Introduction to Scilab
application to feedback control

Yassine Ariba

Brno University of Technology - April 2014

Sommaire

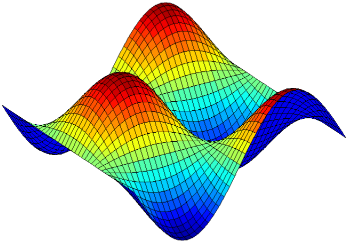
Sommaire

What is Scilab ?

Scilab is the contraction of *Scientific Laboratory*. Scilab is :

- a numerical computing software,
- an interpreted programming environment,
- used for any scientific and engineering applications,
- multi-platform : Windows, MacOS et Linux,

Created by researchers from Inria in the 90's, the software is now developed by Scilab Entreprises



`www.scilab.org`

Scilab includes hundreds of functions for various applications

- Mathematics and simulation
- 2D and 3D visualization
- Optimization
- Statistics
- Control system design and analysis
- Signal processing
- Application development

More informations : `www.scilab.org`

License

- Scilab is an *open source* software.
- It is distributed under a GPL-compatible license.
- It is a free open source alternative to MATLAB® [1].
- Scilab can be downloaded from :

`http://www.scilab.org/download/`

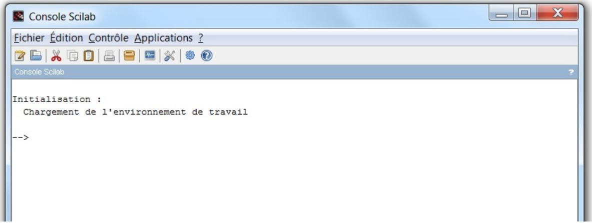
The version used in this introduction is

`version 5.4.1`

1. MATLAB is a registered trademark of The MathWorks, Inc.

Getting started

Firstly, Scilab can be used in an interactive way by typing instructions on the console.



- type scilab code on the prompt `-->`
- type `enter`, to execute it.
- Scilab return its answer on the console or in a new window for graphics.

A first simple example :

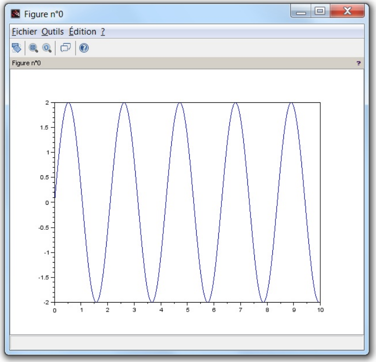
```
--> A = 2;
--> t = [0:0.01:10];
--> y = A*sin(3*t);
--> plot(t,y);
```

- Line 1 : assign the value 2 to the variable A.
- Line 2 : define a vector t that goes from 0 to 10 with a step of 0.01.
- Line 3 : compute a vector y from some mathematical operations.
- Line 4 : plot y with respect to t on a 2D graphic.

Note that “; ” prevents from printing the result of an instruction.

A first simple example :

```
--> A = 2;
--> t = [0:0.01:10];
--> y = A*sin(3*t);
--> plot(t,y);
```



A second simple example :

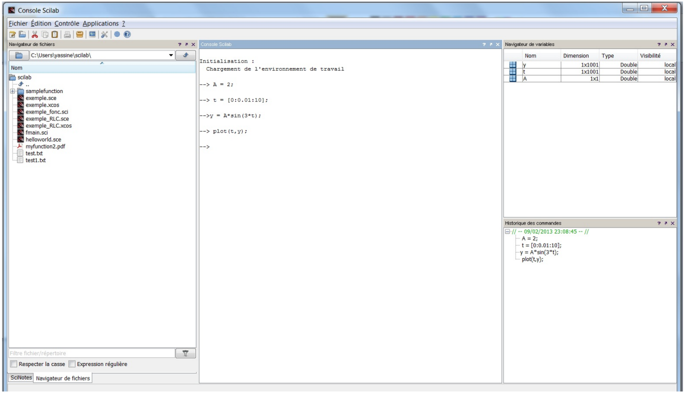
Let consider a system of linear equations

$$\left\{ \begin{array}{rcc} 2x_1 + x_2 & = & -5 \\ 4x_1 - 3x_2 + 2x_3 & = & 0 \\ x_1 + 2x_2 - x_3 & = & 1 \end{array} \right.$$

Let solve it with Scilab

```
--> A = [2 1 0 ; 4 -3 2 ; 1 2 -1];
--> b = [-5;0;1];
--> x = inv(A)*b
 x  =
    1.75
  - 8.5
  - 16.25
```

Scilab provides a graphical environment with several windows.



- the console
- command history
- file browser
- variable browser
- and others : editor, graphics, help, ...

Sommaire

Elementary operations

Simple numerical calculations :

```
--> (1+3)*0.1
 ans  =
    0.4

--> 4^2/2
 ans  =
    8.

--> 2*(1+2*%i)
 ans  =
    2. + 4.i

--> %i^2
 ans  =
  - 1.

--> cos(3)^2 + sin(3)^2
 ans  =
    1.

--> exp(5)
 ans  =
    148.41316

--> abs(1+%i)
 ans  =
    1.4142136
```

elementary operations

+	addition
-	subtraction
*	multiplication
/	right division
\	left division
^	exponents

elementary functions

sin	cos	tan	cotg
asin	acos	atan	sec
sinh	cosh	tanh	csc
abs	real	imag	conj
exp	log	log10	log2
sign	modulo	sqrt	lcm
round	floor	ceil	gcd

```
--> conj(3+2*%i)
 ans  =
    3. - 2.i

--> log10(10^4)
 ans  =
    4.
```

boolean operations

- the boolean value *true* is written : %T.
- the boolean value *false* is written : %F.

&	logical *and*
|	logical *or*
∼	logical *not*
==	equal
∼= or <>	different
< (<=)	lower than (or equal)
> (>=)	greater than (or equal)

```
--> %T & %F
 ans  =
  F

--> 2 == 2
 ans  =
  T

--> 2 < 3
 ans  =
  T
```

Variables

A variable can be directly defined via the assignment operator : “ = ”

```
--> a = 2.5;
--> b = 3;
--> c = a*b
 c  =
    7.5

--> c+d
     !--error 4
Undefined variable : d
```

- Variable names may be defined with letters a → z, A → Z, numbers 0 → 9 and few additional characters %, _, !, #, ?, $.
- Scilab is case sensitive.
- Do not confused the assignment operator “ = ” with the mathematical equal.
- Variable declaration is implicit, whatever the type.

Pre-defined variables

`%i`	imaginary number $i = \sqrt{-1}$
`%e`	Euler's number e
`%pi`	constant π
`%inf`	infinity ∞
`%t ou %T`	boolean true
`%f ou %F`	boolean false

```
--> cos(2*%pi)
 ans  =
    1.

--> %i^2
 ans  =
  - 1.
```

Sommaire

Defining and handling vectors

A vector is defined by a list of numbers between brackets :

```
--> u = [0 1 2 3]
 u  =
    0.    1.    2.    3.
```

Automatic creation

```
--> v = [0:0.2:1]
 v  =
    0.    0.2    0.4    0.6    0.8    1.
```

Syntax : **`start:step:end`**

Mathematical functions are applied element-wise

```
--> cos(v)
 ans  =
    1.    0.980    0.921    0.825    0.696    0.540
```

column vectors can also be defined with semi colon separator “ ; ”

```
--> u = [1;2;3]
 u  =
    1.
    2.
    3.
```

Some useful functions :

length	return the length of the vector
max	return the maximal component
min	return the minimal component
mean	return the mean value
sum	return the sum of all components
prod	return the product of all components

```
--> length(v)
 ans  =
    6.

--> mean(v)
 ans  =
    0.5
```

Defining and handling matrices

Matrices are defined row by row with the separator ";"

```
 --> A = [1 2 3 ; 4 5 6 ; 7 8 9]
  A  =
     1.    2.    3.
     4.    5.    6.
     7.    8.    9.
```

Particular matrices :

zeros(n,m)	$n \times m$ matrix of zeros
ones(n,m)	$n \times m$ matrix of ones
eye(n,n)	identity matrix
rand(n,m)	$n \times m$ matrix of random numbers (values $\in [0,1]$)

Accessing the elements of a matrix : `A(`*select row(s)*`,`*select column(s)*`)`

```
--> A(2,3)
 ans  =
    6.

--> A(2,:)
 ans  =
    4.    5.    6.

--> A(:,[1 3])
 ans  =
    1.    3.
    4.    6.
    7.    9.
```

For vectors, one argument is enough `v(3)` (gives `0.4`)

Elements may be modified

```
--> A(2,3) = 0;
--> A
 A  =
    1.    2.    3.
    4.    5.    0.
    7.    8.    9.
```

Some useful functions :

`size`	return the dimensions of a matrix
`det`	compute the determinant of a matrix
`inv`	compute the inverse matrix
`rank`	return the rank of a matrix
`diag`	extract the diagonal of a matrix
`triu`	extract the upper triangular part of a matrix
`tril`	extract the lower triangular part of a matrix
`spec`	return the eigenvalues of a matrix

```
--> B = [1 0 ; 2 2];
--> det(B)
 ans  =
    2.

--> inv(B)
 ans  =
    1.     0.
  - 1.     0.5

--> triu(A)
 ans  =
    1.     2.     3.
    0.     5.     6.
    0.     0.     9.
```

Matrix operations

Basic operations **+, -, *, /, ^** can be directly performed

- Watch out for dimension compatibility !
- transpose operator : ".'" , transpose and conjugate operator : "'"

```
--> C = [ 1 0 ; 3 1 ; 0 2];
--> D = [1 1 ; 4 0];
--> B + D
 ans  =
    2.    1.
    6.    2.

--> B * inv(B)
 ans  =
    1.    0.
    0.    1.

--> A * C
 ans  =

    7.     8.
    19.    17.
    31.    26.

--> A + B
       !--error 8
Inconsistent addition.
```

Elementary functions are applied element-wise

```
--> M = [0 %pi/2 ; -%pi/2 %pi ];
--> sin(M)
 ans  =
    0.     1.
  - 1.     1.225D-16

--> t = [0:0.2:1];
--> exp(t)
 ans  =
    1.   1.2214   1.4918   1.8221   2.2255   2.7182
```

There are specific versions of those functions for matrix operations

`expm`	`logm`	`sqrtm`
`sinm`	`cosm`	`^`

Element-wise operations

```
.*    ./    .^
```

```
--> A = [0 4 ; 1 2];
--> B = [1 2 ; 5 -3];
--> A * B
 ans  =
    20.  - 12.
    11.  - 4.

--> A .* B
 ans  =
    0.     8.
    5.   - 6.

--> A.^2
 ans  =
    0.     16.
    1.     4.

--> exp(t)./(t+1)
 ans  =
    1.    1.0178    1.0655    1.1388    1.2364    1.3591
```

Sommaire

2D graphics

To plot a curve in the x-y plan use function **plot**

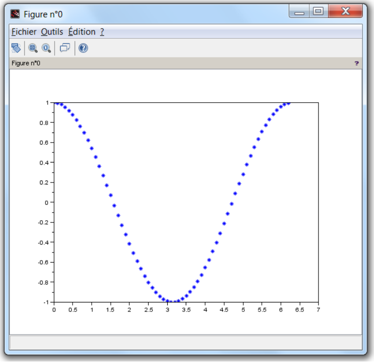
```
--> x = [0:0.1:2*%pi];
--> y = cos(x);
--> plot(x,y,'*')
```

