OPEN SOURCE ENGINEERING

openeering

A SCILAB PROFESSIONAL PARTNER

powered by ENGINSOFT

CONTROLLER FOR AN INVERTED PENDULUM

This Scilab tutorial is dedicated to the study of a linear quadratic regulator for an inverted pendulum based on optimal control theory. In this tutorial the reader will learn how to develop a controller for an inverted pendulum starting from the equations of motion and how to use the animated plots in Scilab/Xcos.

Level 

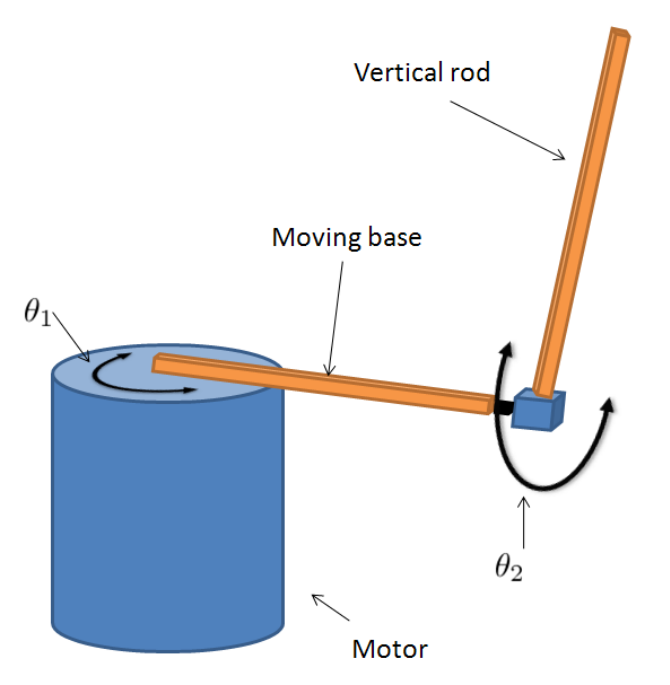
www.openeering.com

Step 1: Problem description

In this tutorial we use Scilab to develop a controller that maintains a rod in its ***vertical position*** (unstable position). In our example, the rod is hinged on a support that allows only one type of lateral movement.

As reported in the figure on the right, the inverted pendulum is composed by a vertical rod (homogeneous rod with length l_2 and mass M_2) hinged on a support rod (homogeneous rod with length l_1 and mass M_1). The vertical rod is in an orthogonal plane with respect to the moving base (rotational joint).

In our experiment, we suppose to know (at each time step) the support angle θ_1 of the moving base and the angle θ_2 of the vertical rod. Moreover, the angle θ_1 is the only parameter in which we can act to control the entire system by using a DC motor.



A schematic drawing of an inverted pendulum with a moving base.

Step 2: Roadmap

This tutorial is particularly challenging and is the first Openeering tutorial with five puffins.

Since the comprehension of this tutorial is demanding and some concepts of Scilab programming are taken for granted, we highly recommend the reader to download the entire source code and Xcos schemes from the registered user area.

Unlike the previous tutorials, in this case the source code should be used as a support documentation material. Source codes and Xcos schemes should be read together with these pages.

In this tutorial the following topics are covered:

- Problem description and system equations;
- Non-linear system solution;
- Linear-quadratic regulator;
- Solving the self-erecting problem
- Plotting animations in Scilab and Xcos
- References and exercises

Note: All the simulations are done using ***Scilab 5.4***.

Descriptions	Steps
Problem description and System equations	3 - 7
Non-linear system solution	8 - 12
Linear-quadratic regulator	13 - 20
Solving self-erecting problem	21 – 24
Plotting animations in Scilab and Xcos	25 - 27
Conclusions and exercises	28 - 31

Step 3: Problem data

The problem is described in term of the ***angles*** data θ_1 and θ_2 and their ***angular velocities*** $\dot{\theta}_1$and $\dot{\theta}_2$.

For real applications, it is assumed that it is possible to have the absolute position of the angle data θ_1 and θ_2 at each time step, while the angular velocities $\dot{\theta}_1$and $\dot{\theta}_2$ are recovered with an interpolation procedure.

The ***control variable*** is the Voltage $V(t)$ which can be applied to the motor.

On the right, we report data and notations used throughout the text.

