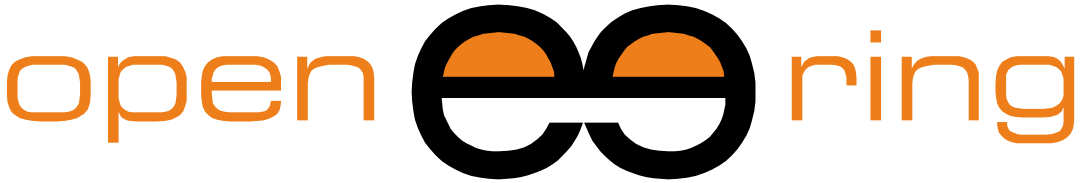
OPEN SOURCE ENGINEERING



A SCILAB PROFESSIONAL PARTNER

powered by ENGIN SOFT

INTRODUCTION TO CONTROL SYSTEMS IN SCILAB

In this Scilab tutorial, we introduce readers to the Control System Toolbox that is available in Scilab/Xcos and known as CACSD. This first tutorial is dedicated to "Linear Time Invariant" (LTI) systems and their representations in Scilab.

Level

www.openeering.com

Step 1: LTI systems

Linear Time Invariant (LTI) systems are a particular class of systems characterized by the following features:

- ***Linearity***: which means that there is a linear relation between the input and the output of the system. For example, if we scale and sum all the input (linear combination) then the output are scaled and summed in the same manner. More formally, denoting with $x_k(t)$ a generic input and with $y_k(t)$ a generic output we have:

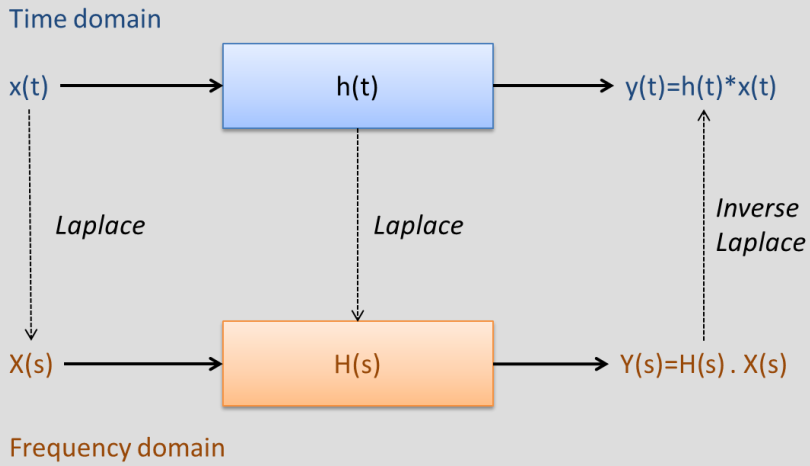
$$\sum_k c_k x_k(t) \xrightarrow{yields} \sum_k c_k y_k(t)$$

- ***Time invariance***: which mean that the system is invariant for time translations. Hence, the output $y_k(t)$ produced by the input $x_k(t)$ is identical to the output $y_k(t-T)$ produced by the input $x_k(t-T)$ which is shifted by the quantity T.

Thanks to these properties, in the ***time domain***, we have that any LTI system can be characterized entirely by a single function which is the response to the ***system's impulse***. The system's output is the convolution of the input with the system's impulse response.

In the ***frequency domain***, the system is characterized by the ***transfer function*** which is the Laplace transform of the system's impulse response.

The same results are true for ***discrete time linear shift-invariant systems*** which signals are discrete-time samples.



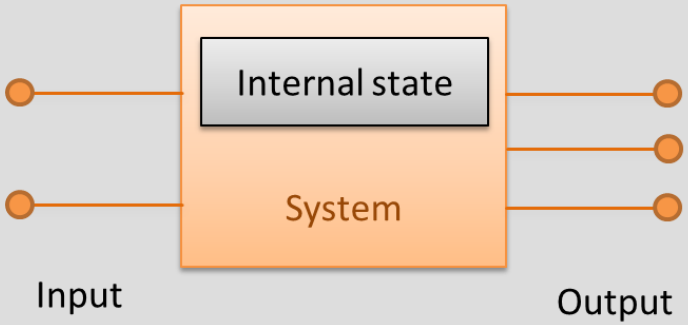
Step 2: LTI representations

LTI systems can be classified into the following two major groups:

- ***SISO***: Single Input Single Output;
- ***MIMO***: Multiple Input Multiple Output.

LTI systems have several ***representation forms***:

- Set of differential equations in the ***state space*** representation;
- ***Transfer function*** representation;
- ***Zero-pole*** representation.



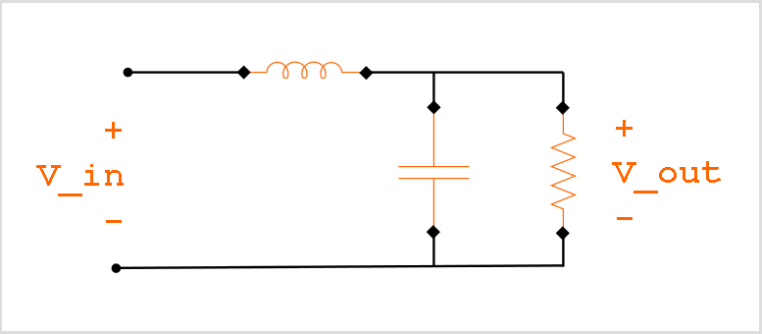
A typical representation of a system with its input and output ports and its internal state

Step 3: RLC example

This RLC example is used to compare all the LTI representations. The example refers to a ***RLC low passive filter***, where the input is represented by the voltage drop "`V_in`" while the output "`V_out`" is voltage across the resistor.