2D graphics

To plot a curve in the x-y plan use function **plot**

```
--> x = [0:0.1:2*%pi];
--> y = cos(x);
--> plot(x,y,'*')
```

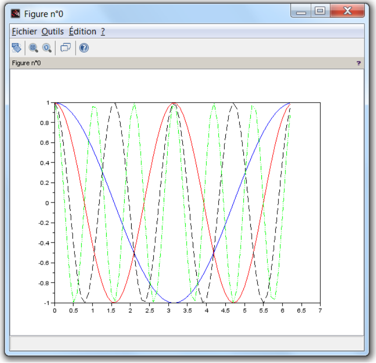
- **plot** traces a point for each couple **x(i)-y(i)**.
- **x** and **y** must have the same size.
- By default, a line is drawn between points.
- The third argument defines the style of the plot.



```
--> x = [0:0.1:2*%pi];
--> y2 = cos(2*x);
--> y3 = cos(4*x);
--> y4 = cos(6*x);
--> plot(x,y1);
--> plot(x,y2,'r');
--> plot(x,y3,'k:');
--> plot(x,y4,'g--');
```

```
--> x = [0:0.1:2*%pi];
--> y2 = cos(2*x);
--> y3 = cos(4*x);
--> y4 = cos(6*x);
--> plot(x,y1);
--> plot(x,y2,'r');
--> plot(x,y3,'k:');
--> plot(x,y4,'g--');
```

- Several graphics can be displayed.
- **clf** : clear the current graphic figure.



3D graphics

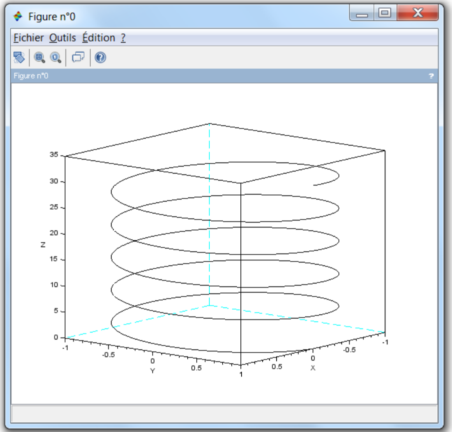
To plot a parametric curve in 3D space use function : **param3d**

```
--> t = 0:0.01:10*%pi;
--> x = sin(t);
--> y = cos(t);
--> z = t;
--> param3d(x,y,z);
```

3D graphics

To plot a parametric curve in 3D space use function : **param3d**

```
--> t = 0:0.01:10*%pi;
--> x = sin(t);
--> y = cos(t);
--> z = t;
--> param3d(x,y,z);
```

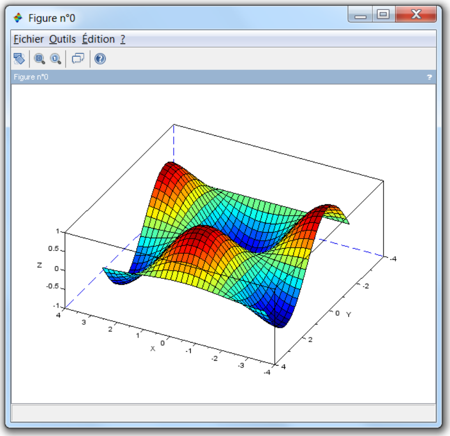


To plot a surface in 3D space use function : **surf**

```
--> x = [-%pi:0.2:%pi];
--> y = [-%pi:0.2:%pi];
--> [X,Y] = meshgrid(x,y);
-->  Z = cos(X).*sin(Y);
--> surf(X,Y,Z)
--> f=gcf();
--> f.color_map = jetcolormap(32);
```

To plot a surface in 3D space use function : **surf**

```
--> x = [-%pi:0.2:%pi];
--> y = [-%pi:0.2:%pi];
--> [X,Y] = meshgrid(x,y);
-->  Z = cos(X).*sin(Y);
--> surf(X,Y,Z)
--> f=gcf();
--> f.color_map = jetcolormap(32);
```



Overview

Scilab provides several graphical functions :

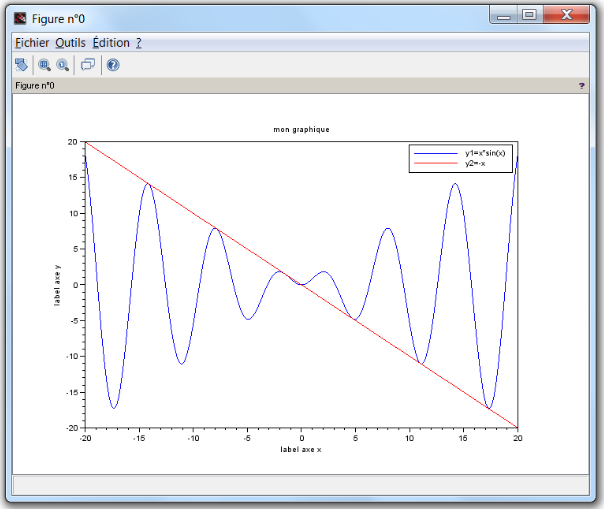
plot	2D graphic
contour	level curves in x-y plan
surf	3D surface
pie	"pie" plot
histplot	histogram plot
hist3d	3D histogram plot
bar	bar plot
polarplot	polar coordinate plot

Some instructions allow to add features to the figure :

title	add a title
xtitle	add a title and labels on axis
legend	add a legend

```
--> x = linspace(-20,20,1000);
--> y1 = x.*sin(x);
--> y2 = -x;
--> plot(x,y1,'b',x,y2,'r')
--> xtitle('mon␣graphique','label␣axe␣x','label␣axe␣y');
--> legend('y1=x*sin(x)','y2=-x');
```

```
--> x = linspace(-20,20,1000);
--> y1 = x.*sin(x);
--> y2 = -x;
--> plot(x,y1,'b',x,y2,'r')
--> xtitle('mon graphique','label axe x','label axe y');
--> legend('y1=x*sin(x)','y2=-x');
```



Sommaire

Scripts

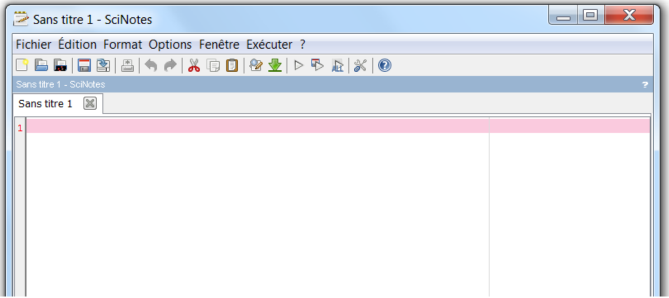
A script is a set of instructions gathered in a file.

- Scilab provides a programming language (interpreted).
- Scilab includes an editor, but any text editor may be used.
- File extension should be "`.sce`" (but this is not mandatory).
- Editor launched from "*Applications > SciNotes*" or by typing `editor` on the console.

Scripts

A script is a set of instructions gathered in a file.

- Scilab provides a programming language (interpreted).
- Scilab includes an editor, but any text editor may be used.
- File extension should be "`.sce`" (but this is not mandatory).
- Editor launched from "*Applications > SciNotes*" or by typing `editor` on the console.



Example of a script : `myscript.sce`

```
// radius of a sphere
 r = 2;

 // calculation of the area
 A = 4*%pi*r^2;

 // calculation of the volume
 V = 4*%pi*r^3/3;

 disp(A,'Area:');

 disp(V,'Volume:');
```

Dans la console :

```
-->exec('myscript.sce', -1)

 Area:

    50.265482

 Volume:

    33.510322
```

The file must be located in the *current directory*

- Comments : words following `//` are not interpreted.
- The current directory can be modified in menu *File* of the console.
- The path may be specified instead

```
exec('C:\Users\yassine\scilab\myscript.sce', -1)
```

- Scripts may also be run from the shortcut in the toolbar.
- Variables defined in the workspace (from the console) are visible and can be modified in the script.

Another example : **myscript2.sce**

```
x1 = -1; x2 = 1;
x = linspace(x1,x2,n);

y = exp(-2*x).*sin(3*x);

plot(x,y);
disp('see␣plot␣on␣the␣figure');
```

On the console :

```
--> n = 50;
-->exec('myscript2.sce', -1)

 see plot on the figure
```

Here the variable **n** must be defined beforehand.

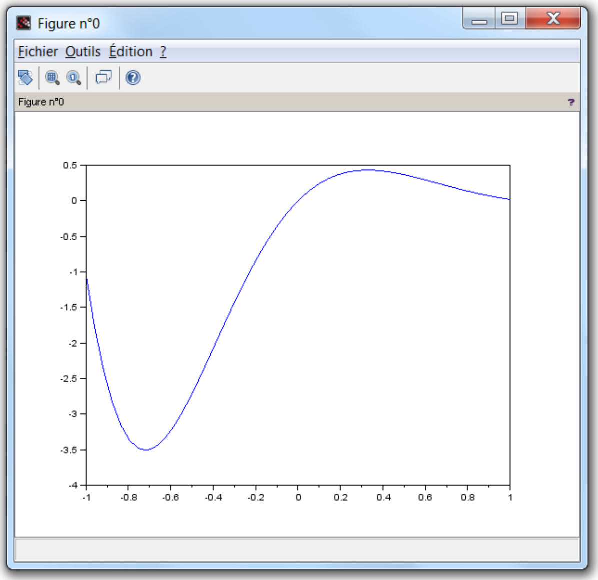


Figure n°0
Fichier Outils Édition ?
Figure n°0
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
-4
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

Looping and branching

Scilab language includes classical control structures

Conditional statements **if**

if *boolean expression* **then**
instructions 1
else
instructions 2
end

```
if (x>=0) then
    disp("x␣is␣positive");
else
    disp("x␣is␣negative");
end
```

Branching with respect to the value of a variable `select`

select *variable*

case *value 1*

    instructions 1

case *value 2*

    instructions 2

else

    instructions 3

end

```
select i
case 1
    disp("One");
case 2
    disp("Two");
case 3
    disp("Three");
else
    disp("Other");
end
```

Loop control statements **for**

for *variable* **=** *start*: *step*: *end*

instructions

end

```
n = 10;
for k = 1:n
    y(k) = exp(k);
end
```

Loop control based on a boolean expression `while`

while (*boolean expression*)

instructions

end

```
x = 16;
while ( x > 1 )
    x = x/2;
end
```

And also :

- instruction **break** interrupt and exit a loop.
- instruction **continue** skip to the next iteration of a loop.

