Control System with Scilab

Table of Contents

Phase Portrait

first you need to define a function \dot{x}=f(t,x), x is a vector with two elements. The following code gives two-dimensional phase portrait.

```
function [y]=field(t,x)
y(1)=x(2);
y(2)=-2*sin(x(1));
endfunction
portrait(field)
```

This plots interactively phase portraits for different initial conditions. You can also specify the initial conditions and other parameters. For example:

```
portrait(field, 'default',[-3,-3,3,3],[100,0.1],[1.5 1.5])
```

where second variable is the integration method, third variable is the graphic frame boundary=[xmin,ymin,xmax,ymax], third variable gives number of points and step size, last variable is the initial condition matrix=[x1(0),x1(1),...,x1(n) x2(0),x2(1),...,x2(n)]. I will suggest using the former one.

[TOC]

Root Locus

Express your characteristic equation as "1+K*L(s)=0". where L(s)=N(s)/D(s). then following piece of code plots root locus

```
s=poly(0,'s');
L=1/(s*(s+1)*(s+4));
evans(syslin('c',L),500);
xgrid(color('gray'))
```

where 'syslin' is a data structure representing a LINEAR SYSTEM. The default maximum gain is 1000. Don't get disheartened if it does not look similar to that produced by matlab. Just play around with gain (start from smaller value) until you get one that looks agreeable.
L(s) = G(s)H(s) is the loop transfer function of the system.
There is a very useful toolbox for SISO system analysis called RLtool developed by Ishan Pendharkar.

How to remove legend from root locus plot

It had been annoying to have legend on root locus plot. This is how you can remove legends. First plot root locus using "evans"

```
--> evans(gs,100)
--> f=gcf() //handle to current figure

Handle of type "Figure" with properties:
========================================
children: ["Axes";"Axes"]
figure_style = "new"
figure_position = [403,6]
figure_size = [610,263]
axes_size = [599,199]
auto_resize = "on"
figure_name = "Scilab Graphic (%d)"
figure_id = 0
color_map= matrix 32x3
```

```
[More (y or n ) ?]
pixmap = "off"
pixel_drawing_mode = "copy"
immediate_drawing = "on"
background = -2

visible = "on"
rotation_style = "unary"


--> a=f.children //handle to children

2 by 1 matrix of handles:
=========================
Axes
Axes
```

One of these two axes represent legends. so try out following command to remove legend.

```
--> a(1).visible="off"
```

Sometime, you would also wish to remove the asymptotes which appear as bold lines on the graph. To do this, one has to try out various options as mentioned above, but I hope following command should work most of the time.

-
  ```
  -->a(2).children
   ans =

  6 by 1 matrix of handles:
  =========================
  Agregation
  Segs
  Segs
  Agregation
  Agregation
  Agregation
  ```

Here the first aggregation refers to various root locus segments. second and third "segs" refer to asymptotes. So in order to remove asymptotes

-
  ```
  --> a(2).children(2).visible="off"
  --> a(2).children(3).visible="off"
  ```
  - You can modify its properties in a similar manner.

    In order to make the symbols of poles and zeros bigger, check other three aggregations. Check for the one that shows `mark_mode="on"`. for instance, for my figure, i get following output

-
  ```
  -->a(2).children(5).children
   ans =

  Handle of type "Polyline" with properties:
  ==========================================
  parent: Agregation
  children: []
  visible = "on"
  data = [0,0;-2,3;-2,-3]
  line_mode = "off"
  line_style = 0
  thickness = 1
  polyline_style = 1
  foreground = -1
  mark_mode = "on"
  mark_style = 2
  mark_size_unit = "point"
  mark_size = 10
  mark_foreground = -1
  mark_background = -2
  clip_state = "clipgrf"
  clip_box = []
  ```

  Hence this particular handle refers to symbols used for representing poles and zeros. Now, in order to change the size of these symbols, use following command

-
  ```
  -->a(2).children(5).children.mark_size=10;
  -->a(2).children(6).children.mark_size=10;
  ```
  - in my case,
    `a(2).children(5)` and
    `a(2).children(6)` are handles to poles and zeros respectively. You may

change other attributes through similar procedure.

Root Locus in Scilab Version 5.2.2

```
s = %s;
p = (s+2)/(s*(s+1)*(s+5)*(s+10));
g = syslin('c',p);

f2 = scf(1);
evans(g,1000); //root locus

xgrid(color('gray'));

a = f2.children.children;
a(1).visible = "off"; //disable legend
b = a(2).children;
for i = 1:length(b),
  b(i).thickness = 3; //increase the thickness of root loci
end
a(7).mark_size=9; //increase the size of marks
a(8).mark_size=9;
```