In this tutorial we use some shortcuts (reported on the right) in order to simplify some equations. These shortcuts are used when managing the equations using symbolic computations.

In this tutorial, all the ***symbolic computations*** are done using **Maxima** which is an open source project for symbolic computations (see References at Step 28).

Problem measure variables:

- Absolute angle position of the first rod θ_1 [rad];
- Angular velocity of the first rod $\dot{\theta}_1$ [rad/s];
- Absolute angle position of the second rod θ_2 [rad];
- Angular velocity of the second rod $\dot{\theta}_2$[rad/s];
- Voltage control $V(t)$ [V].

Problem data:

- Moment of inertia of the first rod $J_1 = \frac{M_1 l_1^2}{3} = 0.005$ [kg m^2];
- Length of the first rod $l_1 = 0.2$ [m];
- Mass of the second $M_2 = 0.15$ [kg];
- Length of the second rod $l_2 = 0.5$ [m];
- Moment of inertia of the second rod $J_2 = \frac{M_2 l_2^2}{3}$ [kg m^2];
- Motor torque $K_\phi = 0.008$;
- Gear ratio $N = 15$;
- Internal motor resistor $R = 2.5$ [Ohm].

Shortcuts:

- $a = J_1 + M_2 l_1^2$;
- $b = \frac{1}{2} M_2 l_1 l_2$;
- $c = J_2$;
- $d = \frac{1}{2} M_2 g l_2$;
- $e = \frac{N K_\phi}{R}$;
- $f = \frac{N^2 K_\phi^2}{R}$;
- $\delta = \mathrm{ac} - \mathrm{b}^2$;

Step 4: System equations

To obtain the system equations we use the Euler-Lagrange approach. It is necessary to define the kinetic energy T and the potential energy U with respect to the generalized free coordinates of the system that, in our case, are the angles and their derivatives.

In particular for the support rod we have:

- Potential energy: $U_1 = 0$;
- Kinetic energy: $T_1 = \frac{1}{2} J_1 \dot{\theta}_1^2$;

while for the vertical rod we have:

- Potential energy: $U_2 = M_2 g \cdot \frac{l_2}{2} \cos\ \theta_2$;
- Kinetic energy: $T_2 = \frac{1}{2} J_2 \dot{\theta}_2^2 + \frac{1}{2} M_2 (l_1 \dot{\theta}_1)^2 + \frac{1}{2} M_2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos\ \theta_2$.

The kinetic energy T_2 can be obtained in an easy way using the Koenig's theorem since the rod is homogeneous, which gives

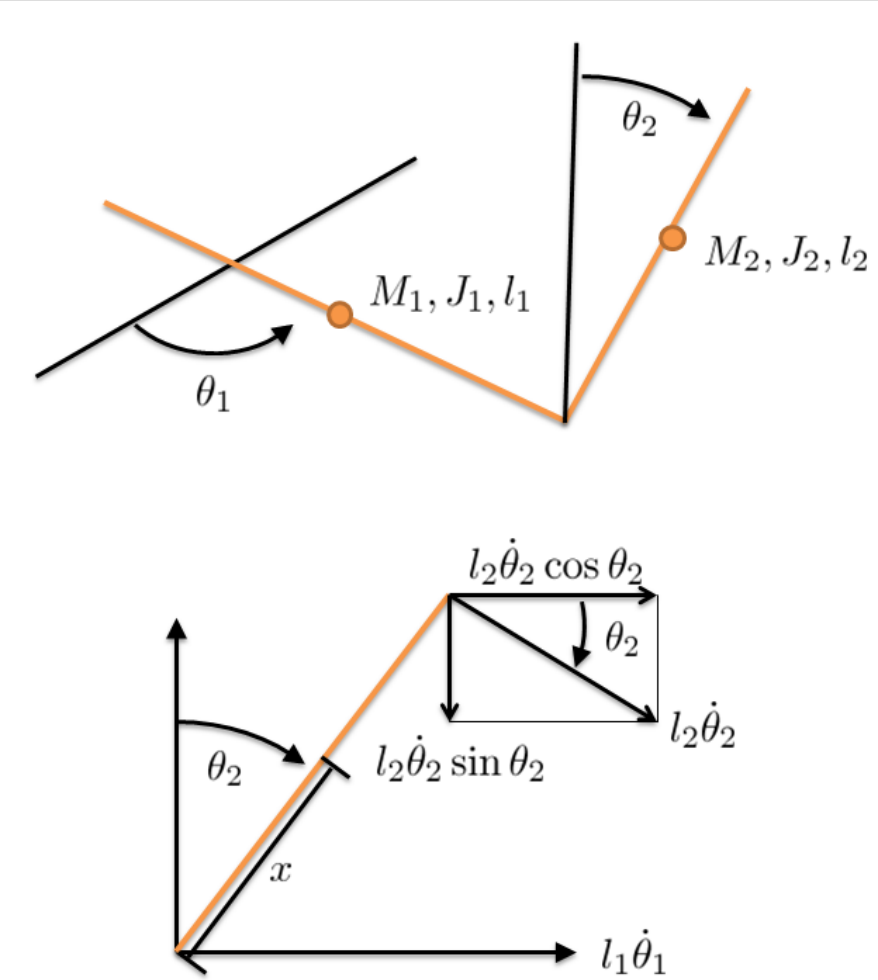
$$T_2 = \frac{1}{2} J_{2,\mathrm{MC}} \dot{\theta}_2^2 + \frac{1}{2} M_2 ((v_1 + v_2 \cos\ \theta_2)^2 + (v_2 \sin\ \theta_2)^2)$$