Note that as much as possible, use vector / matrix operations instead of loops. The code may run 10 to 100 times faster. This feature of Scilab is known as the *vectorization*.

```
tic
S = 0;
for k = 1:1000
    S = S + k;
end
t = toc(); disp(t);

tic
N = [1:1000];
S = sum(N);
t = toc(); disp(t);
```

```
-->exec('myscript.sce', -1)

    0.029

    0.002
```

Functions

A function is a command that makes computations from variables and returns a result

```
outvar = afunction( invar )
```

- **afunction** is the name of the function
- **invar** is the input argument
- **outvar** is the output argument, returned by the function

Examples :

```
--> y = sin(1.8)
 y  =

    0.9738476

--> x =[0:0.1:1];

--> N = length(x)
 N  =

    11.
```

User can define its own functions

```
function [out1,out2,...] = myfunction(in1,in2,...)

    body of the function

endfunction
```

- once the environment `function`...`endfunction` is executed `myfunction` is defined and loaded in Scilab
- after any change in the function, it must be reloaded to be taken into account
- files including functions generally have the extension "`.sci`"

Example 1 : calculation of the roots of a quadratic equation.

Define and load the function

```
function [x1,x2] = roots_equ2d(a,b,c)
    // roots of ax^2 + bx + c = 0
    delta = b^2 - 4*a*c
    x1 = (-b - sqrt(delta))/(2*a)
    x2 = (-b + sqrt(delta))/(2*a)
endfunction
```

Then, you can use it as any other Scilab function

```
--> [r1,r2] = roots_equ2d(1,3,2)
 r2  =
  - 1.

 r1  =
  - 2.
```

Example 2 : functions are appropriate to define mathematical functions.

$$f(x) = (x+1)\, e^{-2x}$$

```
function y = f(x)
      y = (x+1).*exp(-2*x);
endfunction
```

```
--> y = f(4)
 y   =
    0.0016773

--> y = f(2.5)
 y   =
    0.0235828

--> t = [0:0.1:5];

--> plot(t,f)
```

- Variables from workspace are known inside the function
- but any change inside the function remain local.

```
function z=mytest(x)
       z = x + a;
       a = a +1;
endfunction
```

```
--> a = 2;
--> mytest(3)
 ans  =
    5.

--> a
 a  =
    2.
```

Sommaire

For MATLAB users

Many instructions have the same syntax, but some others not...

A dictionary gives a list of the main MATLAB functions with their Scilab equivalents

`http://help.scilab.org/docs/5.4.1/en_US/section_36184e52ee88ad558380be4e92d3de21.html`

Some tools are provided to convert MATLAB files to Scilab (e.g. `mfile2sci`)

`http://help.scilab.org/docs/5.4.1/en_US/About_M2SCI_tools.html`

A good note on Scilab for MATLAB users

Eike Rietsch, *An Introduction to Scilab from a Matlab User's Point of View*, May 2010

`http://www.scilab.org/en/resources/documentation/community`

Somme differences about the syntax

In MATLAB

- search with keywords `lookfor`
- comments `%`
- predefined constants `i`, `pi`, `inf`, `true`
- special characters in name of variables `_`
- continuation of a statement `...`
- flow control `switch case otherwise`
- last element of a vector `x(end)`

In Scilab

- search with keywords `apropos`
- comments `//`
- predefined constants `%i`, `%pi`, `%inf`, `%t`
- special characters in name of variables `_`, `#`, `!`, `?`, `$`
- continuation of a statement `..`
- flow control `select case else`
- last element of a vector `x($)`

Different responses for a same command

In MATLAB

- **length**, the larger of the number of rows and columns
- after a first **plot**, a second one clears the current figure
- division by a vector
```
>> x = 1/[1 2 3]
Error using / Matrix dimensions must
agree.
```
- operators == and ~= compare elements
```
>> [1 2 3] == 1
ans =
1 0 0
>> [1 2 3] == [1 2]
Error using ==
Matrix dimensions must agree.
>> [1 2] == ['1','2']
ans =
0 0
```

In Scilab

- **length**, the product of the number of rows and columns
- after a first **plot**, a second one holds the previous
- division by a vector
```
--> x = 1/[1 2 3]
x =
0.0714286
0.1428571
0.2142857
```
x is solution of `[1 2 3]*x = 1`
- operators == and ~= compare objects
```
--> [1 2 3] == 1
ans =
T F F
--> [1 2 3] == [1 2]
ans =
F
--> [1 2] == ['1','2']
ans =
F
```

Different responses for a same command

In MATLAB

- for a matrix `A=[1 2 4;4 8 2;6 0 9]`

```
>> max(A)
ans =
7 8 9
>> sum(A)
ans =
12 10 18
```

- `disp` must have a single argument

```
>> a=3;
>> disp(['the result is
',int2str(a),' ...bye!'])

the result is 3 ...bye!
```

In Scilab

- for a matrix `A=[1 2 4;4 8 2;6 0 9]`

```
--> max(A)
ans =
9.
--> sum(A)
ans =
36.
```

- `disp` may have several arguments

```
--> a = 3;
--> disp(a,'the result is ' +
string(a),'hello!')

hello!
the result is 3
3.
```

- note that : `prettyprint` generates the Latex code to represent a Scilab object

Difference when running a script

In MATLAB

- script is invoked by typing its name `myscript`
- the m-file must be in a directory of the search path (or specify the path in the call)
- use a semi-colon to print or not the result of an instruction

In Scilab

- script is invoked with the `exec` command

```
--> exec('myscript.sce')
```

- the file must be the working directory (or specify the path in the call)
- a second argument may be appended (mode) to specify what to print
- it does not seem to do what the documentation says... not clear for me

a simple example, *myscript.sce* :

```
// a simple script: myscript
a = 1
b = a+3;
disp('result␣is␣'+string(b))
```

the second argument *mode*

Value	Meaning
0	the default value
-1	print nothing
1	echo each command line
2	print prompt $-->$
3	echo + prompt
4	stop before each prompt
7	stop + prompt + echo

```
--> exec('myscript.sce',0)
 a  =

    1.

 result is 4
```

(as Matlab works)

```
--> exec('myscript.sce',-1)

 result is 4
```

(only output of disp is printed)

```
--> exec('myscript.sce',1)
-->// a simple script: myscript
-->a = 1
 a  =

    1.
-->b = a+3;
-->disp('result␣is␣'+string(b))

 result is 4
```

(everything is printed (instructions and outputs)

Difference when using user defined functions

In MATLAB

- a function is a file, they must have the same name
- variables in the function are local variables
- any other functions defined in the file are local functions

In Scilab

- a function is a variable
- variables in the function are local variables and variables from the calling workspace are known
- when defined (`function ... endfunction`), functions are not executed bu loaded
- any change in the function requires to reload it (executing the environment)

Sommaire

Xcos

Xcos is a graphical environment to simulate dynamic systems.

It is the Simulink® counterpart of Scilab.

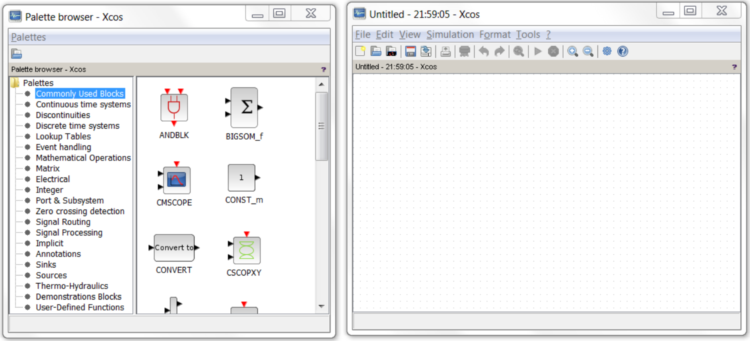
It is launched in *Application/Xcos* or by typing `xcos`

Xcos

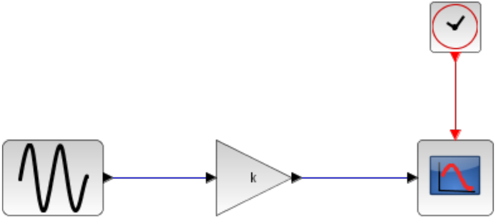
Xcos is a graphical environment to simulate dynamic systems.

It is the Simulink® counterpart of Scilab.

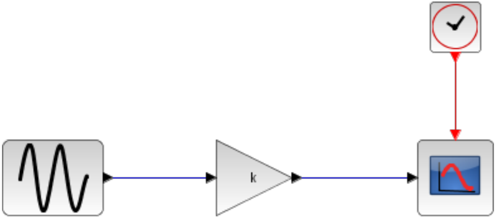
It is launched in *Application/Xcos* or by typing `xcos`



A simple example



A simple example

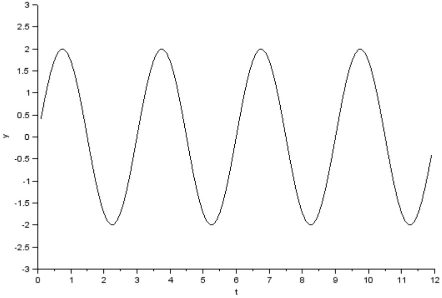


block	sub-palette
sinus	Sources/GENSIN_f
gain	Math. Operations/GAINBLK_f
scope	Sinks/CSCOPE
clock	Sources/CLOCK_c

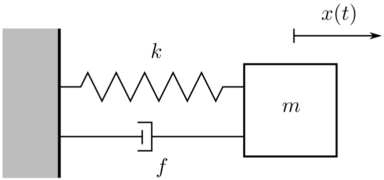
- drag and drop blocks from the palette browser to the editing window
- k is variable from the workspace (or from Simulation/Set context)
- black lines are data flows and red lines are event flows

Settings : frequency $= 2\pi/3$, $k = 2$, final integral time $= 12$, Ymin$= -3$, Ymax$= 3$, Refresh period $= 12$

Run simulation from Simulation/Start



Let simulate a mass-spring-damper system

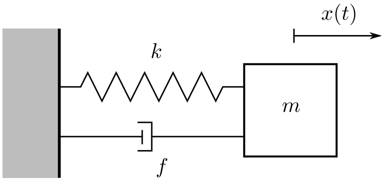


The system can be described by the equation of motion

$$m\ddot{x}(t) + f\dot{x}(t) + kx(t) = 0$$

with the initial conditions : $x(0) = 5$ and $\dot{x}(0) = 0$

Let simulate a mass-spring-damper system



The system can be described by the equation of motion

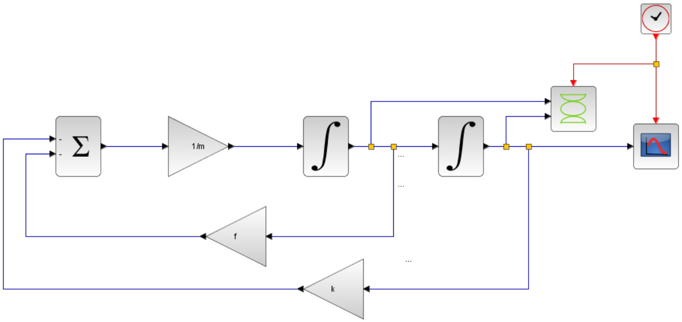
$$m\ddot{x}(t) + f\dot{x}(t) + kx(t) = 0$$

with the initial conditions : $x(0) = 5$ and $\dot{x}(0) = 0$

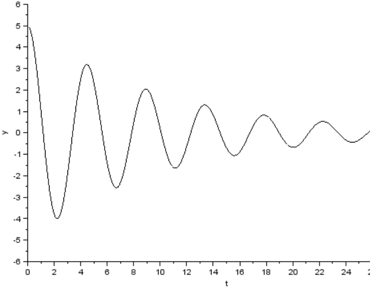
The acceleration of the mass is then given by

