     disp(["p=zpk_dk(sl)";"[p, z]=zpk_dk(sl)";"[p, z, k]=zpk_dk(sl)"]);
     disp(["p:- Poles of the system";"z:- zeros of the system"]);
```

```
     disp("k:- DC gain of the system");
     return;
    end

    if typeof(sysmodel)=="rational" then
        sys=tf2ss(sysmodel);
        pl=spec(sys.A);
        zr=trzeros(sys);
        temp1=poly(zr,'s','roots')/poly(pl,'s','roots');
        temp2=sysmodel/temp1;
        temp3=tf2ss(temp2);
        k=temp3.D;
    elseif typeof(sysmodel)=="state-space" then
        pl=spec(sysmodel.A);
        zr=trzeros(sysmodel);
        g=ss2tf(sysmodel);
        temp1=poly(zr,'s','roots')/poly(pl,'s','roots');
        temp2=g/temp1;
        temp3=tf2ss(temp2);
        k=temp3.D
    else
        error("Wrong type of input argument.")
    end
endfunction
```