where $J_{2,MC}$ is the inertia with respect to the center of mass M_2, $v_1 = l_1 \dot{\theta}_1$ and $v_2 = \frac{l_2}{2} \dot{\theta}_2$. The J_2 can be express in terms of $J_{2,MC}$ using the theorem of Huygens-Steiner which gives $J_2 = J_{2,MC} + M_2 l_2^2$.

The kinetic energy T_2 can also be obtained using integration procedure over infinitesimal segments, i.e.

$$T_2 = \frac{1}{2} \int_0^{l_2} \left[\left(x\dot{\theta}_2 \cos\ \theta_2 + l_1 \dot{\theta}_1 \right)^2 + \left(x\dot{\theta}_2 \sin\ \theta_2 \right)^2 \right] \rho S\, dx$$

where ρ is the density and S is the section area.



Some adopted conventions

Step 5: System equations

The Lagrangian of the system can be written as

$$\begin{aligned} L &= T - U = (T_1 + T_2) - (U_1 + U_2) \\ &= \frac{1}{2}(J_1 + M_2 l_1^2)\dot{\theta}_1^2 + \frac{1}{2}J_2\dot{\theta}_2^2 + \frac{1}{2}M_2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos\,\theta_2 - \frac{1}{2}M_2 g l_2 \cos\,\theta_2 \end{aligned}$$

in the generalized coordinates θ_1 and θ_2 and generalized velocities $\dot{\theta}_1$ and $\dot{\theta}_2$.

The equations of motion are obtained using the Euler-Lagrange equation:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = 0$$

This expresses the equations starting from the system kinetic energy and system potential energy.

The equations of motion are reported on the right (further details will be presented in the following steps).

The nonlinear system of equations is:

$$\begin{cases} \left(J_1 + M_2 l_1^2\right)\ddot{\theta}_1 + \frac{1}{2}M_2 l_1 l_2 \ddot{\theta}_2 \cos\,\theta_2 - \frac{1}{2}M_2 l_1 l_2 \dot{\theta}_2^2 \sin\,\theta_2 = \tau \\ J_2\ddot{\theta}_2 + \frac{1}{2}M_2 l_1 l_2 \ddot{\theta}_1 \cos\,\theta_2 - \frac{1}{2}M_2 g l_2 \sin\,\theta_2 = 0 \end{cases}$$

where:

- τ is the applied torque to the support rod;
- $J_1 = \frac{M_1 l_1^2}{3}$ is the momentum of inertia of the first rod;
- $J_2 = \frac{M_2 l_2^2}{3}$ is the momentum of inertia of the second rod.

Step 6: System equations (details of derivation)

Details of the derivation of equations are reported here, while the Maxima script we used for the symbolic computation is reported on the right.

For the first equations we have:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_1} = (J_1 + M_2 l_1^2)\ddot{\theta}_1 + \frac{1}{2}M_2 l_1 l_2 \ddot{\theta}_2 \cos\,\theta_2 - \frac{1}{2}M_2 l_1 l_2 (\dot{\theta}_2)^2 \sin\,\theta_2$$

$$\frac{\partial L}{\partial \theta_1} = 0$$

That corresponds exactly to the eq1_a and eq1_b of the Maxima script.