In our examples we choose:

- Input signal: $V_{in}(t) = A \cdot \sin(2\pi f \cdot t)$;
- Resistor: $V_R(t) = R \cdot I_R(t)$;
- Inductor: $V_L(t) = L \cdot \frac{dI_L(t)}{dt}$;
- Capacitor: $I_C(t) = C \cdot \frac{dV_C(t)}{dt}$.



(Example scheme)

Step 4: Analytical solution of the RLC example

The relation between the input and the output of the system is:

$$
\begin{aligned}
V_{in}(\mathrm{t}) &= V_L(\mathrm{t}) + V_C(\mathrm{t}) \\
&= L \cdot \frac{dI_L(t)}{dt} + V_C(\mathrm{t}) \\
&= L \cdot \frac{d}{dt}\left(I_C(t) + I_R(t)\right) + V_C(\mathrm{t}) \\
&= L \cdot \frac{dI_C(t)}{dt} + L \cdot \frac{d}{dt}\left(\frac{V_R(\mathrm{t})}{R}\right) + V_C(\mathrm{t}) \\
&= L \cdot \frac{d}{dt}\left(C \cdot \frac{dV_C(\mathrm{t})}{dt}\right) + \frac{L}{R}\frac{dV_C(t)}{dt} + V_C(\mathrm{t}) \\
&= V_C(\mathrm{t}) + \frac{L}{R}\frac{dV_C(t)}{dt} + LC \cdot \frac{d^2V_C(t)}{dt^2} \\
&= V_{out}(\mathrm{t}) + \frac{L}{R}\frac{dV_{out}(t)}{dt} + LC \cdot \frac{d^2V_{out}(t)}{dt^2}
\end{aligned}
$$

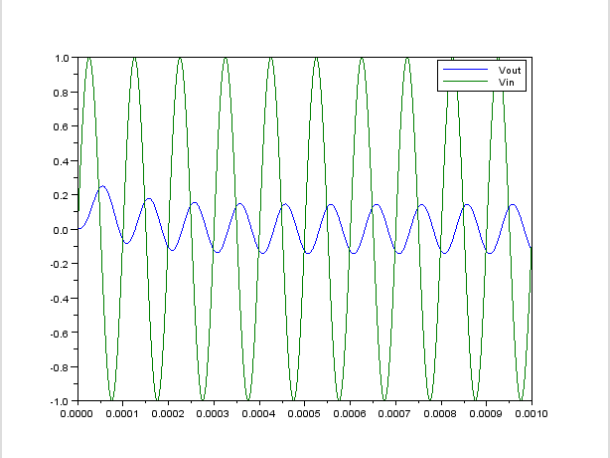
On the right we report a plot of the solution for the following values of the constants:

- $A = 1.0$ [V];
- $f = 10^4$ [Hz];
- $R = 10$ [Ohm];
- $L = 10^{-3}$ [H];
- $C = 10^{-6}$ [F];

with initial conditions: $V_{out}(\mathrm{t}) = V'_{out}(\mathrm{t}) = 0$.

```
// Problem data
A = 1.0; f = 1e+4;
R = 10;       // Resistor  [Ohm]
L = 1e-3;     // Inductor  [H]
C = 1e-6;     // Capacitor [F]

// Problem  function
function zdot=RLCsystem(t, y)
    z1 = y(1); z2 = y(2);
    // Compute input
    Vin = A*sin(2*%pi*f*t);
    zdot(1) = z2; zdot(2) = (Vin - z1 - L*z2/R) /(L*C);
endfunction

// Simulation time [1 ms]
t = linspace(0,1e-3,1001);

// Initial conditions and solving the ode system
y0 = [0;0]; t0 = t(1);
y = ode(y0,t0,t,RLCsystem);

// Plotting results
Vin = A*sin(2*%pi*f*t)';
scf(1); clf(1); plot(t,[Vin,y(1,:)']); legend(["Vin";"Vout"]);
```

(Numerical solution code)

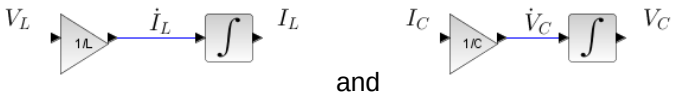


(Simulation results)

Step 5: Xcos diagram of the RLC circuit

There can be many Xcos block formulation for the RLC circuit but the one which allows fast and accurate results is the one that ***uses only integration blocks*** instead of derivate blocks.