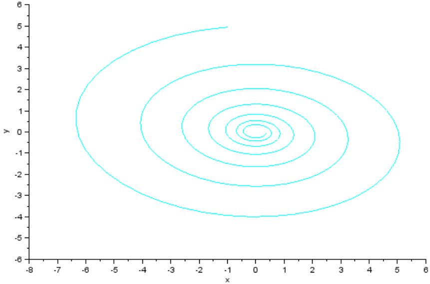
$$\ddot{x}(t) = -\frac{1}{m}\Big(kx(t) + f\dot{x}(t)\Big)$$

modeling and simulation with Xcos



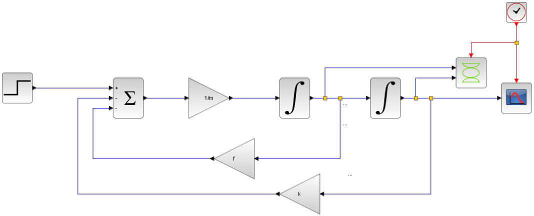
block	sub-palette
sum	Math. Operations/BIGSOM_f
gain	Math. Operations/GAINBLK_f
integral	Cont. time systems/INTEGRAL_m
scope	Sinks/CSCOPE
x-y scope	Sinks/CSCOPXY
clock	Sources/CLOCK_c



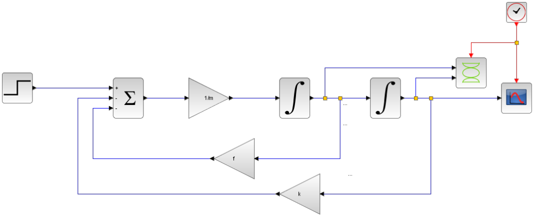


parameters : $m = 1$, $k = 2$ and $f = 0.2$

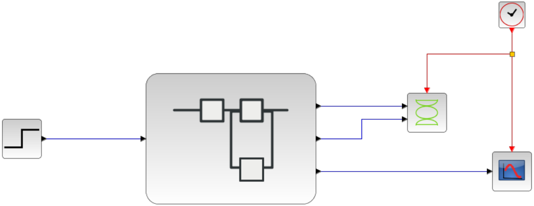
Let add an external force



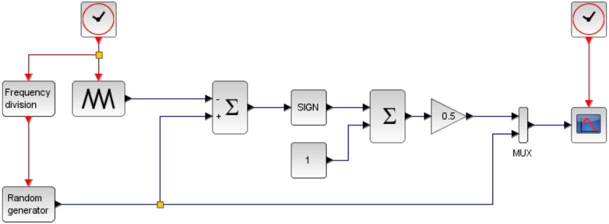
Let add an external force



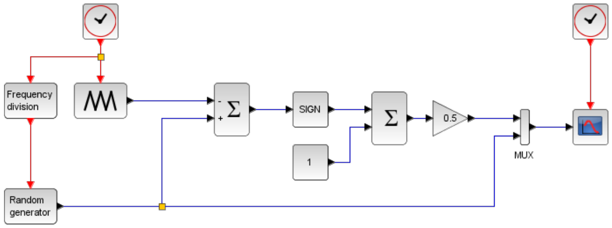
Define a superblock : *Edit/Region to superblock*

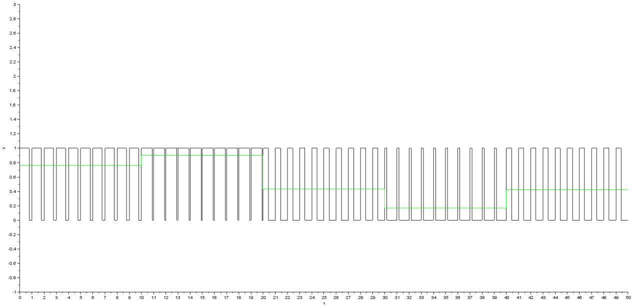


Example 3 : simulation of a PWM signal



Example 3 : simulation of a PWM signal

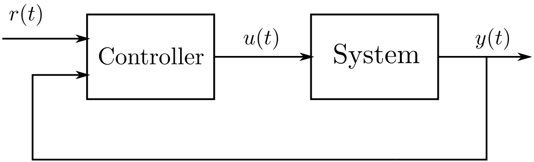




Sommaire

A brief review

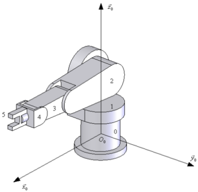
Objective : Design a controller to control a dynamical system.

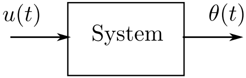


The output to be controlled is measured and taken into account by the controller.

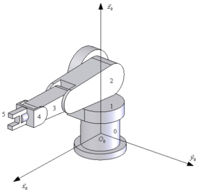
$\Rightarrow$ feedback control

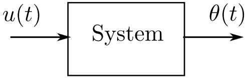
Example : angular position control of a robotic arm.





Example : angular position control of a robotic arm.





- $u(t)$ is the control voltage of the DC motor (actuator)
- $\theta(t)$ is the angular position of the arm (measured with a sensor)

The input-output relationship is given by :

$$\ddot{\theta}(t) + \dot{\theta}(t) = u(t)$$

The corresponding transfer function is

$$G(s) = \frac{\hat{\theta}(s)}{\hat{u}(s)} = \frac{1}{(s+1)s}$$

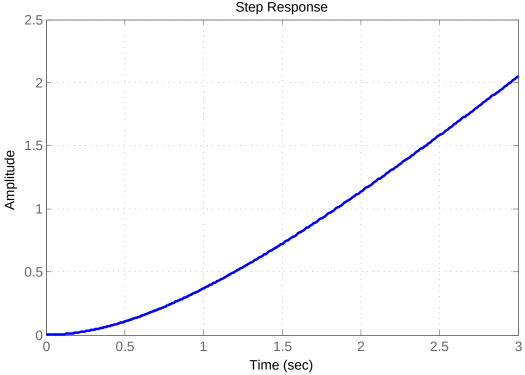
It has 2 poles : -1 and $0 \Rightarrow$ system is unstable

The corresponding transfer function is

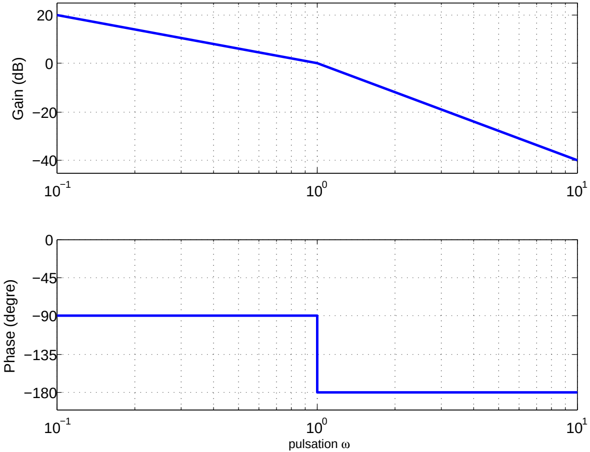
$$G(s) = \frac{\hat{\theta}(s)}{\hat{u}(s)} = \frac{1}{(s+1)s}$$

It has 2 poles : -1 and $0 \Rightarrow$ system is unstable

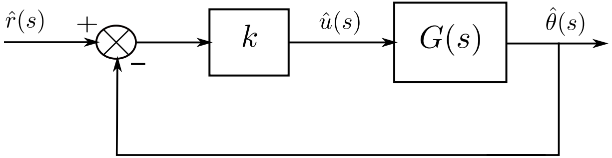
Its step response is divergent



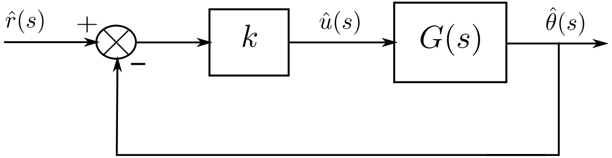
The asymptotic bode diagram :



Closed-loop control with a proportional gain k



Closed-loop control with a proportional gain k

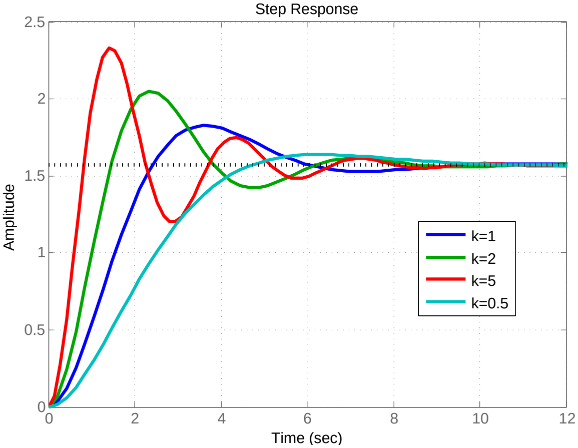


The closed-loop transfer function is

$$F(s) = \frac{k}{s^2 + s + k}$$

The Routh criterion shows that $F(s)$ is stable $\forall k > 0$.

Response of $\theta(t)$ for a step reference $r(t) = \frac{\pi}{2}$



Quick analysis of the feedback system

The tracking error is given by : $\varepsilon(t) = r(t) - \theta(t)$

$$\hat{\varepsilon}(s) = \frac{s^2 + s}{s^2 + s + k}\hat{r}(s)$$

the static error is zero : $\varepsilon_s = \lim_{s \to 0} s\,\hat{\varepsilon}(s) = 0$ (with $\hat{r}(s) = \frac{\pi/2}{s}$)

Quick analysis of the feedback system

The tracking error is given by : $\varepsilon(t) = r(t) - \theta(t)$

$$\hat{\varepsilon}(s) = \frac{s^2 + s}{s^2 + s + k}\hat{r}(s)$$

the static error is zero : $\varepsilon_s = \lim_{s \to 0} s\,\hat{\varepsilon}(s) = 0$ (with $\hat{r}(s) = \frac{\pi/2}{s}$)

Using the standard form of 2$^{\text{nd}}$ order systems :

$$F(s) = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \qquad \Rightarrow \qquad \left\{ \begin{array}{rcl} K & = & 1, \\ \omega_n & = & \sqrt{k} \\ \zeta & = & 1/2\sqrt{k} \end{array} \right.$$

we can conclude that

- when $k \nearrow$, damping $\zeta \searrow$ and oscillations $\nearrow$
- settling time $t_{5\%} \approx \frac{3}{\zeta\omega_n} = 6s$.