The sum of eq1_a and eq1_b gives the equation of motion where the free variable is θ_1.

For the second equations we have:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_2} = J_2\ddot{\theta}_2 + \frac{1}{2}M_2 l_1 l_2 \ddot{\theta}_1 \cos\,\theta_2 - \frac{1}{2}M_2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \sin\,\theta_2$$

$$\frac{\partial L}{\partial \theta_2} = -\frac{1}{2}M_2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \sin\,\theta_2 + \frac{1}{2}M_2 g l_2 \sin\,\theta_2$$

That corresponds exactly to the eq2_a and eq2_b of the Maxima script. The sum of eq2_a and eq2_b gives the equation of motion where the free variable is θ_2.

Maxima script used to obtain the system equations:

```
/* Equations of motion */
theta1: theta1(t);
theta2: theta2(t);

omega1: diff(theta1,t);
omega2: diff(theta2,t);

/* Lagrangian of the system */
T1: 1/2*J1*omega1^2;
U1: 0;
T2: 1/2*J2*omega2^2 + 1/2*M2*(l1*omega1)^2 +
1/2*M2*l1*l2*omega1*omega2*cos(theta2);
U2: M2*g*l2/2*cos(theta2);
L: (T1+T2) -(U1+U2);

/* Lagrangian derivation */
eq1_a: expand(diff(diff(L,omega1),t));
eq1_b: diff(L,theta1);
eq1: eq1_a + eq1_b;


eq2_a: diff(diff(L,omega2),t);
eq2_b: expand(diff(L,theta2));
eq2: eq2_a + eq2_b;
```

Step 7: System equations

The applied torque is given by a DC motor which is controlled in voltage V. This relation is typically written as:

$$\tau(t) = NK_\phi I(t) = NK_\phi\left(V(t) - K_\phi\dot{\theta}_1\right)\frac{1}{R} = NK_\phi\frac{V(t)}{R} - \frac{N^2K_\phi^2}{R}\dot{\theta}_1$$

where

- N represents the gear ratio;
- K_ϕ is the torque constant that depends on the motor geometry;
- $I(t)$ is the current of the motor;
- $V(t)$ is the voltage used to control the motor;
- $K_\phi\dot{\theta}_1$ represents the counter electromotive force;
- R is the resistor of the rotor winding.

The nonlinear complete model is:

$$\begin{cases}\left(J_1 + M_2l_1^2\right)\ddot{\theta}_1 + \frac{1}{2}M_2l_1l_2\ddot{\theta}_2\cos\,\theta_2 - \frac{1}{2}M_2l_1l_2\dot{\theta}_2^2\sin\,\theta_2 = \tau \\ J_2\ddot{\theta}_2 + \frac{1}{2}M_2l_1l_2\ddot{\theta}_1\cos\,\theta_2 - \frac{1}{2}M_2gl_2\sin\,\theta_2 = 0\end{cases}$$

where

$$\tau(t) = NK_\phi\frac{V(t)}{R} - \frac{N^2K_\phi^2}{R}\dot{\theta}_1$$

We now use the following notations for the nonlinear system

$$\begin{cases}a\cdot\ddot{\theta}_1 + b\cdot\ddot{\theta}_2\cos\,\theta_2 - b\cdot\dot{\theta}_2^2\sin\,\theta_2 = e\cdot V(t) - f\cdot\dot{\theta}_1 \\ c\cdot\ddot{\theta}_2 + b\cdot\ddot{\theta}_1\cos\,\theta_2 - d\cdot\sin\,\theta_2 = 0\end{cases}$$

where shortcuts notations are explained at step 3.

Step 8: Nonlinear system solution

In order to solve the nonlinear system it is necessary to state the problem in the form $\dot{y}(t) = f(t, y)$. This can be easy done since the system is linear in $\ddot{\theta}_1$ and $\ddot{\theta}_2$.