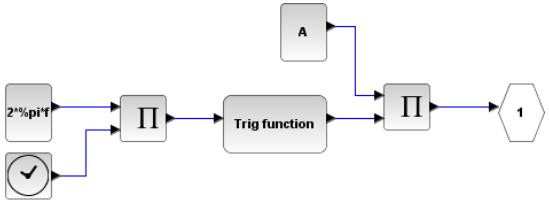
The idea is to start assembling the differential part of the diagram as:



and

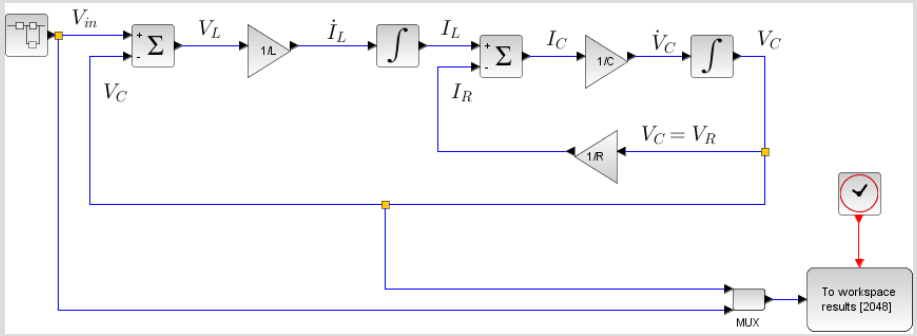
and then to complete the scheme taking into consideration the relations (Kirchhoff's laws) $V_L = V_{in} - V_C$ and $I_C = I_L - I_R$ with $I_R = V_C/R$.

At the end, we add the model for V_{in} as follows:

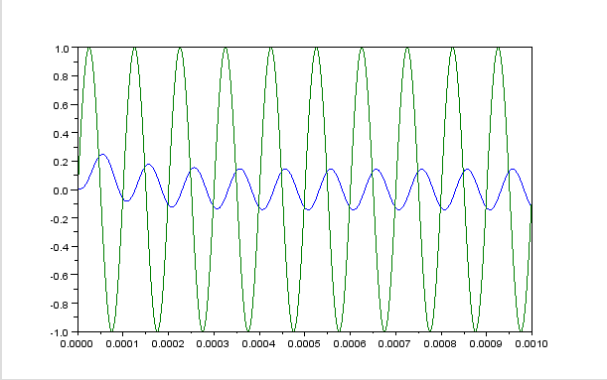


The simulation results are stored in the Scilab mlist variable "`results`" and plotted using the command

```
// Plotting data
plot(results.time,results.values)
```



(Simulation diagram)



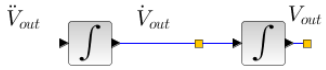
(Simulation results)

Step 6: Another Xcos diagram of the RLC circuit

On the right we report another Xcos representation of the system which is obtained starting from the system differential equation:

$$V_{in}(\mathrm{t}) = V_{out}(\mathrm{t}) + \frac{L}{R}\frac{dV_{out}(t)}{dt} + LC \cdot \frac{d^2V_{out}(t)}{dt^2}$$

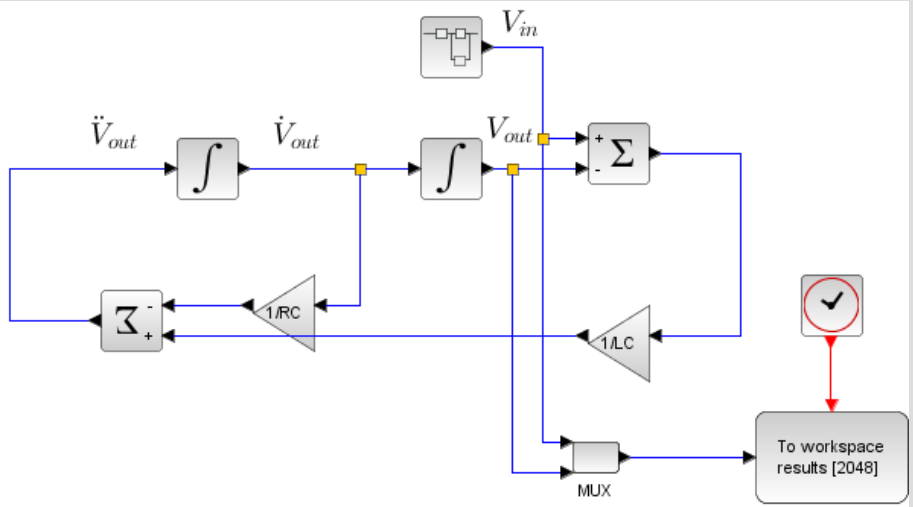
As previously done, the scheme is obtained starting from



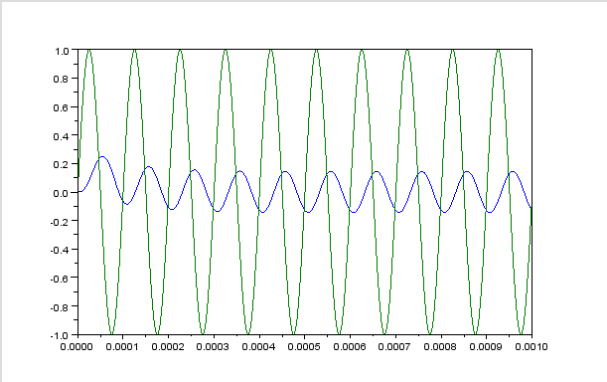
and using the relation

$$\frac{d^2V_{out}(t)}{dt^2} = \frac{\left(V_{in}(\mathrm{t}) - V_{out}(\mathrm{t})\right)}{LC} - \frac{1}{RC}\frac{dV_{out}(t)}{dt}$$

which relates the second derivative of the $V_{out}(t)$ to the other variables.



(Simulation diagram)



(Simulation results)

Step 7: State space representation

The state space representation of any LTI system can be stated as follows:

$$\begin{aligned} \dot{\boldsymbol{x}}(\boldsymbol{t}) &= \boldsymbol{A}\boldsymbol{x}(\boldsymbol{t}) + \boldsymbol{B}\boldsymbol{u}(\boldsymbol{t}) \\ \boldsymbol{y}(\boldsymbol{t}) &= \boldsymbol{C}\boldsymbol{x}(\boldsymbol{t}) + \boldsymbol{D}\boldsymbol{u}(\boldsymbol{t}) \end{aligned}$$

where x is the state vector (a collection of all internal variables that are used to describe the dynamic of the system) of dimension n, y is the output vector of dimension m associated to observation and u is the input vector of dimension r .