Displaying the gain on any point on root locus (Scilab Version 5.2.2/5.3.1)

```
//Root Locus Function
//Successfully tested on
// 05 September 2010, Sunday
//-----------------------------------
function f = rlocus(g,K)
  if(typeof(g) ~= "rational")
    disp("Wrong Input");
    abort;
  else
    //Root Locus
    f=scf();
    evans(g,K);
    xgrid(color('gray'));
    a = f.children.children;
    a(1).visible = "off";
    b = a(2).children;
    for i = 1:length(b),
      b(i).thickness = 3;
    end
    a($).mark_size=10;
    a($-1).mark_size=10;
    save('TMPDIR/rlocus_var',g);
    gwin = get(f,'figure_id');
    addmenu(gwin, "Time Response", list(2,'rlocusgain'));
  end
endfunction
//-----------------------------------------------
function r = rlocusgain(f,K)
    load('TMPDIR/rlocus_var','g');
  print(%io(2),"Click on the Root Locus");
  show_window(f);
  scf(f);
    [b,xc,yc] = xclick(); // get a point
    if(b == 3) then
      r = xc + yc*%i; //root location
      rep = horner(g,r);
      //[mag,phi] = polar(rep); //mag in absolute value
      mag = abs(rep); // for check
      phi = atan(imag(rep),real(rep)); //for check
      angle = real(phi)*180/%pi; //in degree
      if (((abs(angle) >= 175) & (abs(angle) < 185)) |
(abs(angle) < 5)) then
        xset("mark",1,2);
        xpoly(xc,yc,"marks");

        //magnitude
        Kg = 1/mag;

        //natural frequency
        wn = abs(r);

        //damping coefficient
        zeta = abs(xc)/wn;

        //Peak Overshoot in percent
        Mp = exp(-%pi*zeta/sqrt(1-zeta^2))*100;

        //Settling time for 1% band
        ts = 4/(zeta*wn);

```

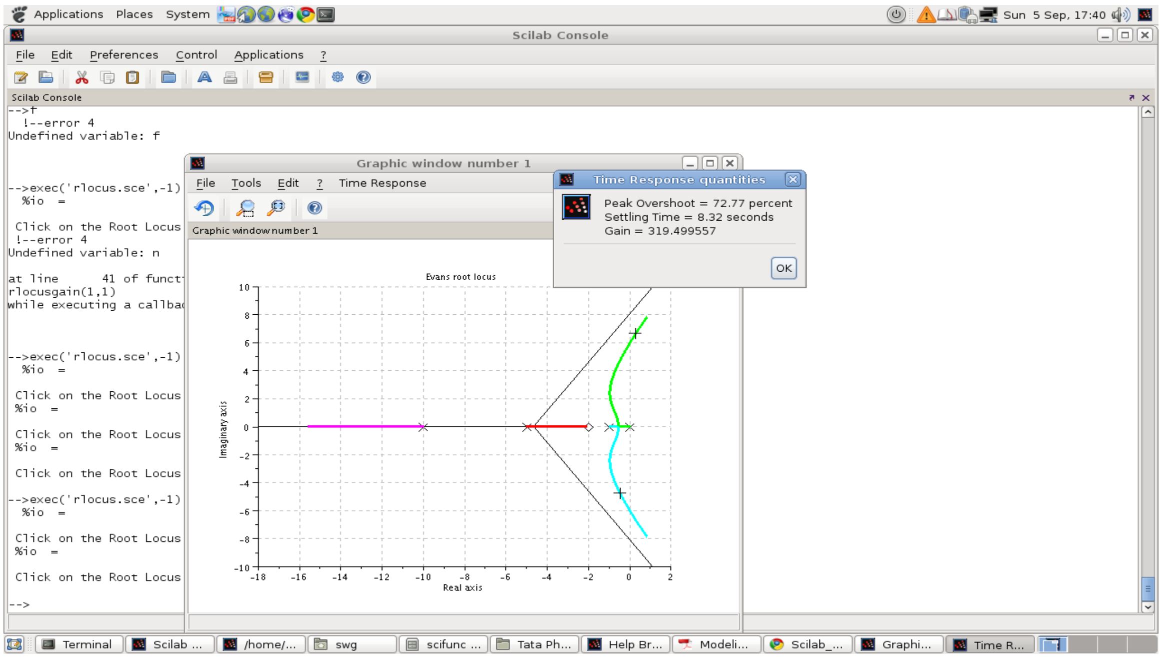
```
•         peak_overshoot = sprintf("Peak Overshoot = %6.2f
percent",Mp);
•         settling_time = sprintf("Settling Time = %6.2f
seconds",ts);
•         Gain = sprintf("Gain = %6.2f", Kg);
•         damp = sprintf("Damping = %6.2f", zeta);
•         natfreq = sprintf("Natural Frequency = %6.2f", wn);
•         dompole = sprintf("Dominant Pole = %6.2f + %6.2f i",
xc, yc);
•         Angle = sprintf("Angle = %6.2f", angle);
•
messagebox([peak_overshoot,settling_time,Gain,damp,natfreq,d
ompole,Angle], "Time Response quantities");
•       end
•     end
•     n = 0;
• endfunction
```

Save the above program as “rlocus.sci” on your system. This is how you will call the function in your program:

```
exec(“Pathtofile/rlocus.sci”,-1);
l = (%s+1)/(%s*(%s^2+9));
rlocus(l,100);
```

Now click on “Time Response” tab on the rlocus window.

Following snapshots shows above routine in action



```
-->f
  !--error 4
Undefined variable: f


-->exec('rlocus.sce',-1)
  %io  =

 Click on the Root Locus
 !--error 4
Undefined variable: n

at line      41 of funct
rlocusgain(1,1)
while executing a callba


-->exec('rlocus.sce',-1)
  %io  =

 Click on the Root Locus
 %io  =

 Click on the Root Locus
 %io  =

 Click on the Root Locus

-->exec('rlocus.sce',-1)
  %io  =

 Click on the Root Locus
 %io  =

 Click on the Root Locus

-->
```

Time Response quantities

Peak Overshoot = 72.77 percent
Settling Time = 8.32 seconds
Gain = 319.499557

[TOC]

Linear Systems

A linear system is defined as

```
s1=syslin('c', A,B,C,D); // continuous time
s2=sysline('d',F,G,H,K); //discrete time
ssprint(s1); //print in State-space format
```

Any matrix of the state-space form can be extracted using dot operator, for instance:

```
a1 = s1.A;
[a,b,c,d] = abcd(s1);
```