System analysis in Scilab

Definition of a transfer function

```
--> num = 1;
--> den = %s^2+%s;
--> G = syslin('c',num,den)
 G  =

       1
     -----
          2
     s + s

--> roots(den)
 ans  =

   - 1.
     0
```

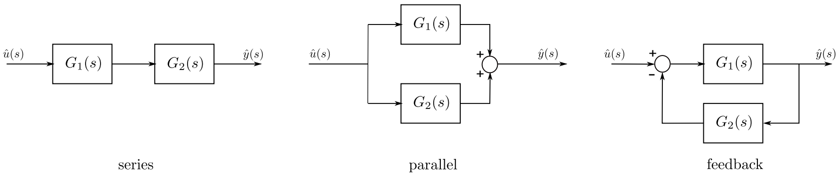
- The argument `c` stands for continuous-time system (`d` for discrete)
- The instruction `roots` is useful to calculate the poles of a transfer function
- The instruction `plzr` plots the pole-zero map in the complex plane

Computation of the time response

```
--> t = [0:0.02:3];

--> theta = csim('step',t,G);

--> plot(t,theta)
```

- The string argument **`step`** is the control, it can be **`impuls`**, a vector or a function.
- To define the time vector, you may also use the **`linspace`** instruction.
- For frequency analysis, different instructions are provided : **`repfreq`**, **`bode`**, **`nyquist`**, **`black`**.

Systems connection



The mathematical operators can handle **syslin** type

Example

$$G_1(s) = \frac{1}{s+2} \qquad \text{and} \qquad G_2(s) = \frac{4}{s}$$

```
--> G1 = syslin('c',1,%s+2);
--> G2 = syslin('c',4,%s);
```

```
--> G1 * G2      // series connection
 ans  =
      4
    -----
            2
    2s + s

--> G1 + G2     // parallel connection
 ans  =
    8 + 5s
    ------
            2
    2s + s

--> G1 /. G2    // feedback connection
 ans  =
        s
    ---------
                2
    4 + 2s + s
```

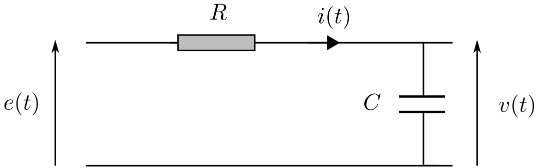
Back to our case study

Let simulate the closed-loop control with a proportional gain

```
--> k = 2;
--> F = (G*k) /. 1
 F   =
          2
    ---------
               2
    2 + s + s

--> routh_t(%s^2+%s+2)
 ans   =

    1.     2.
    1.     0.
    2.     0.

--> [wn, zeta] = damp(F)
 zeta   =
    0.3535534
    0.3535534
 wn   =
    1.4142136
    1.4142136

--> t = linspace(0,12,200);
--> theta = csim('step',t,F)*%pi/2;
--> plot(t,theta)
```

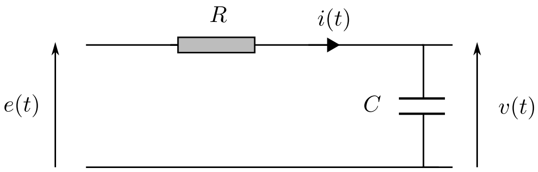
Bode plot

Introductory example : RC circuit



Bode plot

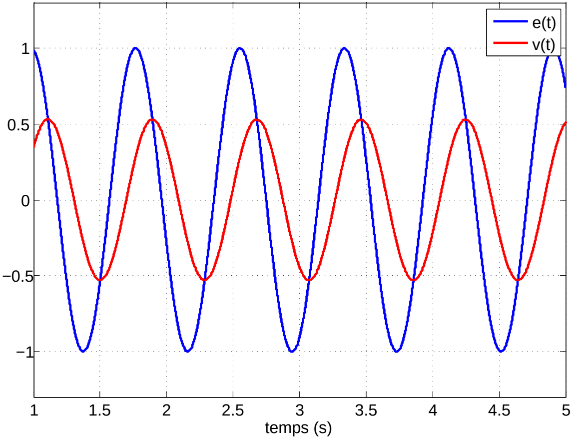
Introductory example : RC circuit



Sinusoidal steady state :

$$\begin{cases} e(t) = e_m \cos(\omega t + \phi_e) \\ v(t) = v_m \cos(\omega t + \phi_v) \end{cases} \Rightarrow \begin{cases} \underline{e} = e_m e^{j\phi_e} \\ \underline{v} = v_m e^{j\phi_v} \end{cases}$$

For $R = 1k\Omega$ and $C = 200\mu F$, let apply a voltage $e(t) = \cos(8t)$.



Ohm's law : $\underline{u} = \underline{Z}\underline{i}$

$$\underline{Z}_R = R \qquad \text{and} \qquad \underline{Z}_C = \frac{1}{j\omega C}$$

Ohm's law : $\underline{u} = \underline{Z}\underline{i}$

$$\underline{Z}_R = R \qquad \text{and} \qquad \underline{Z}_C = \frac{1}{j\omega C}$$

Applying the voltage divider formula :

$$\underline{v} = \frac{\underline{Z}_C}{\underline{Z}_C + \underline{Z}_R}\underline{e}$$

Ohm's law : $\underline{u} = \underline{Z}\underline{i}$

$$\underline{Z}_R = R \qquad \text{and} \qquad \underline{Z}_C = \frac{1}{j\omega C}$$

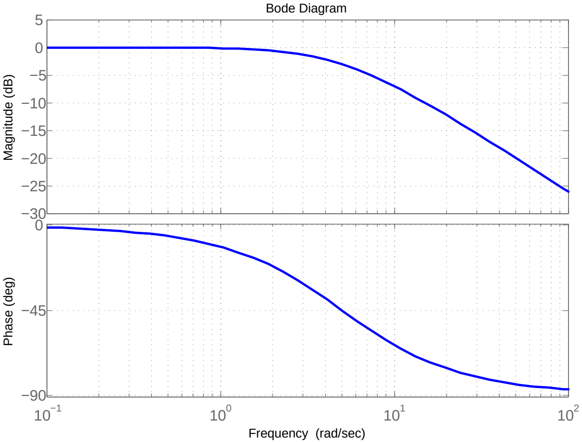
Applying the voltage divider formula :

$$\underline{v} = \frac{\underline{Z}_C}{\underline{Z}_C + \underline{Z}_R}\underline{e}$$

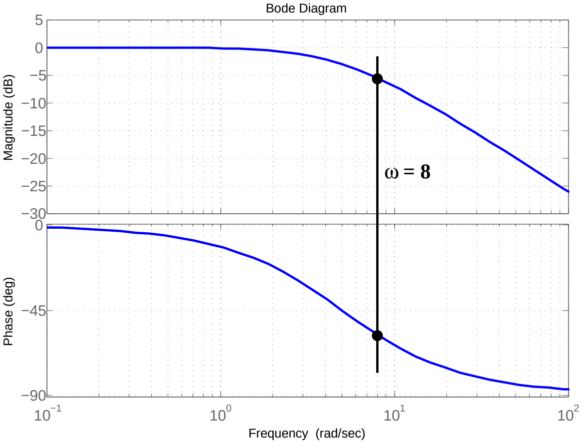
Hence, the transfer function from $e(t)$ to $v(t)$ is :

$$\underline{T} = \frac{\frac{1}{j\omega C}}{\frac{1}{j\omega C} + R} = \frac{1}{j\omega RC + 1}.$$

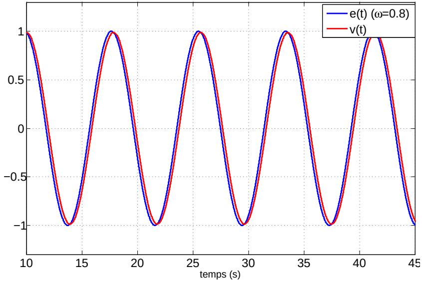
Bode diagram of the transfer function



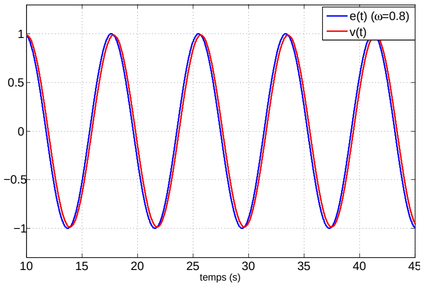
Bode diagram of the transfer function

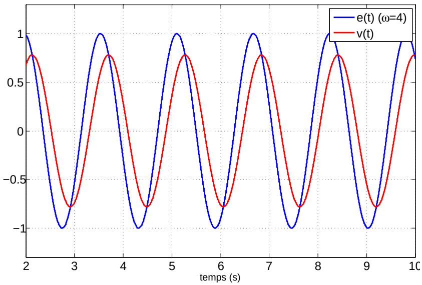


Responses of the circuit with $\omega = \{0.8, 4, 8, 40\}$

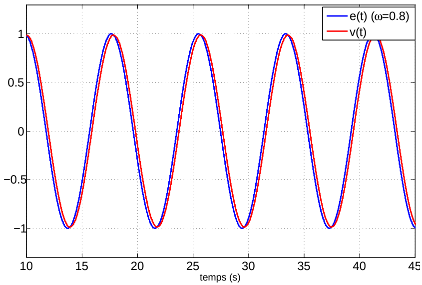


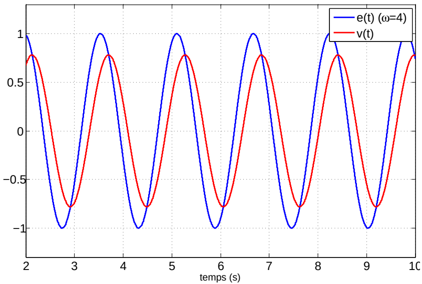
Responses of the circuit with $\omega = \{0.8, 4, 8, 40\}$

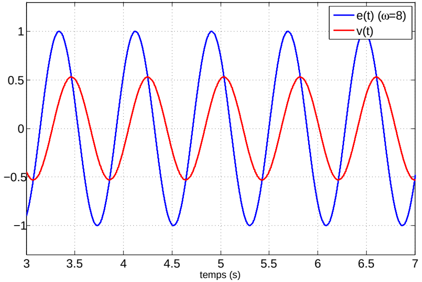




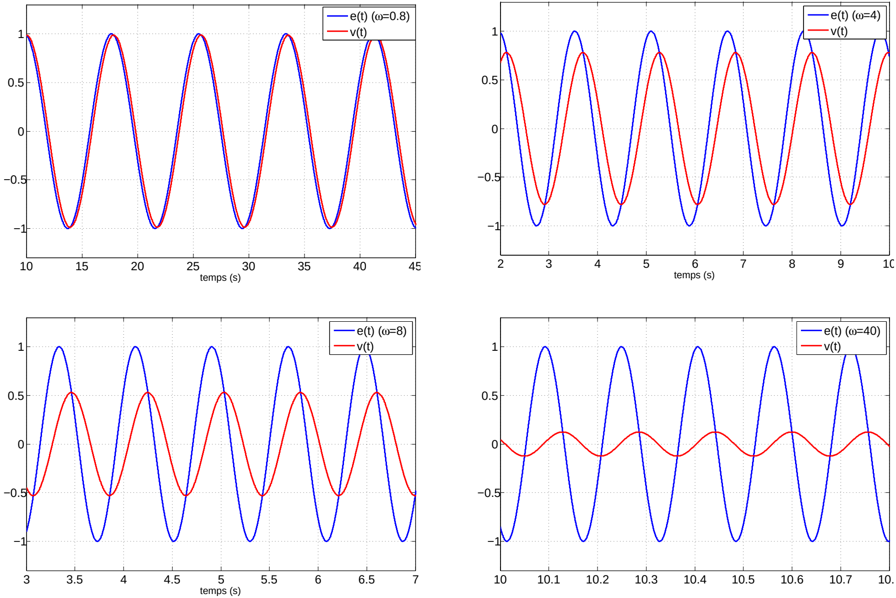
Responses of the circuit with $\omega = \{0.8, 4, 8, 40\}$