Here, the first equation represents the state updating equations while the second one relates the system output to the state variables.

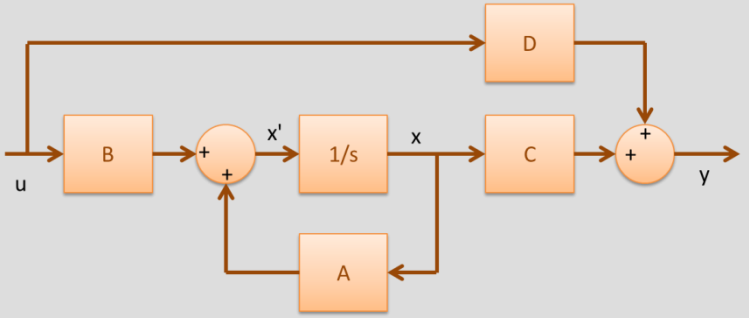
In many engineering problem the matrix D is the null matrix, and hence the output equation reduces to $y = Cx$, which is a weight combination of the state variables.

A space-state representation in term of block is reported on the right. Note that the representation requires the choice of the state variable. This choice is not trivial since there are many possibilities. The number of state variables is generally equal to the order of the system's differential equations. In electrical circuit, a typical choice consists of picking all variables associated to differential elements (capacitor and inductor).

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_{1,1} & \cdots & b_{1,r} \\ b_{2,1} & \cdots & b_{2,r} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,r} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_r \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m,1} & c_{m,2} & \cdots & c_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} d_{1,1} & \cdots & d_{1,r} \\ d_{2,1} & \cdots & d_{2,r} \\ \vdots & \ddots & \vdots \\ d_{m,1} & \cdots & d_{m,r} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_r \end{bmatrix}$$

State space representation



Block diagram representation of the state space equations

Step 8: State space representation of the RLC circuit

In order to write the space state representation of the RLC circuit we perform the following steps:

- Choose the modeling variable: Here we use $x = (I_L(t), V_C(t))$, $u = (V_{in}(t))$ and $y = (V_{out}(t))$;
- Write the state update equation in the form $\dot{x}(t) = Ax(t) + Bu(t)$;

  For the current in the inductor we have:

  $$\dot{I}_L(t) = \frac{1}{L} V_L(t) = \frac{1}{L}(V_{in}(t) - V_C(t)) = -\frac{1}{L}V_C(t) + \frac{1}{L} V_{in}(t)$$

  For the voltage across the capacitor we have:

  $$\dot{V}_C(t) = \frac{1}{C} I_C(t) = \frac{1}{C}\left(I_L(t) - I_R(t)\right) = \frac{1}{C} I_L(t) - \frac{1}{C}\frac{V_R(t)}{R}$$

  $$= \frac{1}{C} I_L(t) - \frac{1}{RC} V_C(t)$$

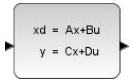
  $$\begin{bmatrix} \dot{I}_L(t) \\ \dot{V}_C(t) \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} \cdot \begin{bmatrix} I_L(t) \\ V_C(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} \cdot [V_{in}(t)]$$

- Write the observer equation in the form $y(t) = Cx(t) + Du(t)$.

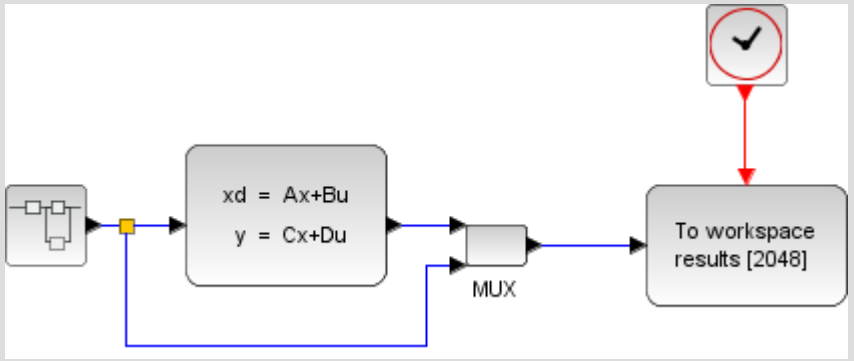
  The output voltage V_{out} is equal to the voltage of the capacitor $v_C(t)$. Hence the equation can be written as

  $$[V_{out}(t)] = [0 \quad 1] \cdot \begin{bmatrix} I_L(t) \\ V_C(t) \end{bmatrix} + [0] \cdot [V_{in}(t)]$$

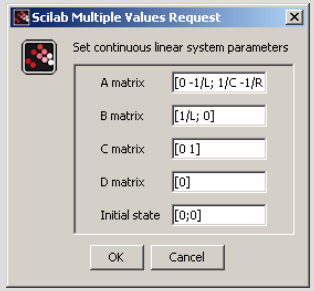
The diagram representation is reported on the right using the Xcos block:



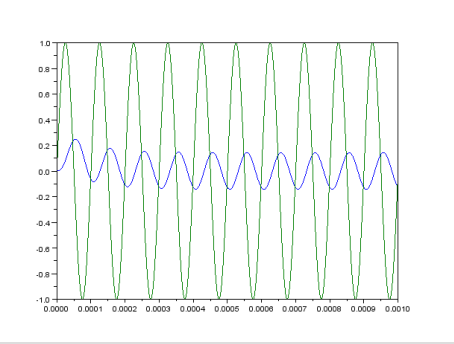
which can directly manage the matrices "A", "B", "C" and "D".