State-space model may be converted into transfer function as shown below. The same command is also applicable for discrete-time systems as well. It is also possible to convert transfer function model into state space model.

```
tf1=ss2tf(s1);
s11=tf2ss(tf1);
```

A linear system can also defined directly as a transfer function in Laplace domain as follows:

```
s=%s;
num = 1+s;
den = 1+5*s+s^2;
P = syslin('c',num,den);
zeroes = roots(P.num);
poles = roots(P.den);
```

Pole Placement
For continuous time pole placement, scilab has a command called 'ppol'. This is demonstrated below:

```
A=[0 1;0 0]; B=[0;1];
K=ppol(A,B,[-1,-2]);
spec(A-B*K)
```

Time response of a linear system

State Space form
This program finds out step response of a linear system

-
  ```
  t=0:0.01:4;
  u=ones(1:length(t));
  A=[-2 1;-1 -3];
  B=[1;-2];
  C=[1 0];
  s1=syslin('c',A,B,C); //linear system definition
  [y,x]=csim(u,t,s1); //time response
  clf();
  plot2d(t,y,style=[2]);
  xgrid(color('lightgray'));
  xtitle(['Step Response of a Linear
  System'],'Time', 'Amplitude'     );
  ```
  -

Transfer function form
Following code finds impulse response for a transfer function

-
  ```
  t=0:0.05:10;
  s=poly(0,'s');
  p=syslin('c', 10/(s^2+0.4*s+4));
  y=csim('impuls',t,p);
  clf();
  plot2d(t,y,style=[2]);
  ```
  -

    you can use "step" instead of "impuls" in order to get step response.

    Feedback Connection
    In order to find out closed loop transfer function with unity feedback, i.e, `C(s) = G(s) / (1+G(s))`Use following steps

-
  ```
  s=poly(0,'s');
  gs=syslin('c',1/(s+1));
  C=gs/. 1
  ```
  -

    Note that '/.' is a feedback connection operator. For a feedback connection of two tranfer functions gs and cs, replace 1 by cs.

    Poles and Zeros
    Let 'Gf' be a transfer function defined with syslin command.

-
  ```
  trfmod(Gf) // returns poles and zeros of Gf
  roots(numer(Gf)) // returns zeros
  roots(denom(Gf)) //returns poles
  plzr(Gf) // gives the pole-zero map
  ```
  -

-- 23 Aug 2005 0408 IST

Step Response

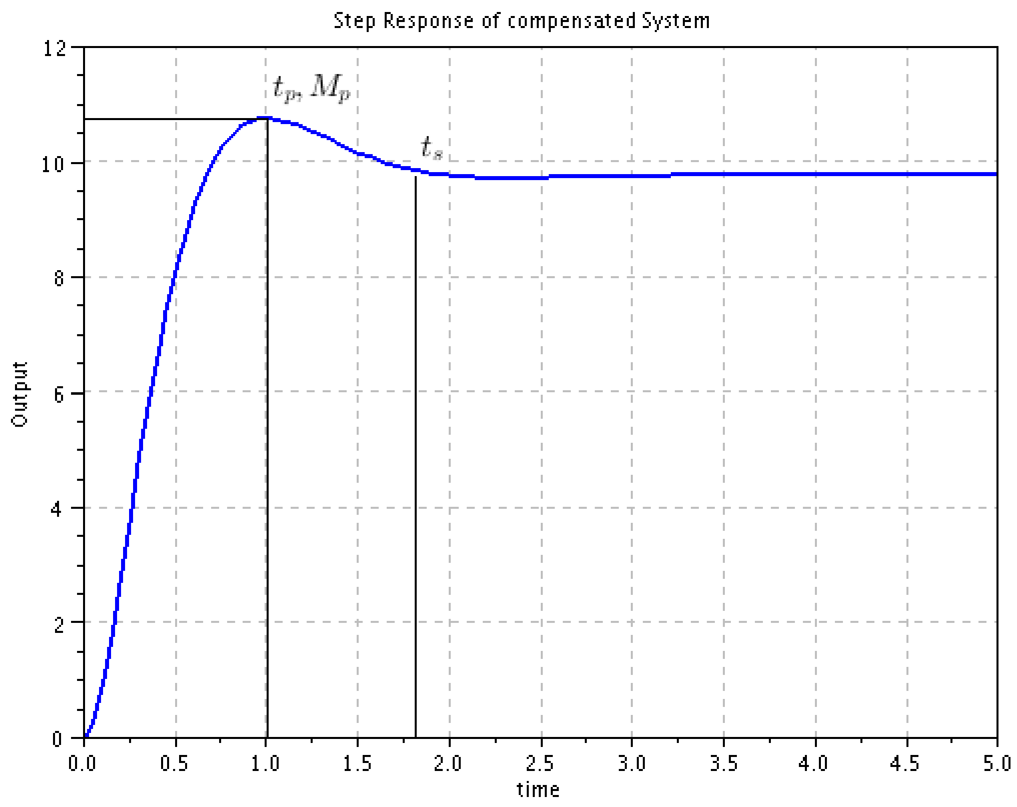
Follow code shows how to compute the time response quantities from the step response plot itself. It shows the values in a pop-up message box.

```
//step.sce
//Scilab Version 5.2.2
//-------------------------------
function h = step(g, t, k)
  //Plot the step response of linear system
  if ~exists('splitvec') then
      exec('/home/swg/PROG/scilab/scifunc/splitvec.sci');
  end
```

```
//get the actual no. of input and output
[lhs,rhs]= argn();

if rhs == 1 then
    t = 0:0.01:10;
    k = 1;
elseif rhs == 2 then
    k = 1;
else
    error("Wrong number of arguments");
end
u = k*ones(t);

y = csim(u,t,g);

h = scf();
plot(t,y,'LineWidth',2);
xgrid(color('gray'));
xtitle('Step Response');
xlabel('time');
ylabel('Output');

//Time-response characteristics
Y=[t,y];
[Y_max,idx_ymax] = max(y);

//Peak Time
tp = t(idx_ymax);

temp_y = y($-10:$);
ty_max = max(temp_y);
ty_min = min(temp_y);

//Steady state value of output
if abs(ty_max - ty_min) < 0.01 then
    ys = y($);
end

//peak overshoot (in %)
Mp = (Y_max - ys)/ys * 100;

//steady-state error (in %)
Ess = (k - ys)/k * 100;

//Settling Time with 1% band
id = find(y >= (ys-0.01*ys) & y < (ys+0.01*ys));
[vc,vl]=splitvec(id);

//findt  the vector with largest length
max_len_idx = 1;
max_len = 0;
for i = 1:length(vl)
```

```
      if length(vl(i)) > max_len then
        max_len = length(vl(i));
        max_len_idx = i;
      end
  end
  //first element of the longest time vector
  ts = t(vl(max_len_idx)(1));

  show_window(h);
  xpoly([tp,tp],[Y_max,0]);
  xpoly([0,tp],[Y_max,Y_max]);
  //xstring(tp,Y_max+0.1,"$t_p,M_p$");
  //xset("color",3);
  xpoly([ts,ts],[0,ys]);
  //xstring(ts,ys+0.1,"$t_s$");
  //xset("color",1);

  peak_overshoot = sprintf("Peak Overshoot = %6.2f percent",Mp);
  peak_time = sprintf("Peak Time = %6.2f seconds",tp);
  settling_time = sprintf("Settling Time = %6.2f seconds",ts);
  ss_error = sprintf("Steady State Error = %6.2f percent", Ess);

  messagebox([peak_overshoot,peak_time,settling_time,ss_error], "Time
response Quantities");
endfunction
```

```
// SCILAB VERSION 5.2.2
//------------------------------

s = %s;

//open-loop transfer function
g = syslin('c',2/((s+2)*(s+10)));

//Compensator is
gc = 12.3*(s+3.3)/(s+0.1);

//Closed-loop transfer function
H = g*gc/. 1;

//Draw Step response:
t = 0:0.01:5.0;
step(H,t,10);
```