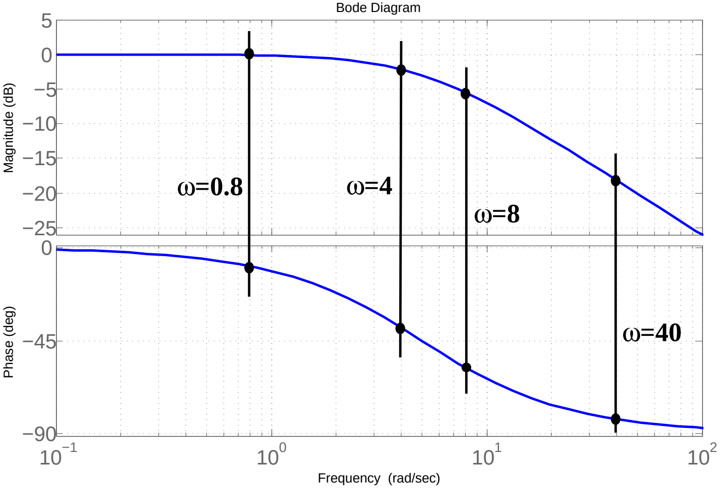




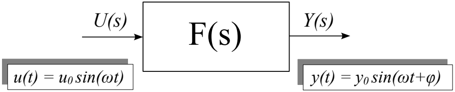


Responses of the circuit with $\omega = \{0.8, 4, 8, 40\}$

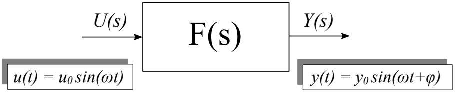


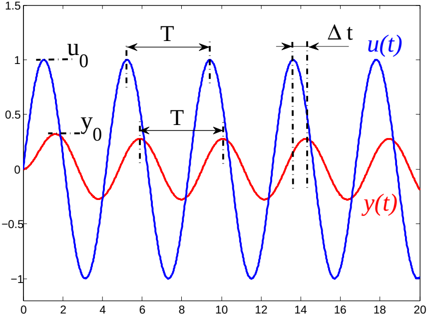

Bode Diagram
Magnitude (dB)
5
0
−5
−10
−15
−20
−25
ω=0.8
ω=4
ω=8
ω=40
Phase (deg)
0
−45
−90
10^{-1}
10^{0}
10^{1}
10^{2}
Frequency (rad/sec)

Frequency analysis consists in studying the response of a LTI system with sine inputs



Frequency analysis consists in studying the response of a LTI system with sine inputs





The output signal is also a sine with the same frequency, but with a different magnitude and a different phase angle.

A system can then be characterized by its

- gain : $\dfrac{y_0}{u_0}$
- phase shift : $\pm 360\dfrac{\Delta t}{T}$

The magnitude and the phase depend on the frequency ω

A system can then be characterized by its

- gain : $\frac{y_0}{u_0}$
- phase shift : $\pm 360\frac{\Delta t}{T}$

The magnitude and the phase depend on the frequency ω

It can be shown that :

- gain $= |F(j\omega)|$,
- phase shift $= \arg F(j\omega)$.

$F(j\omega)$ is the transfer function of the system where the Laplace variable s has been replaced by $j\omega$.

Example : let consider system

$$F(s) = \frac{1/2}{s+1}$$

What are the responses to these inputs ?

$$u_1 = \sin(0.05\, t)$$
$$u_2 = \sin(1.5\, t)$$
$$u_3 = \sin(10\, t)$$

Example : let consider system

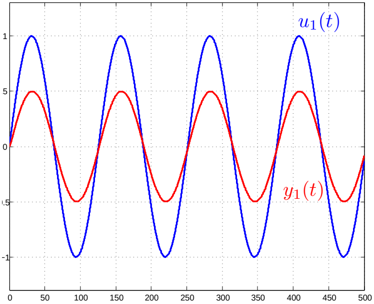
$$F(s) = \frac{1/2}{s+1}$$

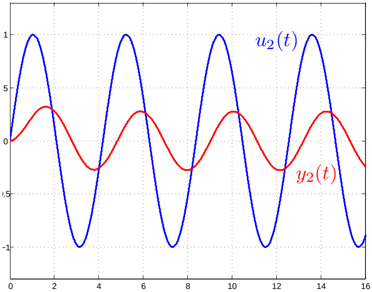
What are the responses to these inputs ?

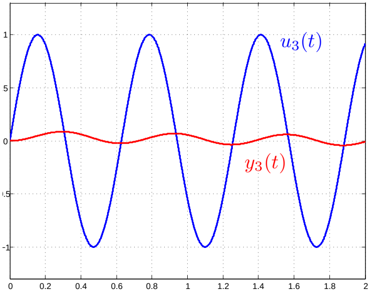
$$u_1 = \sin(0.05\,t)$$
$$u_2 = \sin(1.5\,t)$$
$$u_3 = \sin(10\,t)$$

we express $F(j\omega) = \dfrac{1/2}{j\omega + 1}$

- for $\omega = 0.05\ rad/s$: $|F(j0.05)| = 0.5$ and $\arg F(j0.05) = -2.86°$.
- for $\omega = 1.5\ rad/s$: $|F(j1.5)| = 0.277$ and $\arg F(j1.5) = -56.3°$.
- for $\omega = 10\ rad/s$: $|F(j10)| = 0.05$ and $\arg F(j10) = -84.3°$.


$u_1(t)$
$y_1(t)$
1
5
0
.5
-1
0 50 100 150 200 250 300 350 400 450 500


$u_2(t)$
$y_2(t)$
1
5
0
.5
-1
0 2 4 6 8 10 12 14 16

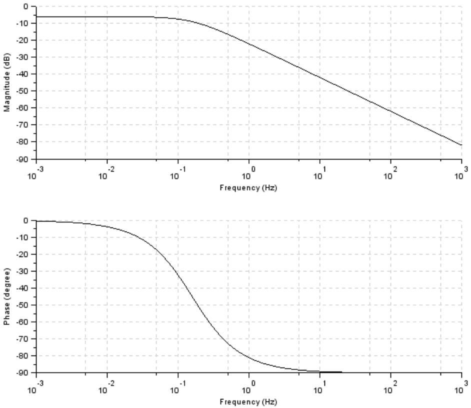

$u_3(t)$
$y_3(t)$
1
5
0
.5
-1
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2

Bode diagram : it plots the gain and the phase shift w.r.t. the frequency ω

- the gain is expressed as decibels : gain dB $= 20\log\frac{y_0}{u_0}$
- property : the Bode diagram of $F(s)G(s)$ is the sum of the one of $F(s)$ and the one of $G(s)$.
- in Scilab, the instruction `bode(F)` plots the Bode diagram of $F(s)$.

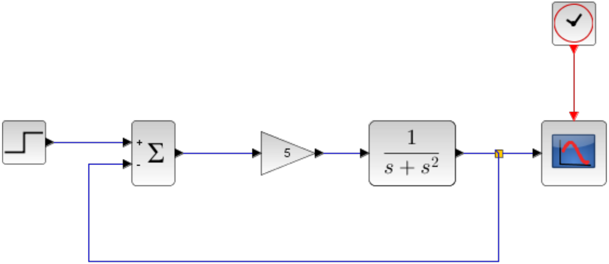
Bode diagram : it plots the gain and the phase shift w.r.t. the frequency ω

- the gain is expressed as decibels : gain dB $= 20 \log \frac{y_0}{u_0}$
- property : the Bode diagram of $F(s)G(s)$ is the sum of the one of $F(s)$ and the one of $G(s)$.
- in Scilab, the instruction `bode(F)` plots the Bode diagram of $F(s)$.



Simulation with Xcos

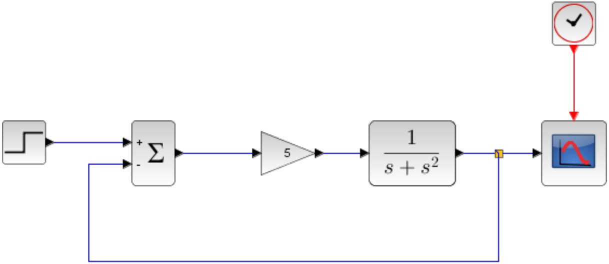
Let simulate the closed-loop control with a proportional gain



...

Simulation with Xcos

Let simulate the closed-loop control with a proportional gain



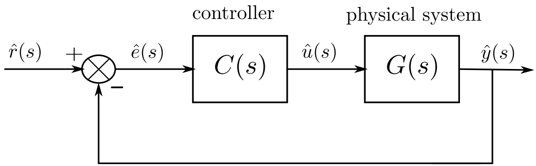
block	sub-palette
step	Sources/STEP_FUNCTION
sum	Math. Operations/BIGSOM_f
gain	Math. Operations/GAINBLK_f
transfert function	Cont. time systems/CLR
scope	Sinks/CSCOPE
clock	Sources/CLOCK_c

settings : final value (step) = $\%pi/2$, final integral time = 12, Ymin= 0, Ymax= 2.5, Refresh period = 12

Sommaire

Classical control design

Control design aims at designing a controller $C(s)$ in order to assign desired performances to the closed loop system



- Classical control is a frequency domain approach and is essentially based on Bode plot
- Main controllers, or compensators, are phase lag, phase lead, PID (proportional integral derivative)

Loopshaping

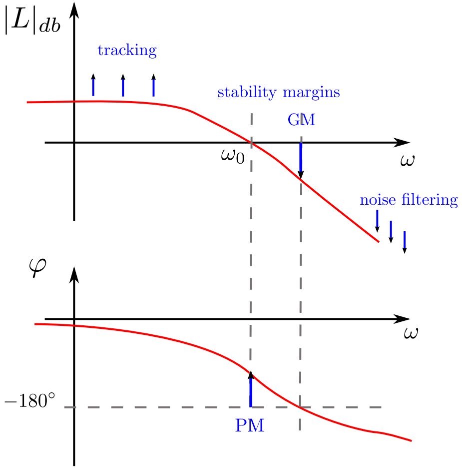
Let express the tracking error

$$\hat{e}(s) = \frac{1}{1 + G(s)C(s)}\hat{r}(s)$$

So, a high open-loop gain results in a good tracking

- it leads to better accuracy and faster response (depending on the bandwidth)
- but it leads to a more aggressive control input (u)
- but it reduces stability margins

Let define the open-loop transfer function $L = GC$
Closed-loop performances can be assessed from the Bode plot of L



- PM and GM are phase and gain margins
- noise disturbances are a high frequency signals

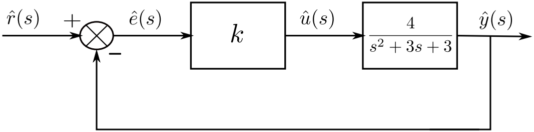
Loopshaping consists in designing the controller $C(s)$ so as to "shape" the frequency response of $L(s)$

we recall that

$$|L|_{db} = |GC|_{db} = |G|_{db} + |C|_{db}$$

The desired "shape" depends on performance requirements for the closed-loop system

