Control Systems with Scilab

Aditya Sengupta

Indian Institute of Technology Bombay
apsengupta@iitb.ac.in

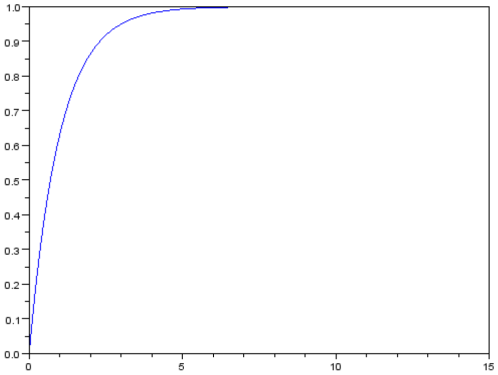
December 1, 2010, Mumbai

A simple first order system

```
// Defining a first order system:
s = %s              // The quicker alternative to using s =
    poly(0, 's')
K = 1, T = 1        // Gain and time constant
SimpleSys = syslin('c', K/(1+T*s))
```

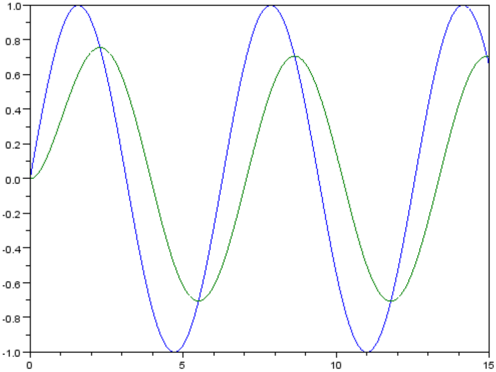
Simulating the system- Step test

```
t=0:0.01:15;
y1 = csim('step', t, SimpleSys); //step response
plot(t, y1)
```



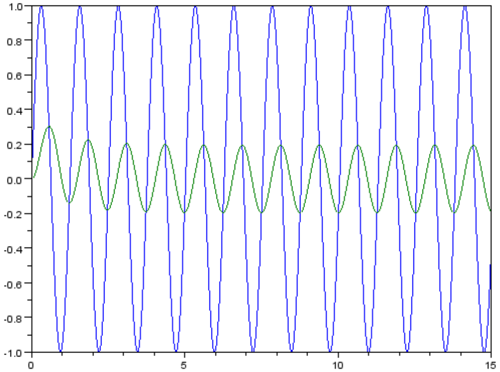
Simulating the system- Sine test

```
u2=sin(t);
y2 = csim(u2, t, SimpleSys);  //sine response
plot(t, [u2; y2]')
```



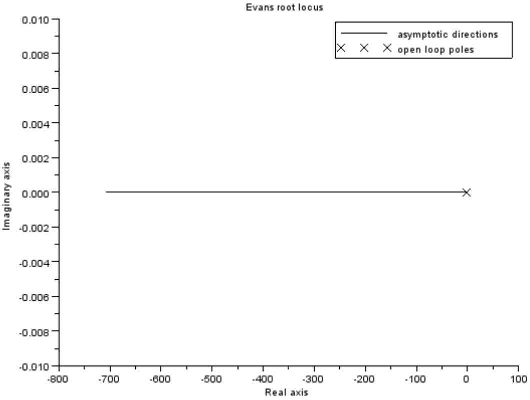
Simulating the system- Sine test

```
u3=sin(5*t);
y3 = csim(u3, t, SimpleSys);  //sine response at different
    frequency
plot(t, [u3; y3]')
```



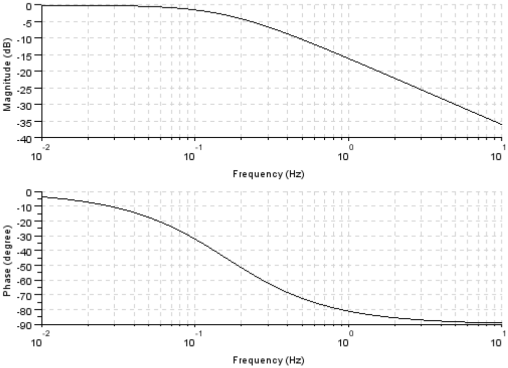
Root Locus

```
evans(SimpleSys)
```



Bode Plot

```
fMin=0.01, fMax=10
bode(SimpleSys, fMin, fMax)
```



Second Order Systems- Overdamped

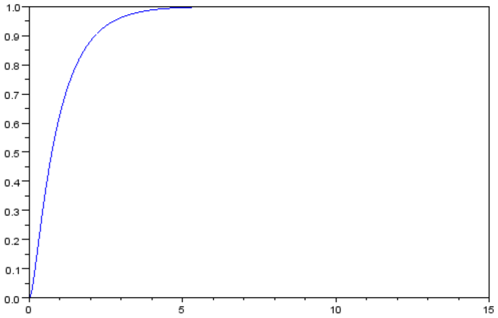
```
p=s^2+9*s+9
OverdampedSystem= syslin('c', 9/p)

roots(p)
```