The function **splitvec** is defined as follows: (Scilab Version 5.2.2)

```
function [vec_cnt, vec_list] = splitvec(a)
  // Splits the vector into several vectors having consecutive elements
  // each vector should have at least 1 elements
  vec_list = list(); //Create an empty list.
  vec_cnt = 0;
  vec_size = 1;
  v = []; // Empty array
  for i = 1:length(a),
          v(vec_size) = a(i);
          if (i < length(a)) & (a(i+1) == a(i)+1) then
            vec_size = vec_size+1;
          else
            vec_cnt = vec_cnt + 1;
            vec_list($+1) = v;
            vec_size = 1;
            v = [];
          end
  end
```

```
endfunction
```

Download the scripts “step.sci” and “splitvec.sci” and put it into a folder. Now execute the following commands from the scilab command window to get it working as shown below:

```
→ exec(“~/scifunc/step.sci”,-1);
→  g = syslin(‘c’, 10/(%s+1));
→  h = g /. 1;
→ step(h);
```

[TOC]

Frequency Response Analysis of a linear system

Bode Plot

In scilab, the function bode(), the magnitude and phase values are plotted against frequency in Hertz. That's why it looks different from that obtained in Matlab. Following code gives bode plot where the frequency is in rad/sec.

```
clear all;
clc;
xbasc();

s = %s/(2*%pi);

g = syslin('c',(s-1)/(s*(s+1)*(s+2)));


//Draw bode plot
bode(g, 0.01, 100);

//--------------------------------
// Changing the label of bode plot
a = gcf();
a.figure_name = "Bode Plot";
a1 = a.children(1); // phase subplot
a2 = a.children(2); // magnitude subplot
a1.x_label.text = "Frequency (rad/sec)";
a2.x_label.text = "Frequency (rad/sec)";
//--------------------------------------------
```

Gain Plot

It is possible to plot gain plot separately for different linear systems:

```
clear;
mtlb_close all;
s=%s/(2*%pi);

f=scf(0);

g = syslin('c',1/(s*(s+1.414)));
g1 = syslin('c',(s+1)/(s*(s+1.414)));
gainplot([g;g1],0.01,10^5,['g';'g1']);
f.children.children(2).children(1).thickness=2;
f.children.children(2).children(2).thickness=2;
f.children.x_label.text="Frequency (rad/s)";
```

Phase Margin and Gain Margin

In scilab, 'p_margin' command returns the phase of the transfer function at gain cross-over frequency.

```
s = %s/(2*%pi);
g = syslin('c',(s-1)/(s*(s+1)*(s+2)));

// Phase margin
[phm, fg] = p_margin(g); // fg in rad/sec

if(phm >= 0) then PM = phm - 180;
else then PM = phm + 180;
end

// Gain Margin
[gm, fp] = g_margin(g); // fp in rad/sec
```

'fg' is the gain cross over frequency in rad/sec if s = %s/(2*%pi) otherwise it is in hertz. Similarly, 'fp' is the phase crossover frequency.