(Simulation diagram)



(Input mask)



(Simulation results)

Step 9: Transfer function representation

In a LTI ***SISO system***, a transfer function is a mathematical relation between the input and the output in the Laplace domain considering its initial conditions and equilibrium point to be zero.

For example, starting from the differential equation of the RLC example,

$$V_{in}(\mathrm{t}) = V_{out}(\mathrm{t}) + \frac{L}{R}\frac{dV_{out}(t)}{dt} + LC \cdot \frac{d^2V_{out}(t)}{dt^2}$$

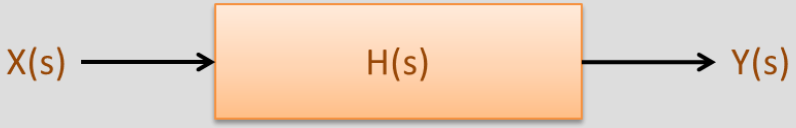
the transfer function is obtained as follows:

$$\begin{aligned} V_{in}(s) &= = V_{out}(s) + \frac{L}{R}s \cdot V_{out}(s) + LC \cdot s^2 \cdot V_{out}(s) \\ &= \left(1 + \frac{L}{R}s + LC \cdot s^2\right) \cdot V_{out}(s) \end{aligned}$$

that is:

$$\frac{\boldsymbol{V_{out}(s)}}{\boldsymbol{V_{in}(s)}} = \frac{\mathbf{1}}{\left(\mathbf{1} + \frac{\boldsymbol{L}}{\boldsymbol{R}}\boldsymbol{s} + \boldsymbol{LC} \cdot \boldsymbol{s^2}\right)}$$

In the case of ***MIMO systems*** we don't have a single polynomial transfer function but a matrix of transfer functions where each entry is the transfer function relationship between each individual input and each individual output.

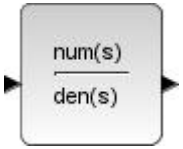


Examples of Laplace transformations:

Time domain	Laplace domain
$y(t)$	$Y(s)$
$\frac{y(\mathrm{t})}{dt}$	$s \cdot Y(s)$
$\int y(\mathrm{t})\mathrm{dt}$	$\frac{1}{s} \cdot Y(s)$

Step 10: Transfer function representation of the RLC circuit

The diagram representation is reported on the right. Here we use the Xcos block:



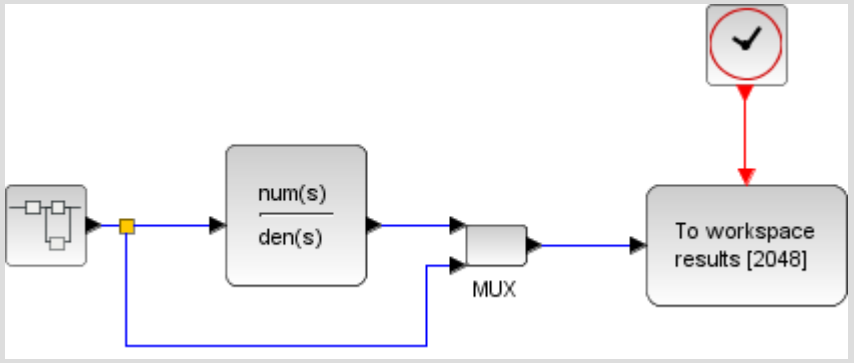
which the user can specify the numerator and denominator of the transfer functions in term of the variable "s".

The transfer function is

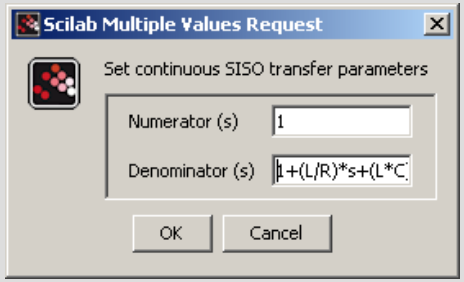
$$\frac{\text{num}(s)}{\text{den}(s)} = \frac{1}{\left(1 + \frac{L}{R}s + LC \cdot s^2\right)}$$

and, hence, we have:

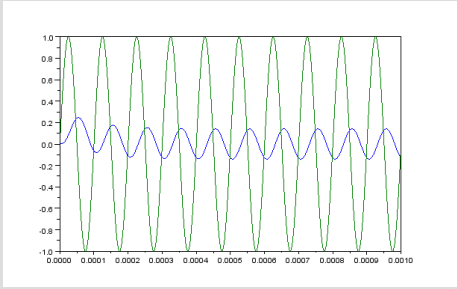
- $\text{num}(s) = 1$
- $\text{den}(s) = \left(1 + \frac{L}{R}s + LC \cdot s^2\right)$



(Simulation diagram)



(Input mask)



(Simulation results)

Step 11: Zero-pole representation and example

Another possible representation is obtained by the use of the partial fraction decomposition reducing the transfer function

$$H(s) = \frac{num(s)}{den(s)}$$

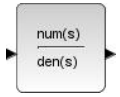
into a function of the form:

$$H(s) = \mathrm{k}\frac{(s-z_1)\cdot(s-z_2)\cdots(s-z_{nz})}{(s-p_1)\cdot(s-p_2)\cdots(s-p_{np})}$$

where k is the gain constant and z_i and p_j are, respectively, the zeros of the numerator and poles of the denominator of the transfer function.