Second Order Systems- Overdamped

```
p=s^2+9*s+9
OverdampedSystem= syslin('c', 9/p)

roots(p)
```

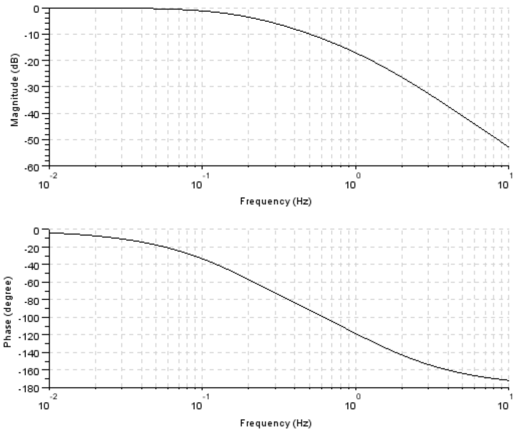
Second Order Systems- Overdamped

```
y4 = csim('step', t, OverdampedSystem);
plot(t, y4)
```



Second Order Systems- Overdamped

```
bode(OverdampedSystem, fMin, fMax)
```

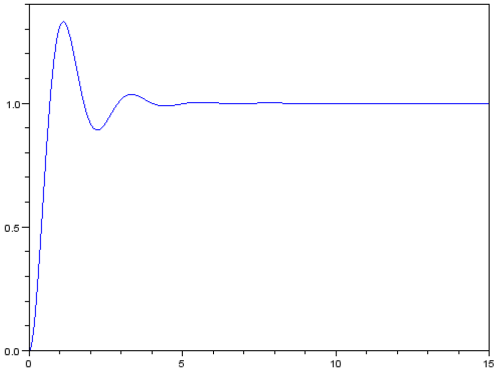


Second Order Systems- Underdamped

```
q=s^2+2*s+9
UnderdampedSystem = syslin('c', 9/q)
roots(q)
```

Second Order Systems- Underdamped

```
y5 = csim('step', t, UnderdampedSystem);
plot(t, y5)
```

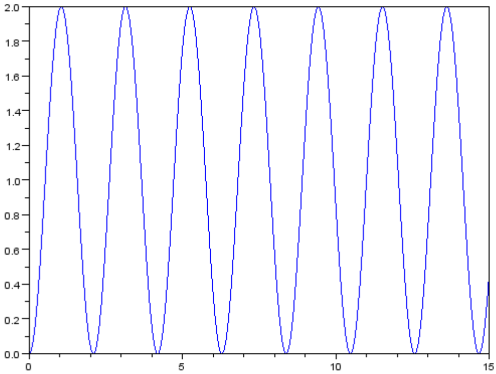


Second Order Systems- Undamped

```
r = s^2+9
UndampedSystem = syslin('c', 9/r)
roots(r)
```

Second Order Systems- Undamped

```
y6 = csim('step', t, UndampedSystem);
plot(t, y6)
```

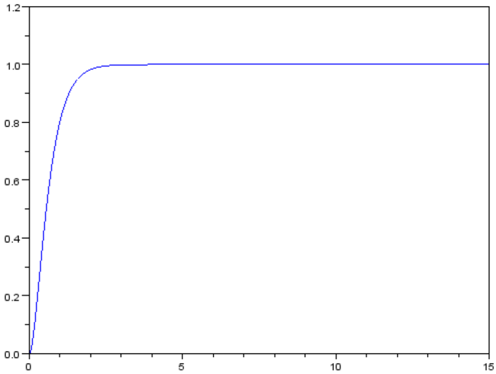


Second Order Systems- Critically damped

```
m = s^2+6*s+9
CriticallyDampedSystem = syslin('c', 9/m)
roots(m)
```

Second Order Systems- Critically damped

```
y7 = csim('step', t, CriticallyDampedSystem);
plot(t, y7)
```

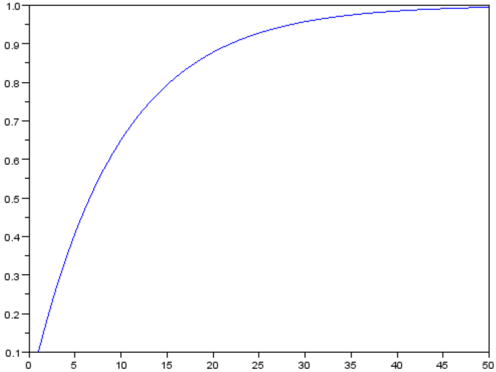


Discrete time systems

```
z = %z
a = 0.1
DTSystem = syslin('d', a*z/(z - (1-a)))
```