-- 03 Aug 2007 Friday

Phase and Magnitude of a transfer function

```
s=poly(0,'s');
p1=10/(s*(s^2+0.4*s+4));
rep=freq(p1("num"),p1("den"),[0.1*%i]);
phi=atan(imag(rep),real(rep));
mag=abs(rep);
```

freq computes the complex value of the transfer function at s = 0.1i . Hence, above code gives phase (in radian) and absolute magnitude at a frequency w = 0.1 rad/sec.

Another method

```
p1=10/(%s*(%s^2+0.4*%s+4))
rep = horner(p1, %i * 0.1)
[A, ph] = polar(rep); // A is the absolute magnitude (NOT in dB)

phase = abs(ph)*180/%pi ; // phase in degrees
```

'horner' evaluates the value of the polynomial p1(s) at s = 0.1i. This code also evaluates the phase and magnitude of the transfer function at w = 0.1 rad/sec. The answer can be verified against matlab's bode command as follows

```
s = tf('s');
p1 = 10/(s*(s^2+0.4*s+4));
[mag, ph] = bode(p1, 0.1);
```

Note that in scilab 3.0, %s is a built-in that evaluates to the polynomial s (%z is the same thing in z).

-- 03 Aug 2007 Friday

Interactive Bode

Following Scilab function enables a user to obtain a gain-freq or phase-freq pair directly on the bode plot when a user clicks at a particular point on the graph:

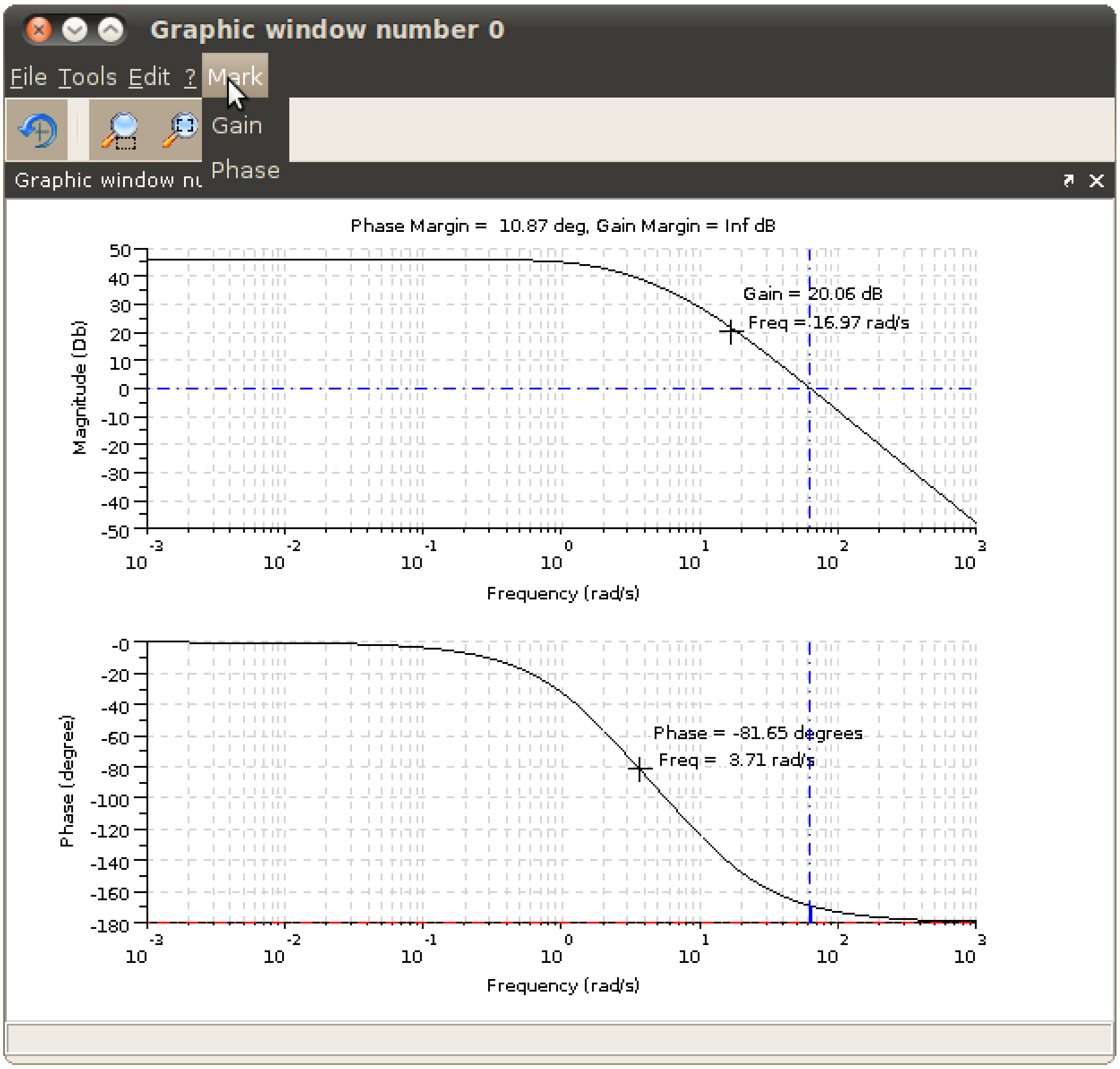
```
//margins.sci (Scilab Version 5.2.2)
function n = markval(k,f)
  show_window(f);
  f1 = gcf();

  if (k == 1) then

      a1 = sca(f1.children(3));
      //printf("Select a point on Gain Plot\n");
      [b,xc,yc] = xclick();
      str = sprintf('Gain = %5.2f dB\n Freq = %5.2f rad/s', yc,xc);
      xset("mark",1,2);
      xpoly(xc,yc,"marks");
      xstring(xc,yc,str);
  elseif (k == 2) then
```

```
      a1 = sca(f1.children(2));
      //printf("Select a point on the Phase Plot\n");
      [b,xc,yc] = xclick();
      str = sprintf('Phase = %5.2f degrees\n Freq = %5.2f rad/s', yc,xc);
      xset("mark",1,2);
      xpoly(xc,yc,"marks");
      xstring(xc,yc,str);
  end
  n = 0; //return 0 on success
 endfunction

 //-------------------------------
function f = margins(g)
  f = scf();
  show_margins(g,'bode');
  f.children(1).x_label.text="Frequency (rad/s)";
  f.children(2).x_label.text="Frequency (rad/s)";
  [phm,wg] = p_margin(g);
  [gm,wp] = g_margin(g);
  str = sprintf("Phase Margin = %6.2f deg, Gain Margin = %6.2f dB", phm,
gm);
  sca(f.children(2));
  xtitle(str);
  gwin = get(f,'figure_id');
  addmenu(gwin, "Mark", ["Gain";"Phase"], list(2,'markval'));
endfunction
```