This representation has the advantage to explicit the zeros and poles of the transfer function and so the performance of the dynamic system.

If we want to specify the transfer function in term of this representation in Xcos, we can do that using the block



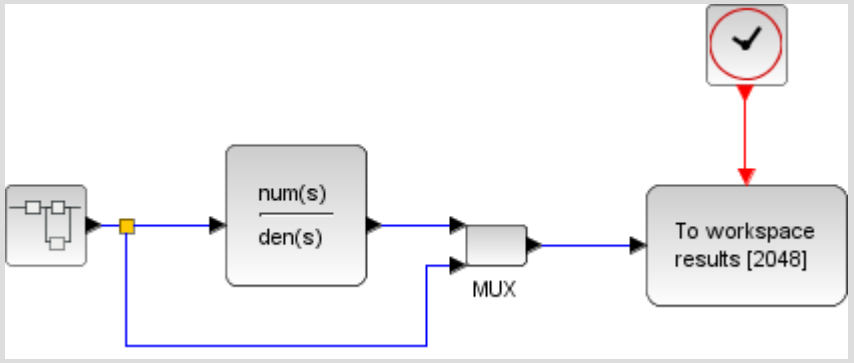
and specifying the numerator and denominator.

In our case, we have

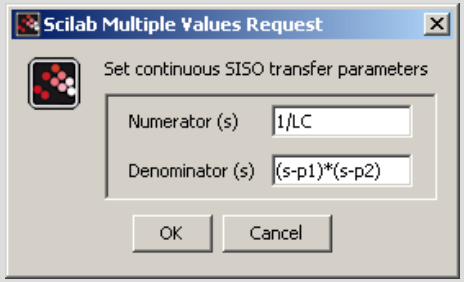
$$\frac{\mathrm{num}(s)}{\mathrm{den}(s)} = \frac{1}{\left(1+\frac{L}{R}s+LC\cdot s^2\right)} = \frac{1/(LC)}{(s-p_1)\cdot(s-p_2)}$$

with

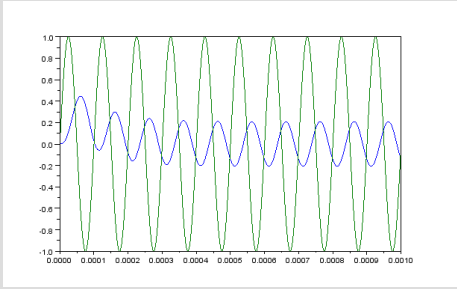
$$p_{1,2} = \frac{-\frac{L}{R} \pm \sqrt{\left(\frac{L}{R}\right)^2 - 4\cdot LC\cdot 1}}{2\cdot LC} = \frac{-L \pm \sqrt{L^2-4R^2LC}}{2RLC}$$



(Simulation diagram)



(Input mask)



(Simulation results)

Step 12: Converting between representations

In the following steps we will see how to change representation in Scilab in an easy way. Before that, it is necessary to review some notes about polynomial representation in Scilab.

A polynomial of degree n is a function of the form:

$$P(s) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Note that in Scilab the order of polynomial coefficient is reversed from that of MATLAB® or Octave®.

The main Scilab commands for managing polynomials are:

- ***%s***: A Scilab variable used to define polynomials;
- ***poly***: to create a polynomial starting from its roots or its coefficients;
- ***coeff***: to extract the coefficient of the polynomial;
- ***horner***: to evaluate a polynomial;
- ***derivat***: to compute the derivate of a polynomial;
- ***roots***: to compute the zeros of the polynomial;
- **+, -, *: standard polynomial operations;
- ***pdiv***: polynomial division;
- **/**: generate a rational polynomial i.e. the division between two polynomials;
- ***inv*** or ***invr***: inversion of (rational) matrix.

```
// Create a polynomial by its roots
p = poly([1 -2],'s')

// Create a polynomial by its coefficient
p = poly([-2 1 1],'s','c')

// Create a polynomial by its coefficient
// Octave/MATLAB(R) style
pcoeff = [1 1 -2];
p = poly(pcoeff($:-1:1),'s','c')
pcoeffs = coeff(p)

// Create a polynomial using the %s variable
s = %s;
p = - 2 + s + s^2

// Another way to create the polynomial
p = (s-1)*(s+2)

// Evaluate a polynomial
res = horner(p,1.0)

// Some operation on polynomial, sum, product and find zeros
q = p+2
r = p*q
rzer = roots(r)

// Symbolic substitution and check
pp = horner(q,p)
res = pp -p-p^2
typeof(res)

// Standard polynomial division
[Q,R] = pdiv(p,q)

// Rational polynomial
prat = p/q
typeof(prat)
prat.num
prat.den

// matrix polynomial and its inversion
r = [1 , 1/s; 0 1]
rinv = inv(r)
```