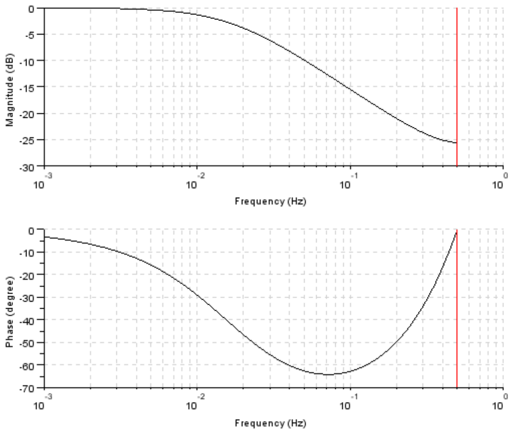
Discrete time systems

```
u = ones(1, 50);
y = flts(u, DTSystem);
plot(y) // Close this when done
```



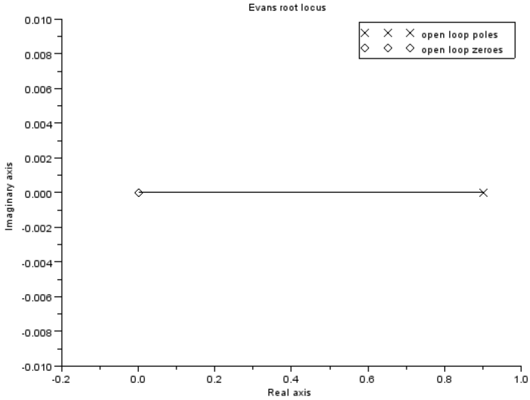
Discrete time systems

```
bode(DTSystem, 0.001, 1)
```



Discrete time systems

```
evans(DTSystem)
```



State space- representation

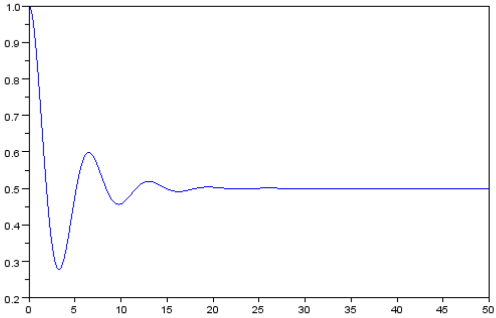
```
A = [0, 1; -1, -0.5]
B = [0; 1]
C = [1, 0]
D = [0]

x0=[1; 0] // The initial state

SSsys = syslin('c', A, B, C, D, x0)
```

State space- simulation

```
t = [0: 0.1: 50];
u = 0.5*ones(1, length(t));
[y,x] = csim(u, t, SSsys);
```

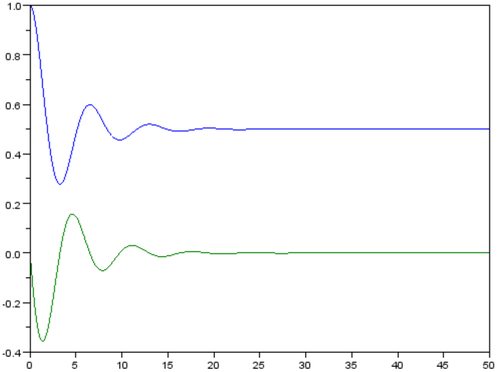
State space- simulation

```
scf(1); plot(t, y)
```



State space- simulation

```
scf(2); plot(t, x)
```



State space

```
evans(SSsys) //zoom in
// Conversion from state space to transfer function:
ss2tf(SSsys)
roots(denom(ans))
spec(A)
```

Try this: obtain the step response of the converted transfer function. Then compare this with the step response of the state space representation (remember to set the initial state (x0) and step size (u) correctly.

State space

```
evans(SSsys) //zoom in
// Conversion from state space to transfer function:
ss2tf(SSsys)
roots(denom(ans))
spec(A)
```

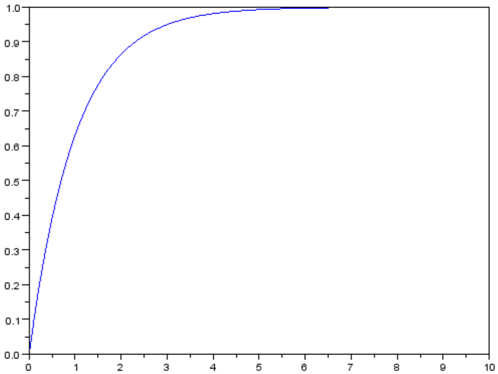
Try this: obtain the step response of the converted transfer function. Then compare this with the step response of the state space representation (remember to set the initial state (x0) and step size (u) correctly.