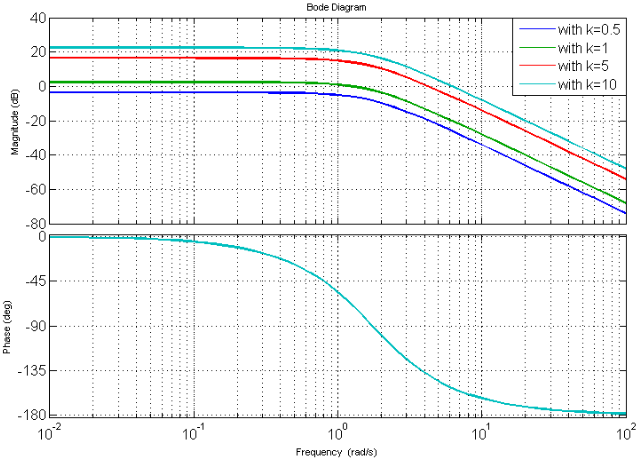
A simple example with a proportional controller



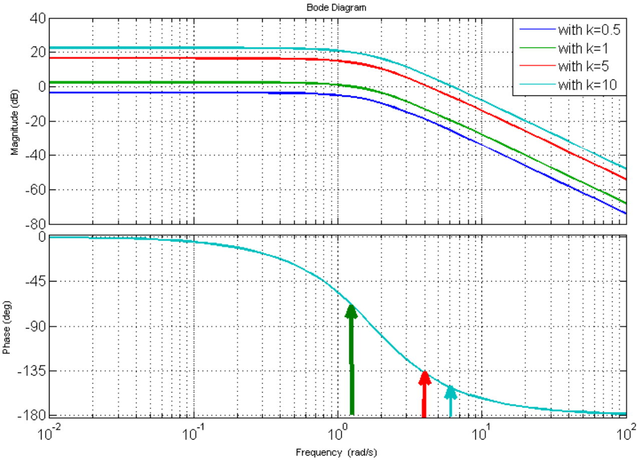
The open-loop transfer function is

$$L(s) = \frac{4k}{s^2 + 3s + 3}$$

Bode plot of $L(s)$ for $k = \{0.5, 1, 5, 10\}$

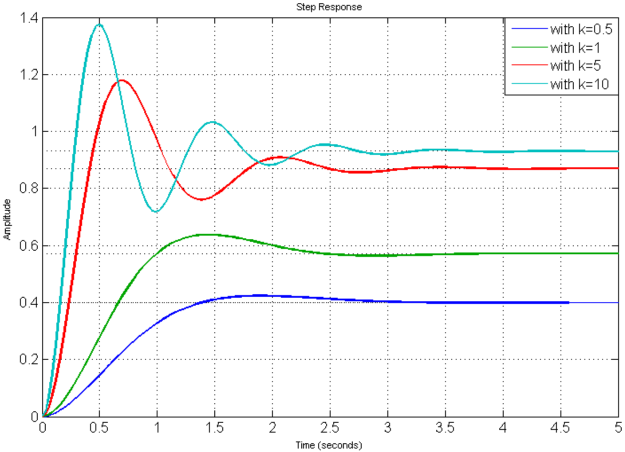


Bode plot of $L(s)$ for $k = \{0.5, 1, 5, 10\}$



when k increases, the phase margin decreases

Step response of the closed-loop system (unit step) for $k = \{0.5, 1, 5, 10\}$



- the static error decreases as k increases
- oscillations increase as k increases

Phase lag controller

The transfer function of the phase lag controller is of the form

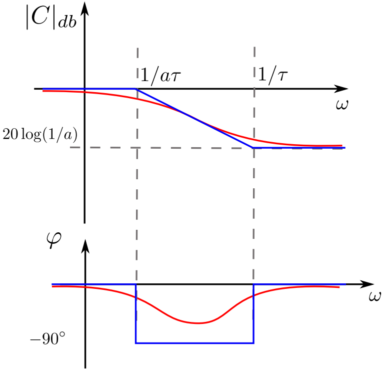
$$C(s) = \frac{1 + \tau s}{1 + a\tau s}, \qquad \text{with } a > 1$$

Phase lag controller

The transfer function of the phase lag controller is of the form

$$C(s) = \frac{1+\tau s}{1+a\tau s}, \qquad \text{with } a > 1$$

- a and τ are tuning parameters
- It allows a higher gain in low frequencies
- But the phase lag must not reduce the phase margin



Example

$$G(s) = \frac{4}{s^2 + 3s + 3}$$

What value for the proportional gain k to have a static error of 10% ?

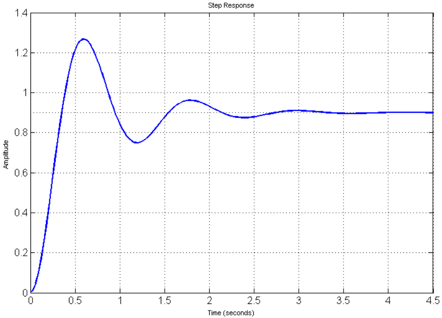
Example

$$G(s) = \frac{4}{s^2 + 3s + 3}$$

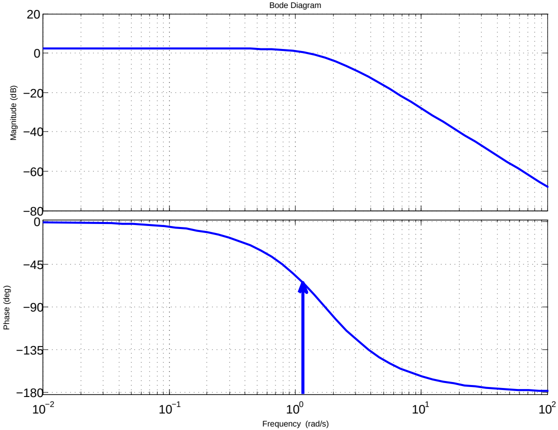
What value for the proportional gain k to have a static error of 10% ?

$$\text{static error } = \frac{1}{1 + \frac{4}{3}k} = 0.1 \qquad \Rightarrow \qquad k = 6.75$$

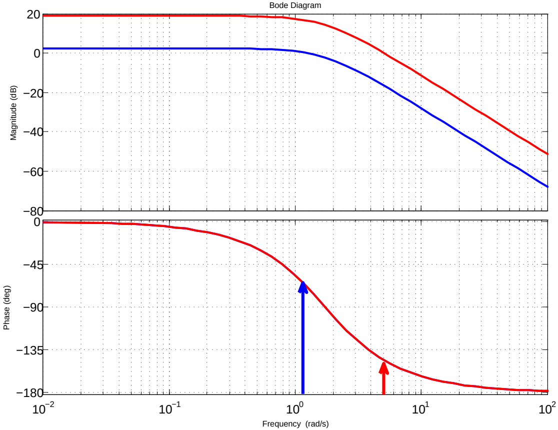
close-loop system response



Precision ok, but too much oscillations



Precision ok, but too much oscillations

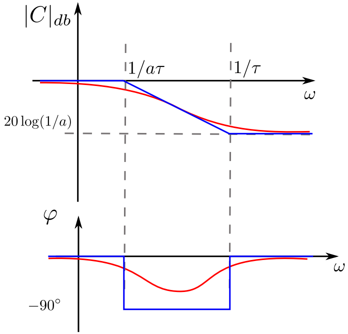


Phase margin : before $= 111^\circ$ (at $1.24\ rd/s$) ; after $= 34^\circ$ (at $5.04\ rd/s$)

Phase lag controller

$$C(s) = \frac{1 + \tau s}{1 + a\tau s}$$

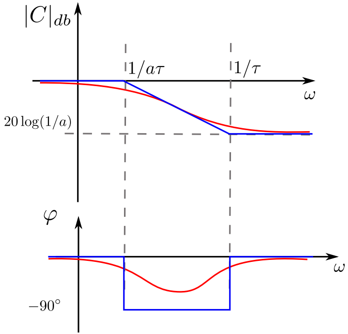
with $a > 1$



Phase lag controller

$$C(s) = \frac{1 + \tau s}{1 + a\tau s}$$

with $a > 1$

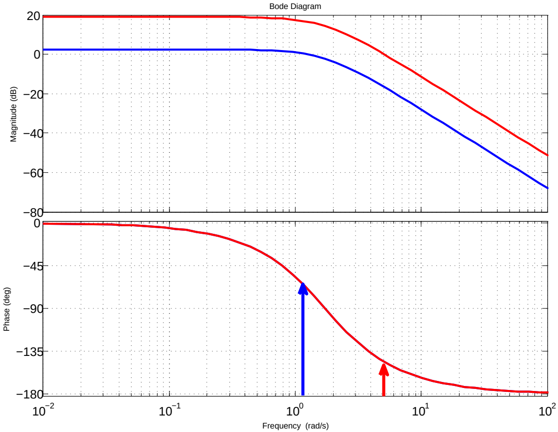


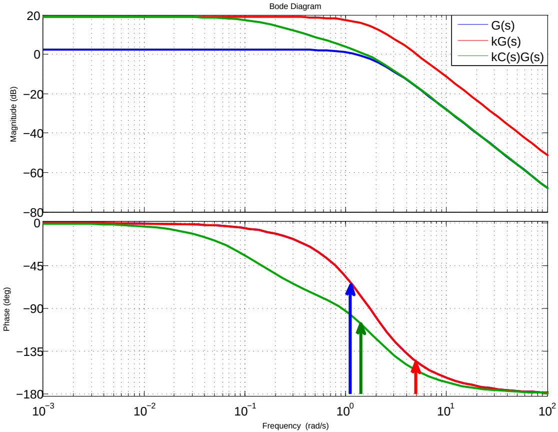
- We want a high gain only at low frequencies
- Phase lag must occur before the crossover frequency

$$\frac{1}{\tau} < \omega_0 = 1.24 \qquad \Rightarrow \qquad \tau = 1$$

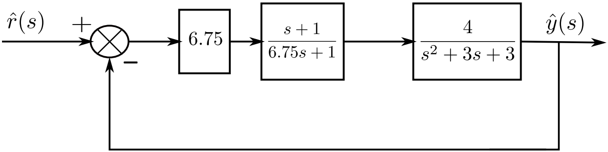
- Then, we want to recover a gain of 1

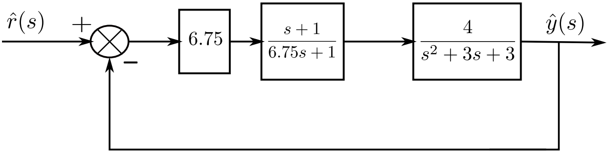
$$a = 6.75$$


Bode Diagram
Magnitude (dB)
20
0
−20
−40
−60
−80
Phase (deg)
0
−45
−90
−135
−180
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Frequency (rad/s)

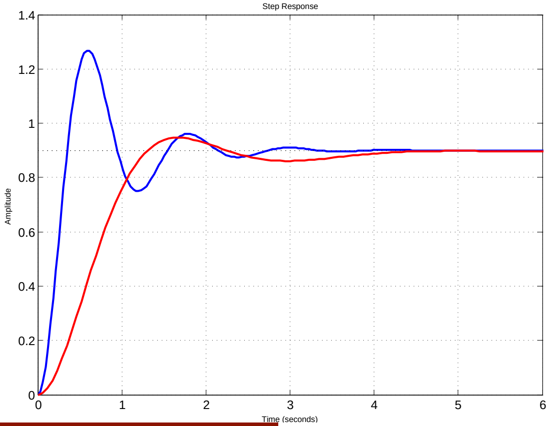


Phase margin : now, with the proportional gain and the phase lag controller $= 70^\circ$ (at 1.56 rd/s)