Step 13: Converting between representations

The state-space representation of our example is:

$$\begin{bmatrix} \dot{I}_L(t) \\ \dot{V}_C(t) \end{bmatrix} = \begin{bmatrix} 0 & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} \cdot \begin{bmatrix} I_L(t) \\ V_C(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} \cdot [V_{in}(t)]$$

$$[V_{out}(t)] = [0 \quad 1] \cdot \begin{bmatrix} I_L(t) \\ V_C(t) \end{bmatrix} + [0] \cdot [V_{in}(t)]$$

while the transfer function is

$$\frac{V_{out}(s)}{V_{in}(s)} = \mathrm{H}(s) = \frac{1}{\left(1 + \frac{L}{R}s + LC \cdot s^2\right)}$$

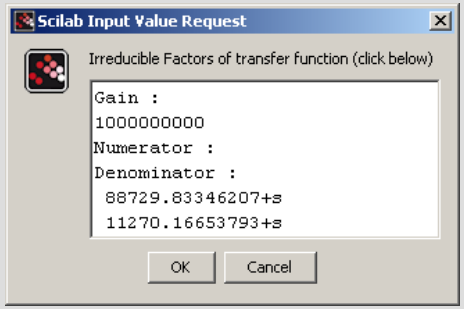
In Scilab it is possible to move from the state-space representation to the transfer function using the command ***ss2tf***. The vice versa is possible using the command ***tf2ss***.

In the reported code (right), we use the "`tf2ss`" function to go back to the previous state but we do not find the original state-space representation. This is due to the fact that the state-space representation is not unique and depends on the adopted change of variables.

The Scilab command ***ss2ss*** transforms the system through the use of a change matrix T, while the command ***canon*** generates a transformation of the system such that the matrix "`A`" is on the Jordan form.

The zeros and the poles of the transfer function can be display using the command ***trfmod***.

```
// RLC low passive filter data
mR = 10;          // Resistor  [Ohm]
mL = 1e-3;          // Inductor [H]
mC = 1e-6;          // Capacitor [F]
mRC = mR*mC;
mf = 1e+4;

// Define system matrix
A = [0 -1/mL; 1/mC -1/mRC];
B = [1/mL; 0];
C = [0 1];
D = [0];

// State space
sl = syslin('c',A,B,C,D)
h = ss2tf(sl)
sl1 = tf2ss(h)

// Transformation
T = [1 0; 1 1];
sl2 = ss2ss(sl,T)

// Canonical form
[Ac,Bc,U,ind]=canon(A,B)

// zero-poles
[hm]=trfmod(h)
```



(Output of the command trfmod)

Step 14: Time response

For a LTI system the output can be computed using the formula:

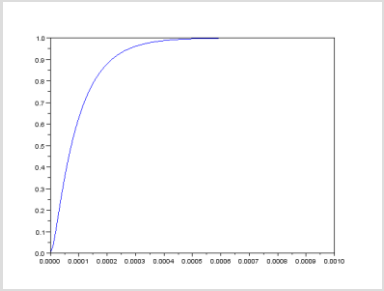
$$\boldsymbol{y(t) = Ce^{At}x_0 + \int_0^t \left(Ce^{A(t-\tau)}B + D\right)u(\tau)d\tau}$$

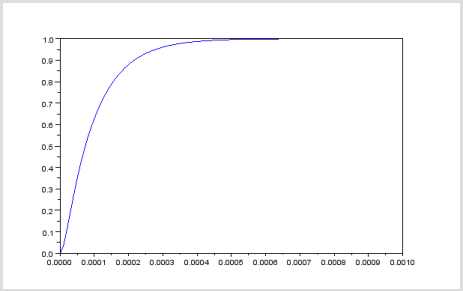
In Scilab this can be done using the command ***csim*** for continuous system while the command ***dsimul*** can be used for discrete systems.

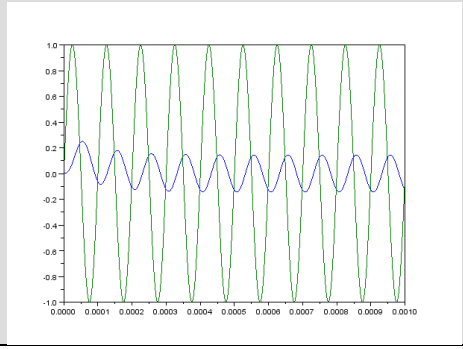
The conversion between continuous and discrete system is done using the command ***dscr*** specifying the discretization time step.

On the right we report some examples: the transfer function is relative to the RLC example.

```
// the step response (continuous system)
t = linspace(0,1e-3,101);
y = csim('step',t,sl);
scf(1); clf(1);
plot(t,y);

// the step response (discrete system)
t = linspace(0,1e-3,101);
u = ones(1,length(t));
dt = t(2)-t(1);
y = dsimul(dscr(h,dt),u);
scf(2); clf(2);
plot(t,y);

// the user define response
t = linspace(0,1e-3,1001);
u = sin(2*%pi*mf*t);
y = csim(u,t,sl);
scf(3); clf(3);
plot(t,[y',u']);
```







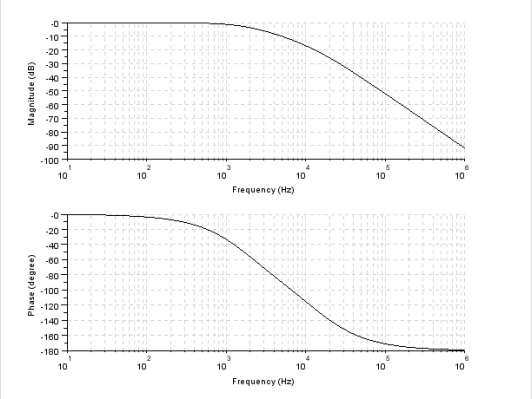
Step 15: Frequency domain response

The Frequency Response is the relation between input and output in the complex domain. This is obtained from the transfer function, by replacing s with $j\omega$.