Discretizing continuous time systems

```
SimpleSysDiscr = ss2tf(dscr(SimpleSys, 0.1))
// Since dscr() returns a state space model
```

Discretizing continuous time systems

```
t = [0: 0.1: 10];
u = ones(t);
y = flts(u, SimpleSysDiscr);
plot(t, y)
```



Multiple subsystems

```
SubSys1 = syslin('c', 1/s)
SubSys2 = 2 // System with gain 2
Series   = SubSys1*SubSys2
Parallel = SubSys1+SubSys2
Feedback = SubSys1/.SubSys2 //Note slash-dot, not dot-slash
// Also try the above step using 2 instead of Subsys2
```

Hint: put a space after the dot and then try.

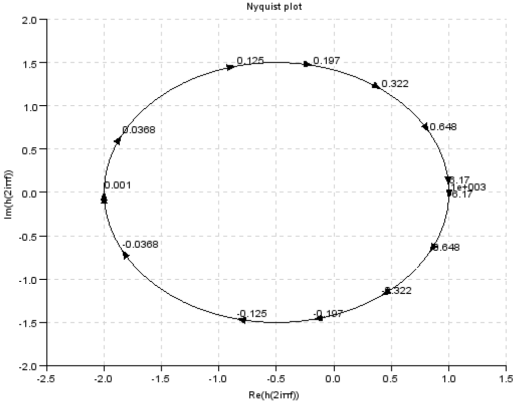
Multiple subsystems

```
SubSys1 = syslin('c', 1/s)
SubSys2 = 2 // System with gain 2
Series   = SubSys1*SubSys2
Parallel = SubSys1+SubSys2
Feedback = SubSys1/.SubSys2 //Note slash-dot, not dot-slash
// Also try the above step using 2 instead of Subsys2
```

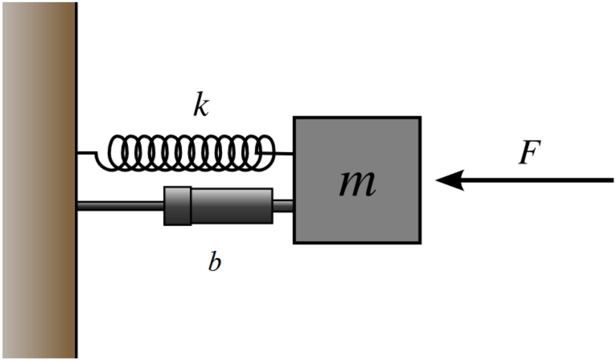
Hint: put a space after the dot and then try.

Nyquist Plot

```
G = syslin('c', (s-2)/(s+1));
H = 1
nyquist(G*H)
```



An Example



Modelling the system

$$m\ddot{x}(t) + b\dot{x}(t) + kx(t) = F(t)$$

Taking the Laplace transform:

$$ms^2X(x) + bsX(s) + kX(s) = F(s)$$

$$\frac{X(s)}{F(s)} = \frac{1}{ms^2 + bs + k}$$

We will use a scaling factor of k and represent the system as:

$$G(s) = \frac{k}{ms^2 + bs + k}$$

Modelling the system in Scilab

```
m = 1   // Mass:       kg
b = 10  // Damper:     Ns/m
k = 20  // Spring:     N/m
s = %s

// System:
System = k/(m*s^2 + b*s + k)
// We use k as our scaling factor.
```

A second order model

Comparing the system:

$$\frac{k}{ms^2 + bs + k}$$

with the standard representation of a second order model:

$$\frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

We have

$$\omega_n = \sqrt{\frac{k}{m}}$$

and

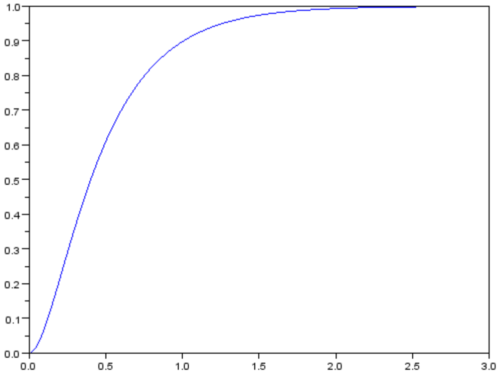
$$\zeta = \frac{b/m}{2\omega_n}$$

Understanding the system

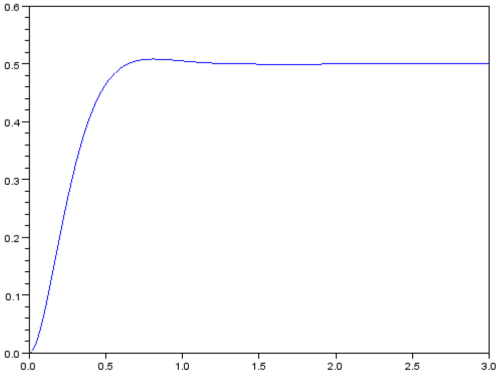
```
wn = sqrt(k/m)
zeta = (b/m)/(2*wn)
```

We note that this is an overdamped system since $\zeta > 1$

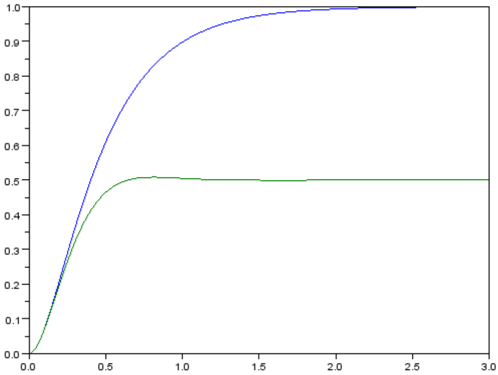
```
// Step response (open loop transfer function):
t = 0:0.01:3;
y = csim('step', t, System);
plot(t, y)
```



```
// Step response of the system with unity feedback:
t = 0:0.01:3;
yFeedback = csim('step', t, System/. 1);
plot(t, yFeedback)
```



```
// Comparing the open loop transfer function and closed loop
    transfer function:
plot(t, [y; yFeedback])
```



Steady State error

From the final value theorem:

$$\lim_{t \to \infty} f(t) = \lim_{s \to 0} sF(s)$$

```
Rs = 1/s
Gs = System

// The steady state value of the system is:

css = horner(s*System*Rs, 0)
```

After adding the feedback loop:

```
css = horner(s*(System/. 1)*Rs, 0)
```

Steady State error

From the final value theorem:

$$\lim_{t \to \infty} f(t) = \lim_{s \to 0} sF(s)$$

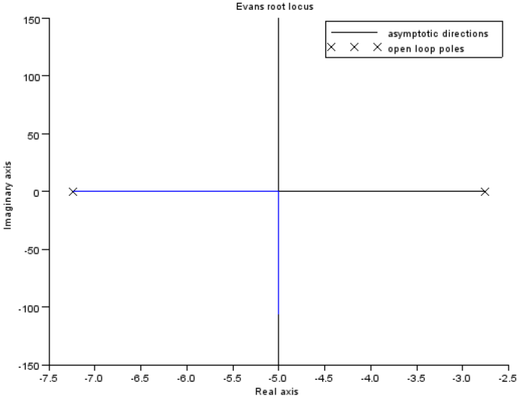
```
Rs = 1/s
Gs = System

// The steady state value of the system is:

css = horner(s*System*Rs, 0)
```

After adding the feedback loop:

```
css = horner(s*(System/. 1)*Rs, 0)
```

Lets take a look at the root locus

```
evans(System)
```



Finding the gain at a point on the root locus

Say we have the root locus in front of us.
Now we wish to calculate some system parameters at some particular point along the root locus. How do we find the position of this point?

Finding the gain at a point on the root locus

The root locus is the plot of the location of the **poles** of:

$$\frac{KG(s)}{1 + KG(s)}$$

as k varies.
That is to say, we need to find the solution of the denominator going to zero:

$$1 + KG(s) = 0$$

or

$$K = \frac{-1}{G(s)}$$

Finding the gain at a point on the root locus

We can find the location of a given point on the root locus using the locate() command.