save this file as "margins.sci". Now one can call this function to show various additional informations on the bode plot as shown below:

```
//scilab version 5.2.2
clc;
clear;
mtlb_close all;
exec('/home/swg/PROG/scilab/scifunc/margins.sci',-1);

s = %s/(2*%pi);
g = syslin('c', 20/((0.5*s+1)*(0.01*s+0.1)+0.0001));
h = margins(g);
```

Multiple points on the plot can be shown by selecting "Gain" or "Phase" option under the menu "Mark" located on the menu bar of the graph.



[TOC]

PD Design Using Bode Plot

Following program demonstrates, how we can design a PD compensator for given frequency domain specifications.

```
// PID Design Using Bode Plot
// Date: 03 August 2007
// -----------------------------
```

```
clear all;
clc;
mtlb_close all;

s = %s/(2*%pi);

g = syslin('c', 10/(s^2+s+3));


//Draw bode plot of open loop system
f0 = scf(0);
bode(g, 0.1, 1000);


//---------------------------------------------------
// This part is for changing the xlabel of bode plot
a = gcf();
a.figure_name = "Bode Plot of Open Loop System";
a1 = a.children(1); // phase subplot
a2 = a.children(2); // magnitude subplot
a1.x_label.text = "Frequency (rad/sec)";
a2.x_label.text = "Frequency (rad/sec)";
//------------------------------------------------------



// Phase margin of open loop system
[phm, wg] = p_margin(g);

if(phm >= 0) then PM = phm - 180;
else PM = phm + 180;
end

// Gain Margin of closed loop system
[gm, wp] = g_margin(g);

disp("Phase Margin (deg) = "); disp(PM);
disp("Gain crossover Frequency(rad/sec) = "); disp(wg);
disp("Gain Margin = " ); disp(gm);
disp("Phase crossover frequency(rad/sec) = "); disp(wp);


//--------------------------------
// Desired Phase margin
```

```
PM_desired = 60; //deg

//Desired gain crossover frequency (rad/sec)
wg_desired = 5; // rad/sec


//---------------------------
// Since, we have s = %s/(2*pi), s represents freq in rad/sec
// However G(s) is the transfer function which is a function of
//frequency in Hertz. That is, s/(2*pi) is in Hz. But s is in
rad/sec
// To compute g at 5 rad/sec we need to multiply 2*pi to it.

// In short, we need to scale 's' back to its original value
before calculating the phase
// gain of the transfer function 'g'.
//-----------------------------------------


// Phase and magnitude of plant at gain crossover freq.
fr_plant = horner(g, %i*2*%pi*wg_desired); // wg_desired in rad/
sec (NOTE)
[G_mag, ph] = polar(fr_plant);
G_ph = real(ph) * 180 / %pi; // in degrees



//Phase angle contributed by controller
C_ph = -180 + PM_desired - G_ph; // in deg


// Proportional gain constant
Kp = cos(C_ph*%pi/180)/G_mag;

//Derivative gain constant
Kd = sin(C_ph*%pi/180)/(wg_desired*G_mag);


gc = real(Kp + Kd*s); // I don't know why I need this ...

g1 = syslin('c', gc*g);

f1 = scf(1);
bode(g1, 0.01, 100);
```

```
//---------------------------------------------------
b = gcf();
b.figure_name = "Bode Plot of Closed Loop system";
b1 = b.children(1); // phase subplot
b2 = b.children(2); // magnitude subplot
b1.x_label.text = "Frequency (rad/sec)";
b2.x_label.text = "Frequency (rad/sec)";
//---------------------------------------------


// Phase margin of close loop system
[phm1, wg1] = p_margin(g1);

if(phm1 >= 0) then PM1 = phm1 - 180;
else PM1 = phm1 + 180;
end

// Gain Margin of closed loop system
[gm1, wp1] = g_margin(g1);


disp("Phase Margin (deg) = "); disp(PM1)
disp("Gain crossover Frequency(rad/sec) = "); disp(wg1)
disp("Gain Margin = " ); disp(gm1)
disp("Phase crossover frequency(rad/sec) = "); disp(wp1)
```

-- 04 August 2007 Saturday

[TOC]

State Space Model

State Space Representation

```
g = syslin('c', A, B, C);
ssprint(g); //pretty print for linear system
cont_mat(A); // controllability matrix
```

Simulating xdot=Ax+Bu using ode solver

```
function xdot=linear(t,x,u)
A=[0 1;-25 -10]; B=[0;25];
xdot=A*x+B*u(t);
```

```
endfunction
function ut=u(t)
ut=0*t;
endfunction
x0=[0.5;0];
t=[0:0.01:3];
x=ode(x0,0,t,list(linear,u));
clf();
plot(x');
xgrid(color('lightgray'));
```