$\hat{r}(s)$
+
−
6.75
$\frac{s+1}{6.75s+1}$
$\frac{4}{s^2+3s+3}$
$\hat{y}(s)$



close-loop system response



Phase lead controller

The transfer function of the phase lead controller is of the form

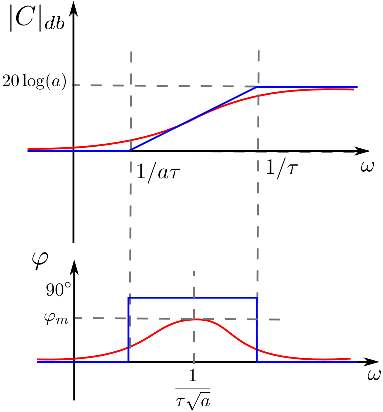
$$C(s) = \frac{1 + a\tau s}{1 + \tau s}, \qquad \text{with } a > 1$$

Phase lead controller

The transfer function of the phase lead controller is of the form

$$C(s) = \frac{1 + a\tau s}{1 + \tau s}, \qquad \text{with } a > 1$$

- a and τ are tuning parameters
- It provides a phase lead in a frequency range
- But the gain may shift the crossover frequency



The phase lead compensator is used to increase the phase margin

Procedure :

- firstly, adjust a proportional gain k to reach a tradeoff between speed/accuracy and overshoot.
- measure the current phase margin and subtract to the desired margin

$$\varphi_m = PM_{\text{desired}} - PM_{\text{current}}$$

- compute a

$$a = \frac{1 + \sin \varphi_m}{1 - \sin \varphi_m}$$

- at the maximum phase lead φ_m, the magnitude is $20 \log \sqrt{a}$. Find the frequency ω_m for which the magnitude of $kG(s)$ is $-20 \log \sqrt{a}$
- compute τ

$$\tau = \frac{1}{\omega_m \sqrt{a}}$$

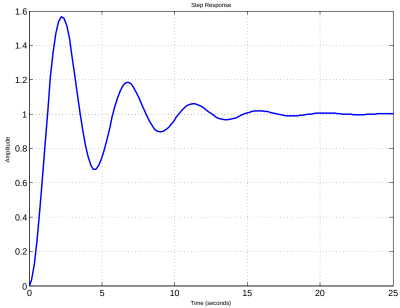
Example

$$G(s) = \frac{4}{s(2s+1)}$$

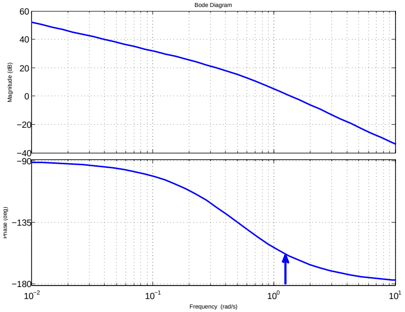
Example

$$G(s) = \frac{4}{s(2s+1)}$$

close-loop system response



open-loop bode diagram



Phase margin : 20° at 1.37 rd/s

Design of a phase lead compensator

- current phase margin is 20°, and the desired margin is, say, 60°

$$\varphi_m = 40^\circ = 0.70 \ rd$$

- compute a

$$a = \frac{1 + \sin \varphi_m}{1 - \sin \varphi_m} = 4.62$$

- at the maximum phase lead φ_m, the magnitude is $6.65 \ db$. At the frequency $\sim 2 \ rd/s$ the magnitude of $G(s)$ is $-6.65 \ db$

- compute τ

$$\tau = \frac{1}{\omega_m \sqrt{a}} = 0.23$$

Design of a phase lead compensator

- current phase margin is 20°, and the desired margin is, say, 60°

$$\varphi_m = 40^\circ = 0.70\ rd$$

- compute a

$$a = \frac{1 + \sin\varphi_m}{1 - \sin\varphi_m} = 4.62$$

- at the maximum phase lead φ_m, the magnitude is 6.65 db. At the frequency $\sim 2\ rd/s$ the magnitude of $G(s)$ is $-6.65\ db$

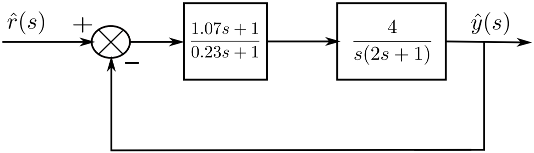
- compute τ

$$\tau = \frac{1}{\omega_m \sqrt{a}} = 0.23$$

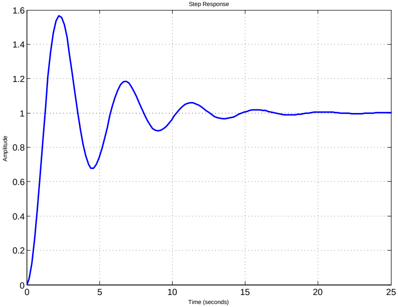
Hence, the controller is of the form

$$C(s) = \frac{1 + 1.07s}{1 + 0.23s}$$

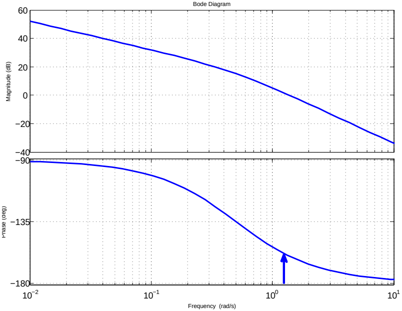
Example



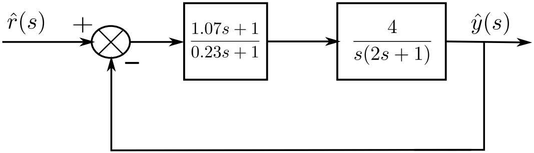
close-loop system response



open-loop bode diagram

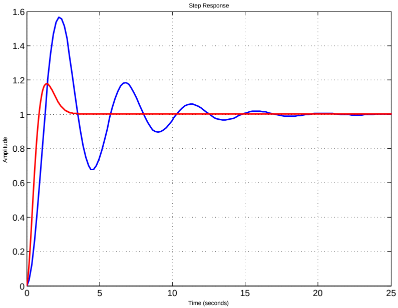


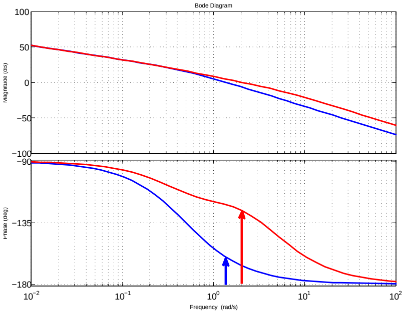
Example



close-loop system response

open-loop bode diagram





New phase margin : $53.7°$ at $2\ rd/s$

PID controller

A PID controller consists in 3 control actions

$\Rightarrow$ proportional, integral and derivative

Transfer function of the form :

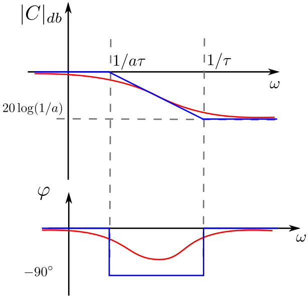
$$
\begin{aligned}
C(s) &= k_p + k_i \tfrac{1}{s} + k_d s \\
&= k_p(1 + \tfrac{1}{\tau_i s})(1 + \tau_d s)
\end{aligned}
$$

The phase lag controller is an approximation of the PI controller

Phase lag controller

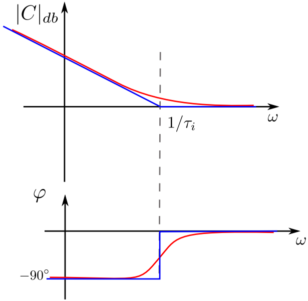
$$C(s) = \frac{1 + \tau s}{1 + a\tau s}$$

with $a > 1$



PI controller

$$C(s) = \frac{1 + \tau_i s}{\tau_i s}$$

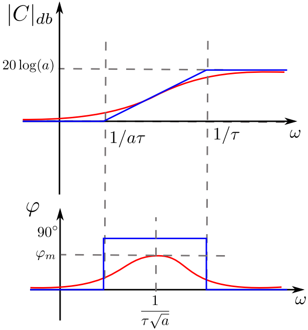


The phase lead controller is an approximation of the PD controller

Phase lead controller

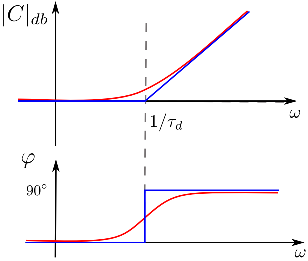
$$C(s) = \frac{1 + a\tau s}{1 + \tau s}$$

with $a > 1$



PD controller

$$C(s) = 1 + \tau_d s$$



A PID controller is a combination of phase lag and phase lead controllers

$$C(s) = k\left(\frac{1+\tau_1 s}{1+a_1\tau_1 s}\right)\left(\frac{1+a_2\tau_2 s}{1+\tau_2 s}\right)$$

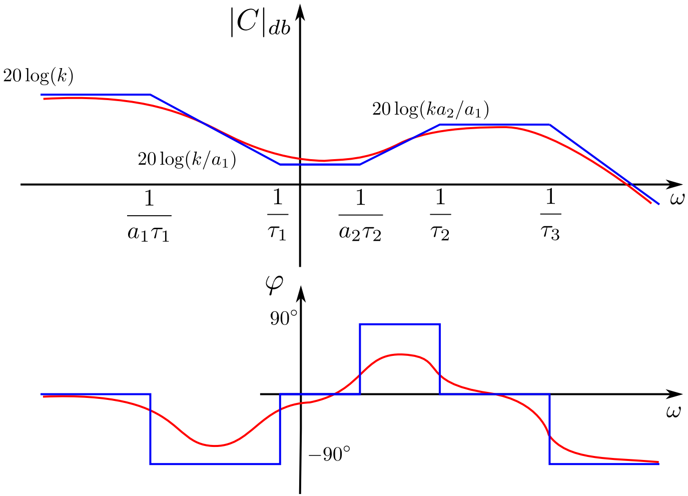
with $a_1 > 1$ and $a_2 > 1$.

Transfer function of the form :

- the phase lag part is designed to improve accuracy and responsiveness
- the phase lead part is designed to improve stability margins
- an extra low-pass filter may be added to reduce noise

$$C_1(s) = \frac{1}{1+\tau_3 s}$$

with $\tau_3 \ll \tau_2$


$|C|_{db}$
$20\log(k)$
$20\log(ka_2/a_1)$
$20\log(k/a_1)$
ω
$\frac{1}{a_1\tau_1}$
$\frac{1}{\tau_1}$
$\frac{1}{a_2\tau_2}$
$\frac{1}{\tau_2}$
$\frac{1}{\tau_3}$
φ
$90°$
$-90°$
ω