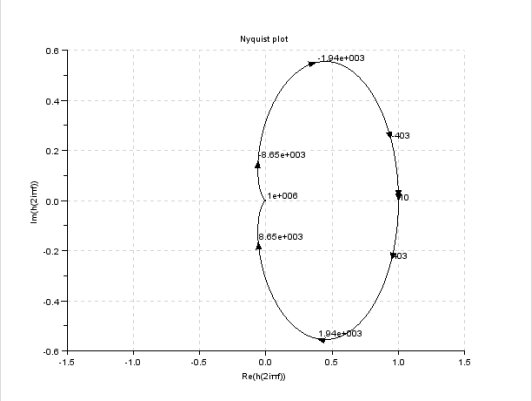
The two main charts are:

- ***Bode diagram***: In Scilab this can be done using the command ***bode***;
- And ***Nyquist diagram***: In Scilab this can be done using the command ***nyquist***.

```
// bode
scf(4); clf(4);
bode(h,1e+1,1e+6);

// Nyquist
scf(5); clf(5);
nyquist(sl,1e+1,1e+6);
```



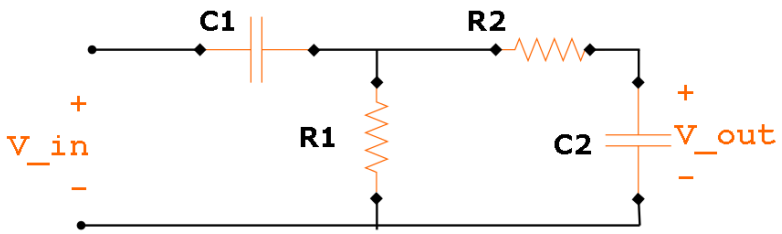
(Bode diagram)



(Nyquist diagram)

Step 16: Exercise

Study in term of time and frequency responses of the following RC band-pass filter:



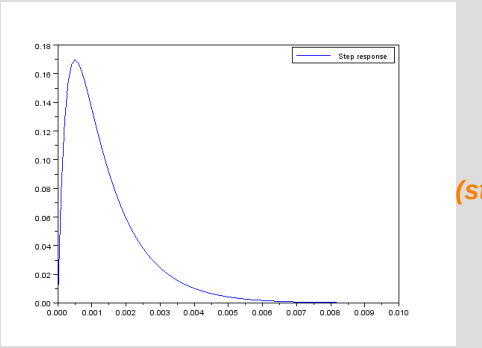
Use the following values for the simulations:

- $R_1 = 100$ [Ohm];
- $C_1 = 3 \cdot 10^{-6}$ [F];
- $R_2 = 1000$ [Ohm];
- $C_2 = 1 \cdot 10^{-6}$ [F].

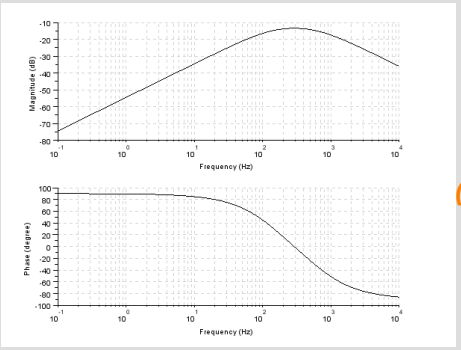
Hints: The transfer function is:

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{\frac{1}{R_2 C_2} s}{\frac{1}{R_1 R_2 C_1 C_2} + \left(\frac{1}{R_1 C_1} + \frac{1}{R_2 C_1} + \frac{1}{R_2 C_2}\right) s + s^2}$$

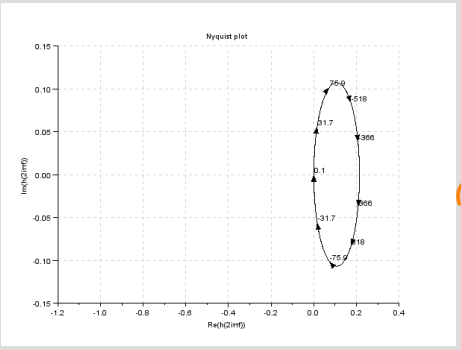
The transfer function can be defined using the command ***syslin.***



(step response)



(Bode diagram)



(Nyquist diagram)

Step 17: Concluding remarks and References

In this tutorial we have presented some modeling approaches in Scilab/Xcos using the Control System Toolbox available in Scilab known as CACSD.

1. Scilab Web Page: Available: www.scilab.org.
2. Openeering: www.openeering.com.

Step 18: Software content

To report bugs or suggest improvements please contact the Openeering team.

www.openeering.com.

Thanks for the bug report: Konrad Kmieciak.

Thank you for your attention,

Manolo Venturin

```
--------------
Main directory
--------------
ex1.sce                       : Exercise 1
numsol.sce                    : Numerical solution of RLC example
poly example.sce              : Polynomial in Scilab
RLC Xcos.xcos                 : RLC in standard Xcos
RLC Xcos ABCD.xcos            : RLC in ABCD formulation
RLC_Xcos_eq.xcos              : Another formulation of RLC in Xcos
RLC_Xcos_tf.xcos              : RLC in transfer function form.
RLC_Xcos_zp.xcos              : RLC in zeros-poles form.
system analysis.sce           : RLC example system analysis
license.txt                   : The license file
```