We then need to multiply the [x; y] coordinates returned by this command with [1, % i] so that we obtain the position in the complex plane as $x + iy$

We then simply evaluate -1/G(s) at the given position using horner()

Try this now:

```
evans(System)
Kp = -1/real(horner(System, [1 %i]*locate(1)))
// Click on a point on the root locus
css = horner(s*(Kp*System/. 1)*Rs, 0)
// This is the steady state value of the closed loop system
// with gain Kp.
```

(Choose any point on the root locus right now, we will shortly see how to decide which point to choose)

Achieving specific parameters- a proportional controller

Lets say we wish to achieve the following parameters:

```
OS = 0.30      // Overshoot
tr = 0.08      // Rise time
```

We know from theory, for a second order system,

$$\zeta = \frac{-\ln OS}{\sqrt{\pi^2 + (\ln OS)^2}}$$

$$\omega_n = \frac{1}{t_r\sqrt{1-\zeta^2}}\left(\pi - \tan^{-1}\frac{\sqrt{1-\zeta^2}}{\zeta}\right)$$

In Scilab:

```
zeta = -log(OS)/sqrt(%pi^2 + log(OS)^2)
wn = (1/(tr*sqrt(1 - zeta^2)))*(%pi - atan(sqrt(1 - zeta^2))
    /zeta)
```

We use these values of ζ and ω_n to decide which point to choose on the root locus.

Achieving specific parameters- a proportional controller

Lets say we wish to achieve the following parameters:

```
OS = 0.30      // Overshoot
tr = 0.08      // Rise time
```

We know from theory, for a second order system,

$$\zeta = \frac{-\ln OS}{\sqrt{\pi^2 + (\ln OS)^2}}$$

$$\omega_n = \frac{1}{t_r\sqrt{1-\zeta^2}}\left(\pi - \tan^{-1}\frac{\sqrt{1-\zeta^2}}{\zeta}\right)$$

In Scilab:

```
zeta = -log(OS)/sqrt(%pi^2 + log(OS)^2)
wn = (1/(tr*sqrt(1 - zeta^2)))*(%pi - atan(sqrt(1 - zeta^2))
    /zeta)
```

We use these values of ζ and ω_n to decide which point to choose on the root locus.

Achieving specific parameters- a proportional controller

Lets say we wish to achieve the following parameters:

```
OS = 0.30      // Overshoot
tr = 0.08      // Rise time
```

We know from theory, for a second order system,

$$\zeta = \frac{-\ln OS}{\sqrt{\pi^2 + (\ln OS)^2}}$$

$$\omega_n = \frac{1}{t_r\sqrt{1-\zeta^2}}\left(\pi - \tan^{-1}\frac{\sqrt{1-\zeta^2}}{\zeta}\right)$$

In Scilab:

```
zeta = -log(OS)/sqrt(%pi^2 + log(OS)^2)
wn = (1/(tr*sqrt(1 - zeta^2)))*(%pi - atan(sqrt(1 - zeta^2))
    /zeta)
```

We use these values of ζ and ω_n to decide which point to choose on the root locus.

Achieving specific parameters- a proportional controller

Lets say we wish to achieve the following parameters:

```
OS = 0.30      // Overshoot
tr = 0.08      // Rise time
```

We know from theory, for a second order system,

$$\zeta = \frac{-\ln OS}{\sqrt{\pi^2 + (\ln OS)^2}}$$

$$\omega_n = \frac{1}{t_r\sqrt{1-\zeta^2}}\left(\pi - \tan^{-1}\frac{\sqrt{1-\zeta^2}}{\zeta}\right)$$

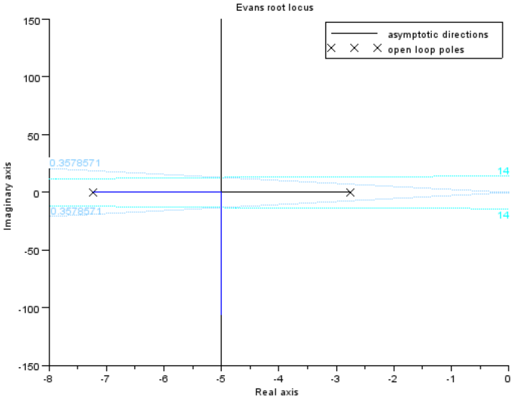
In Scilab:

```
zeta = -log(OS)/sqrt(%pi^2 + log(OS)^2)
wn = (1/(tr*sqrt(1 - zeta^2)))*(%pi - atan(sqrt(1 - zeta^2))
    /zeta)
```

We use these values of ζ and ω_n to decide which point to choose on the root locus.