-- 03 June 2005

```
function xdot = bb(t,x,u)

A = [0 1; 0 0];
B = [0;-0.5765];

xdot = A * x + B * u;

endfunction

x0 = [0.4;0];
t = 0:0.01:5;

u = 5 * ones(size(t));
x = ode(x0, 0, t, list(bb,u));

clf();
plot(x');
xgrid(color('gray'));
```

-- 02 Aug 2007

[TOC]

Discrete time ode

This example demonstrates the use of discrete time ode solver. Note that here 'x' and 'xnext' are vectors.

```
function [xnext]=dsystem(k,x)
 fo=1.5*x(1)*x(2)/(1+x(2)*x(2)+x(1)*x(1));
 go=1.2;
 yd=sin(2*%pi*(k+1)/30);
 u=(yd-fo)/go;
 xnext=[x(2);fo+go*u];
```

```
endfunction

xbasc(); //clear the graphic window
x0=[0;0];
kvect=[1:30];
ydvect=sin(2*%pi*kvect/30);
x=ode("discrete",x0,0,kvect,dsystem); //ode solver
plot(kvect, x(2,:),'r-',kvect,ydvect,'b-.');
legend("y(k)","yd(k)");
xgrid(color('gray'));
```

The following program shows a case where f(t,Y) returns a matrix unlike the previous case. Note that Y=[X P T] is a matrix.

```
function [Ynext]=dsystem(k,Y)
 theta=[10;5];
 Si=[1+Y(1,1);1+Y(1,1)^2];
 x=Y(:,1);
 P=Y(:,2:3);
 T=Y(:,4);
 alpha=1;//+Y(1,1)^2;
 enext=(theta-T)'*Si;
 ut=-T'*Si;
 xnext=[theta'*Si+ut;0];
 Pnext=P-alpha*P*Si*Si'*P/(1+alpha*Si'*P*Si);
 Tnext=T+alpha*Pnext*Si*enext;
 Ynext=[xnext,Pnext,Tnext];
endfunction

x0=[10;0];
P0=[10 0;0 10];
T0=[0;0];
Y0=[x0,P0,T0]; //initial condition
kvect=[1:70]; //time vector
Y=ode("discrete",Y0,0,kvect,dsystem);
fin=file('open',"mat.txt","unknown");//open a file
fprintfMat('mat.txt',Y','%10.5f'); // write the matrix
```

[TOC]

ODE Solver

A number of options are available for solving differential equations. The behaviour of ode solver can be altered by changing the %ODEOPTIONS variable. In order to change this vector, type in following lines before calling 'ode' function. Note that it is case-sensitive.

```
%ODEOPTIONS = [1, 0, 0, %inf, 0, 2, 10000, 12, 5, 0, -1, -1];
```

7th option is the max number of steps (mxstep) allowed. The default value is 500. You should change it when scilab complains about it as shown below:

```
-->t0=0;x0=[0.1;0;0;sqrt(19)];tf=20;
-->y=ode(x0,t0,t,D5);
lsoda-- at t (=r1), mxstep (=i1) steps
needed before reaching tout
     where i1 is :        500
     where r1 is :   0.1256768952473E+02
```

Sometimes you get error about very small time step as this one. This shows that the problem is either stiff or mildly stiff.

```
lsoda--  caution... t (=r1) and h (=r2) are
    such that t + h = t at next step
     (h = pas). integration continues
     where r1 is :   0.6810465354779E+00   and r2 :   0.4615304787376E-20
```

For mildly stiff problems, use options like "rkf" and for stiff problems use "stiff" option. It is advisable to provide a Jacobian in case of stiff problems. A discussion on stiffness of ODE is available here.

```
t = 0:0.01:1;
rtol=1e-2; atol=1e-2;
q=ode("rkf",q0,0,t,rtol,atol,soln);
```

If ode is taking too much time then try reducing the relative and absolute tolerance variables 'rtol' and 'atol' respectively as shown above.

-- 08 April 2008, Tuesday

[TOC]

Linear Quadratic Regulator

LQR computes a control u=K*x for a system xdot=A*x+B*u so as to stabilize the plant and also to minimize a cost function given by

```
 infty
```

```
      /
J=   |  [x'Qx+u'Ru] dt
     |
    /
  0
```

The optimal control is given by

```
u= - inv(R)*B'*Px = Kx

where P is the solution of following ARE

 A'P + PA + Q - P*B*inv(R)*B'*P = 0
```

In scilab, lqr function does not take directly the matrices Q and R.

```
A=rand(2,2);B=rand(2,1); //two states, one input
Q=diag([2,5]);R=2; //Usual notations x'Qx + u'Ru

Big=sysdiag(Q,R); //Now we calculate C1 and D12

[w,wp]=fullrf(Big);C1=wp(:,1:2);D12=wp(:,3:$); //
[C1,D12]'*[C1,D12]=Big

P=syslin('c',A,B,C1,D12); //The plant (continuous-time)
[K,X]=lqr(P)
spec(A+B*K) //check stability
A'*X+X*A+Q-X*B*inv(R)*B'*X //check riccati equation
```

-- 02 April 2006

Now consider following example

```
A=[0 1;0 0];
B=[0;1];
C=[1 0];

//LQR design matrices
Q=[1 0; 0 0];
R=1;
Big=sysdiag(Q,R);
[w wp]=fullrf(Big,1e-30);
C1=wp(:,1:2);
D12=wp(:,3:$);

s1=syslin('c',A,B,C1,D12);
[K,S]=lqr(s1)
```

This gives me following error:

```
 Warning:
 Non convergence in the QZ algorithm.
 The top   4 x   4 blocks may not be in generalized Schur form.
 !--error 9999
lqr: stable subspace too small!
at line27 of function lqr called by :
[K,S]=lqr(s1)
line   -85 of exec file called by :
exec("/home/swagat/programs/PHD/ADP2/acd_so.sce");
in  execstr instruction    called by :
```

Francois Delebecque at scilab newsgroup agrees that there is a convergence problem in the Lapack routine
dhgeqz (in very specific situations) which is known but not yet definitely fixed. He provided me another version of lqr function which works for me. This file is available here.