We obtain the following expressions using the Maxima script on the right. The expressions for $\ddot{\theta}_1$ and $\ddot{\theta}_2$ are:

$$\begin{cases} \ddot{\theta}_1 = -\dfrac{\mathrm{c\,e\,V(t)} + \mathrm{b\,c}\sin(\theta_2)\,\dot{\theta}_2^2 - \mathrm{bd}\cos(\theta_2)\sin(\theta_2) - \mathrm{c\,f}\,\dot{\theta}_1}{(\mathrm{b}^2\cos^2(\theta_2) - \mathrm{a\,c})} \\ \ddot{\theta}_2 = \dfrac{\mathrm{b\,e}\cos(\theta_2)\,\mathrm{V(t)} + \mathrm{b}^2\cos(\theta_2)\sin(\theta_2)\,\dot{\theta}_2^2 - a\,d\sin(\theta_2) - bf\dot{\theta}_1\cos(\theta_2)}{(\mathrm{b}^2\cos^2(\theta_2) - \mathrm{a\,c})} \end{cases}$$

which leads the following ***state representation*** of the system:

$$\begin{cases} \dot{\theta}_1 = \theta_{1p} \\ \dot{\theta}_2 = \theta_{2p} \\ \dot{\theta}_{1p} = -\dfrac{\mathrm{c\,e\,V(t)} + \mathrm{b\,c}\sin(\theta_2)\,\theta_{2p}^2 - \mathrm{bd}\cos(\theta_2)\sin(\theta_2) - \mathrm{c\,f}\,\theta_{1p}}{(\mathrm{b}^2\cos^2(\theta_2) - \mathrm{a\,c})} \\ \dot{\theta}_{2p} = \dfrac{\mathrm{b\,e}\cos(\theta_2)\,\mathrm{V(t)} + \mathrm{b}^2\cos(\theta_2)\sin(\theta_2)\,\theta_{2p}^2 - a\,d\sin(\theta_2) - bf\theta_{1p}\cos(\theta_2)}{(\mathrm{b}^2\cos^2(\theta_2) - \mathrm{a\,c})} \end{cases}$$

Note: We always have $(\mathrm{b}^2\cos^2(\theta_2) - \mathrm{a\,c}) \neq 0$.

Maxima script used to obtain the expressions for $\ddot{\theta}_1$ and $\ddot{\theta}_2$:

```
/* Non-linear system formulation */
eqnl1 : a*theta1_pp + b*theta2_pp*cos(theta2)  -
b*theta2_p^2*sin(theta2) = (e*V - f*theta1_p);
eqnl2 : b*theta1_pp*cos(theta2) + c*theta2_pp -
d*sin(theta2) = 0;

sol: linsolve([eqnl1,eqnl2],[theta1_pp,theta2_pp]);

display(sol[1]);
display(sol[2]);
```

Step 9: Free evolution in Scilab

We now use previous equations to write our Scilab code. In this example we study the free evolution ($V(t) = 0$) of the nonlinear system. The solution is implemented in Scilab language where the numerical solution is obtained using the ***ode*** Scilab function*.

We study the problem using the initial conditions:

$$\begin{cases} \theta_1 &= 0 \\ \theta_2 &= 90° \\ \dot{\theta}_1 &= 0 \\ \dot{\theta}_2 &= 0 \end{cases}$$

The obtained results are reported on the figure below.