Achieving specific parameters- a proportional controller

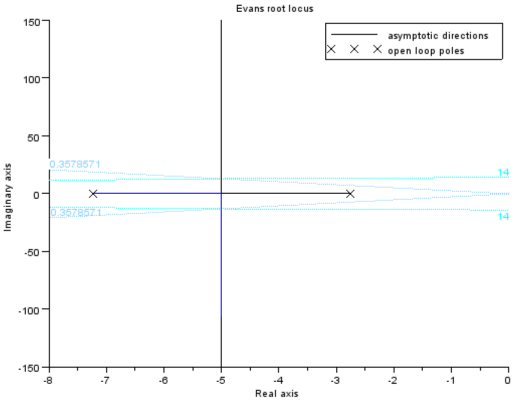
```
evans(System)
sgrid(zeta, wn)
```



Achieving specific parameters- a proportional controller

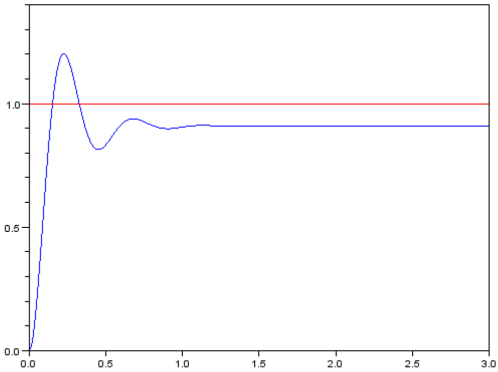
Find the value of proportional gain at some point on the root locus:

```
Kp = -1/real(horner(System, [1 %i]*locate(1)))
// Click near the intersection
```

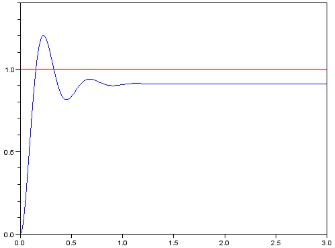


Achieving specific parameters- a proportional controller

```
PropSystem = Kp*System/. 1
yProp = csim('step', t, PropSystem);
plot(t, yProp)
plot(t, ones(t), 'r'), // Compare with step
```



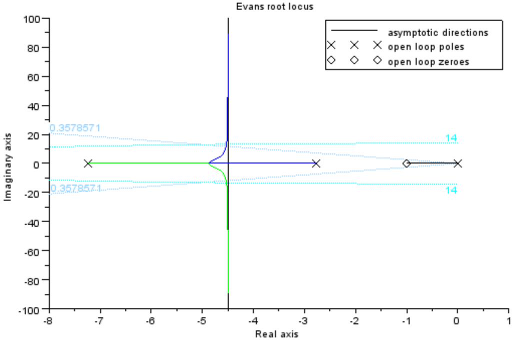
A PI controller



Note the steady state error and overshoot.
In order to eliminate the steady state error, we need to add an integrator- that is to say, we add a pole at origin.
In order to have the root locus pass through the same point as before (and thus achieve a similar transient performance), we add a zero near the origin

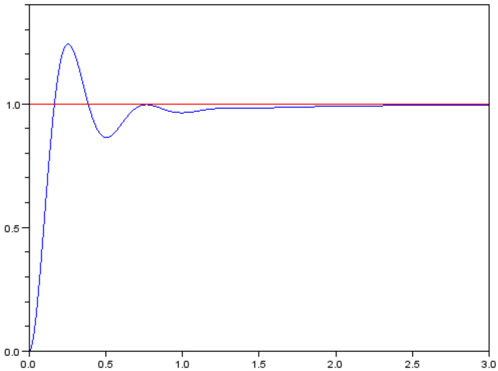
A PI controller

```
PI = (s+1)/s
evans(PI*System)
sgrid(zeta, wn)  // These values are from the Proportional
    controller
Kpi = -1/real(horner(PI*System, [1 %i]*locate(1)))
```



A PI controller

```
PISystem = Kpi*PI*System/. 1
yPI = csim('step', t, PISystem);
plot(t, yPI)
plot(t, ones(t), 'r'), // Compare with step
```



A PD controller

We wish to achieve the following parameters:

```
OS = 0.05
Ts = 0.5
```

From theory, we know the corresponding values of ζ and ω_n are:

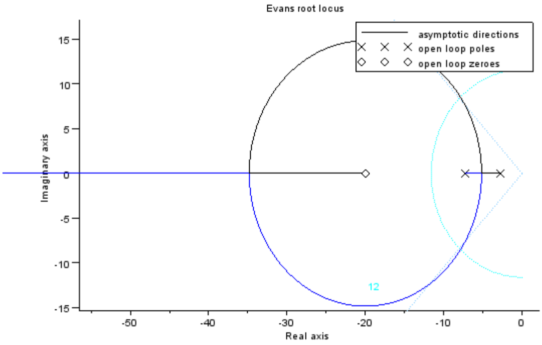
$$\zeta = \sqrt{\frac{(\log OS)^2}{(\log OS)^2 + \pi/2}}$$

$$\omega_n = \frac{4}{\zeta T_s}$$

```
zeta = sqrt((log(OS))^2/((log(OS))^2 + %pi^2))
wn = 4/(zeta*Ts)
```