Stephen at scilab newsgroup provides me the alternate solution which involves directly solving the riccati equation.

```
 -->A=[0,1;0,0];
 -->B=[0;1];
 -->C=[1,0];
 -->Q=[1,0;0,0];
 -->R=0.1;
 -->s1=syslin('c',A,B,C);
 -->P=riccati(A,B*inv(R)*B',Q,'c'); //solve riccati equation
 -->K=-inv(R)*B'*P; //find controller gain
 -->evals=spec(A+B*K) //check close loop eigenvalues
  evals  =
   - 1.2574334 + 1.2574334i
   - 1.2574334 - 1.2574334i
```

LQR for discrete-time system

```
//discrete-time plant
A1=s1d(2);
B1=s1d(3);
C1=s1d(4);
D1=s1d(5);

//LQR design
```

```
Q=[1 0; 0 0];
R=0.1;
G1=B1;G2=R;G=G1/G2*G1';
P=ricc(A1,G,Q,"disc");
Ko=-inv(G2+G1'*P*G1)*G1'*P*A1
spec(A1+B*Ko)
```

-- 06 April 2006

[TOC]

Discretization of linear model

In scilab, discretization can be done only through 'zero order hold' method. The following piece of code demonstrates how we can discretize a linear system in scilab

```
//continuous time linear system
A=[0 1;0 0];
B=[0;1];
C=[1 0];
D=0;
s1=syslin('c',A,B,C,D);
s1d=dscr(s1,1); //sampling time is T=1s
```

The following matlab code gives the same result

```
A=[0 1;0 0];
B=[0;1];
C=[1 0];
D=0;
s1=ss(A,B,C,D);
s1d=c2d(s1,1,'zoh'); or s1d=c2d(s1,1);
```

-- 05 April 2006

[TOC]

Boundary value problem for differential equations

Consider following differential equation:
```
xdot1 = x2
```

```
xdot2 = - x4 - 10 * sin x1
xdot3 = - x1 + 10 * x4 * cos x1
xdot4 = - x2 - x3
```

with following boundary conditions:

```
x1(0) = 2 ; x1(6) = 0 ;
x2(0) = 0 ; x2(6) = 0
```

The task is to find suitable initial conditions x3(0) and x4(0) so that the above boundary conditions at t=6 are met. Following code demonstrates the use of 'fsolve' function to solve this problem:

```
//Differential equations
function xdot=f(t,x)
 xdot=[x(2);-x(4)-10*sin(x(1));-x(1)+10*x(4)*cos(x(1));-x(2)-
x(3)];
endfunction


//for each initial condition 's', compute the value of states at
t = 6 secs.
function y = fshoot(s)
 t=0:0.01:6;
 y0=[2;0;s];
 y1=ode(y0,0,t,f);
 y=y1(1:2,find(t==6));
endfunction

// Find the values of s that makes y = 0 at t=6. Find zeros of
fshoot function.
[s,v,info]=fsolve([2;2],fshoot);

//check
t=0:0.01:6;
y0=[2;0;s];
y=ode(y0,0,t,f);
xbasc();
plot2d(t,y');
xgrid(color('gray'));
xtitle('','time','state');
```

Note that , for 'fsolve' function to work properly, there must be as many outputs of 'fshoot' function as the number of parameters ('s') to be determined. For instance, in the above example, we need to find two initial conditions s(1) and s(2) and hence, fshoot has two outputs y=y1(1:2).

-- 19 May 2006, Thursday

[TOC]

Luenberger Observer

This program demonstrates luenberger observer design for mass-spring-damper system.

```
// Mass-Spring-Damper System parameters
// xdot = A * x + B * u

A = [0 1; -0.4 -0.1];
B = [0; 1];
C = [ 1 0];

// Observer dynamics is given as follows:
// xhat_dot = A xhat + B u + MC ( x - xhat )


// Error dynamics
// edot = ( A - MC) e

//Compute Observer gain so that (A-MC) is stable
MT = ppol(A',C', [-5 -5]); //pole placement
M = MT';


// State feedback gain for controller u = - K x
K = ppol(A, B, [-3 -4]);


//combined dynamics
// z = Abar * z
//  _ _
// | A-BK | 0 |
// Abar = | .... ..... |
// |_ MC | A-MC-BK _|
//

Abar = [A-B*K zeros(2,2); M*C A-M*C-B*K];

Bbar = [0;0;0;0];
```

```
Cbar = [1 0 0 0;0 0 1 0];

sl = syslin('c', Abar, Bbar, Cbar);

z0 = [0.5; 0.2; 0.1; 0.1];
t = 0:0.01:3;

//time simulation
y = csim('step',t,sl,z0);

// plotting
xbasc();
plot(t,y)
xgrid(color('gray'));
xtitle('Luenberger Observer', 'time(sec)', 'States');
legend('Original state', 'estimated state');
```

[TOC]