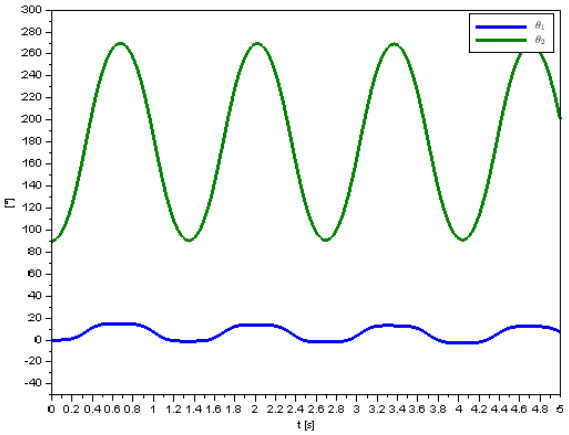


Figure 1: Free evolution plot

** Refer to the tutorial "Modeling in Scilab: pay attention to the right approach - Part 1" for more information on how to model a physical system described by ODE using Scilab standard programming language.*

```
// Inverted pendulum parameters
J1 = 0.05; l1 = 0.2; M2 = 0.15; l2 = 0.5;
J2 = M2*l2^2/3; Kphi = 0.008; N = 15; R = 2.5; g = 9.8;

// Derived data
a = J1 + M2*l1^2; b = 1/2*M2*l1*l2; c = J2;
d = 1/2*M2*g*l2; e = N*Kphi/R; f = N^2*Kphi^2/R;
delta = a*c-b^2;

function ydot=inv_nl_pend(t, y)
// The non-linear system inverted pendulum
// get variable
theta1 = y(1); theta2 = y(2);
theta1p = y(3); theta2p = y(4);
s2 = sin(theta2); c2 = cos(theta2);
den = b^2*c2^2 - a*c;
Vt = 0;
ydot = zeros(4,1);
ydot(1) = theta1p;
ydot(2) = theta2p;
ydot(3) = -(c*e*Vt + b*c*s2*theta2p^2-b*d*c2*s2-
c*f*theta1p)/den;
ydot(4) = (b*e*c2*Vt + b^2*c2*s2*theta2p^2-a*d*s2-
b*f*theta1p*c2)/den;
endfunction

// Initial condition
y0 = [0;90*%pi/180;0;0];
t = linspace(0,5000,2001)/1000;
t0 = 0;
// Solving the system
sol = ode(y0, t0, t, inv_nl_pend);

// Plotting of model data
plot(t, sol(1:2,:)/%pi*180);
p = get("hdl");
p.children.mark_mode = "on";
p.children.thickness = 3;
xlabel("t [s]"); ylabel("[°]");
legend(["$\theta_1$";"$\theta_2$"]);
```

Step 10: Free evolution using Xcos

The Xcos scheme that implements the previous system is available in the Xcos file ***inv_pend.zcos*** (see figure) and it uses the following blocks:

- A "***CONST_m***" constant for modeling the V(t) signal (equal to zero in this case);
- A ***superblock*** that models the nonlinear system (see step 11);
- A "***MUX***" block to split signals;
- A "***TOWS_c***" block for saving data to workspace that gives the values of the angles to a variable named ***sol***;

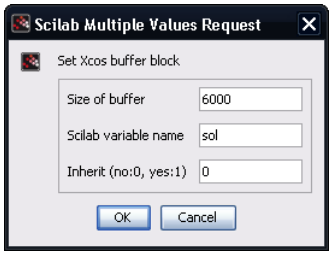


Figure 2: data to Scilab workspace configuration

All files are available for download in the Openeering registered users' area. Again, we highly recommend downloading the source codes for a better understanding of this tutorial.

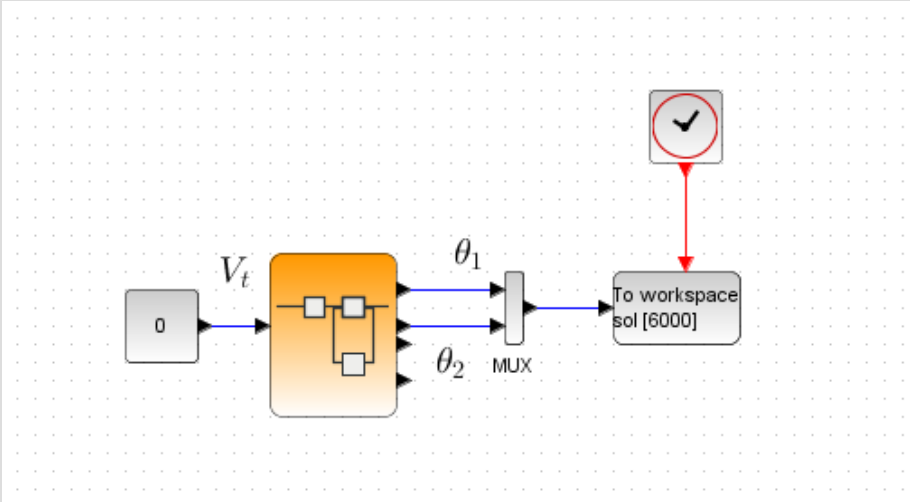


Figure 3: The Xcos scheme for free evolution

Step 11: Nonlinear system block

The modeling of the ***nonlinear system block*** is reported on the right and it uses the following basic blocks:

- Four integration blocks for recovering the angles starting from the angular accelerations;
- Two expression blocks that model nonlinearities. The two expressions are exactly the two differential equations described in step 8.