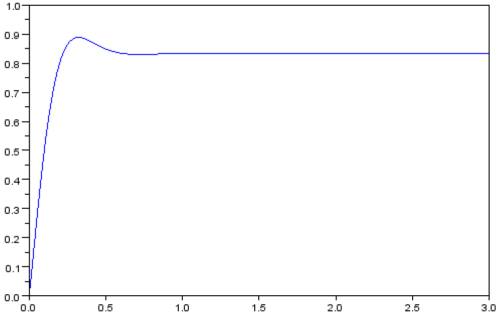
A PD controller

```
PD = s + 20
evans(PD*System)
sgrid(zeta, wn)  // Zoom first, then execute next line:
Kpd = -1/real(horner(PD*System, [1 %i]*locate(1)))
```



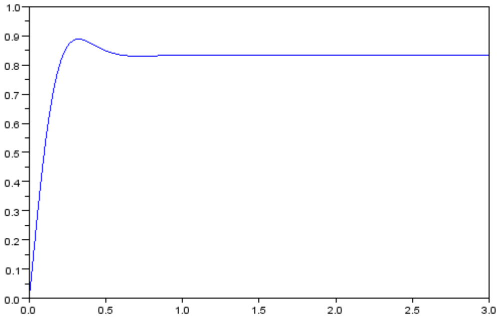
A PD controller

```
PDSystem = Kpd*PD*System/. 1
yPD = csim('step', t, PDSystem);
plot(t, yPD)
plot(t, ones(t), 'r'), // Compare with step
```



A PD controller

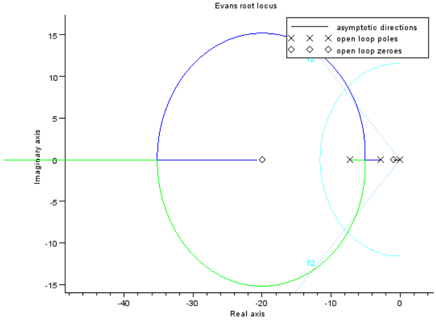
Note the improved transient performance, but the degraded steady state error:



A PID Controller

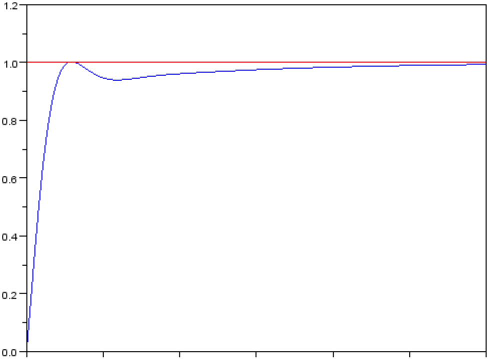
We design our PID controller by eliminating the steady state error from the PD controlled system:

```
PID = (s + 20)*(s+1)/s
evans(PID*System)
sgrid(zeta, wn) // These values are from the PD system
```

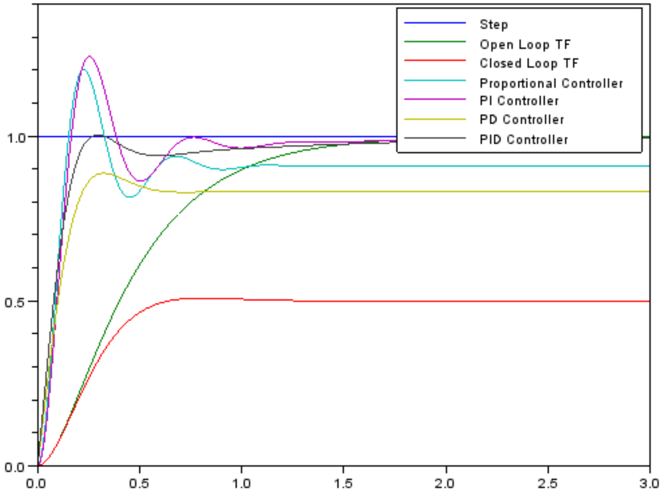


A PID Controller

```
Kpid = -1/real(horner(PID*System, [1 %i]*locate(1)))
PIDSystem = Kpid*PID*System/. 1
yPID = csim('step', t, PIDSystem);
plot(t, yPID)
plot(t, ones(t), 'r'), // Compare with step
```



A Comparison



References

- Control Systems Engineering, *Norman Nise*
- Modern Control Engineering, *Katsuhiko Ogata*
- Digital Control of Dynamic Systems, *Franklin and Powell*
- Master Scilab by Finn Haugen. http://home.hit.no/~finnh/scilab_scicos/scilab/index.htm
- Mass, damper, spring image: released into the public domain by Ilmari Karonen. http://commons.wikimedia.org/wiki/File:Mass-Spring-Damper.png