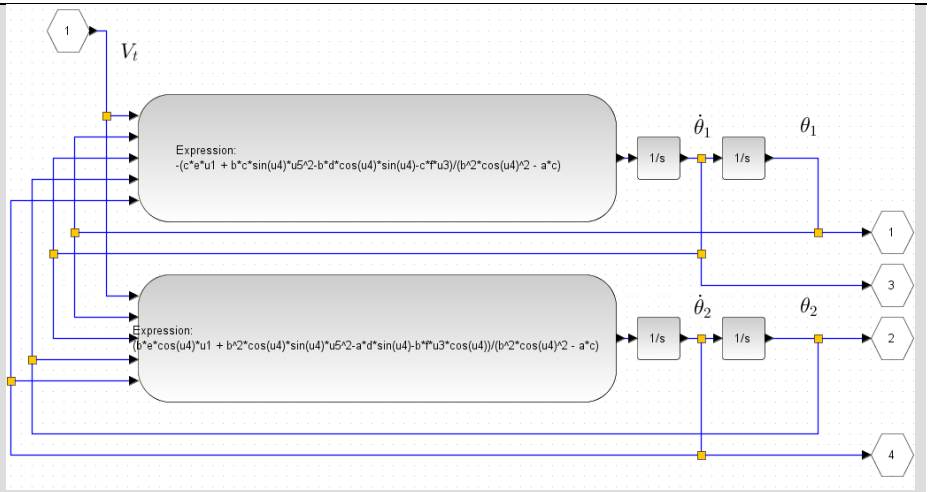


Figure 4: The nonlinear system block

Step 12: Plot the results

Since the variable ***sol*** contains the results of the Xcos simulation, we can plot the two angles θ_1 and θ_2 using the command:

```
plot(sol.time, sol.values*180/%pi)
```

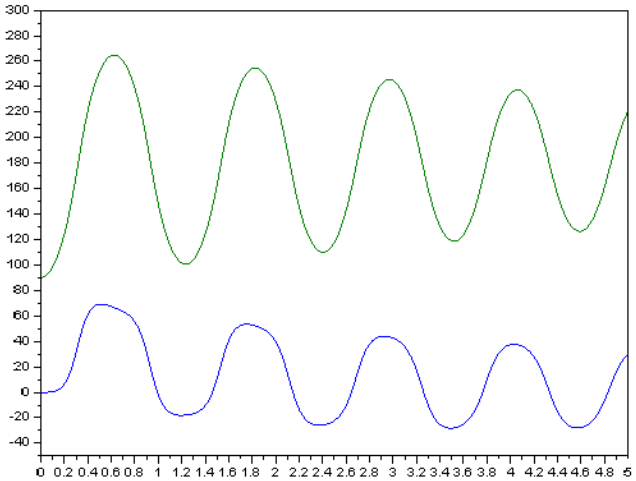


Figure 5: Angles positions at time steps

Step 13: Linearized model

It is possible to linearize the model around the point

$$\theta_1 = \theta_2 = \dot{\theta}_1 = \dot{\theta}_2 = 0$$

using small angle displacements and small velocity displacements having the following approximations:

$$\sin\theta \approx \theta \text{ and } \cos\theta \approx 1.$$

This gives:

$$\begin{bmatrix} J_1 + M_2 l_1^2 & \frac{1}{2} M_2 l_1 l_2 \\ \frac{1}{2} M_2 l_1 l_2 & J_2 \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -\frac{1}{2} M_2 g l_2 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} \tau \\ 0 \end{bmatrix}$$

Hence we use the Maxima script on the right to obtain the state model representation of the system:

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{bd}{\delta} & -c\frac{K_\phi^2 N^2}{\delta R} & 0 \\ 0 & \frac{ad}{\delta} & b\frac{K_\phi^2 N^2}{\delta R} & 0 \end{bmatrix}}_{A} \cdot \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \frac{cK_\phi N}{\delta R} \\ -\frac{bK_\phi N}{\delta R} \end{bmatrix}}_{B} \cdot V(t)$$

In system theory, this system is typically written in the from

$$\dot{x} = A \cdot x + B \cdot V.$$

Maxima script used to obtain the state model representation:

```
/* Maxima script for obtaining state representation of an
inverted pendulum in V*/
e : N*Kt/R;
f : N^2*Kt^2/R;

/* System */
/* tau = e*V - f*theta1_p */
eq1 : a*theta1_pp + b*theta2_pp = e*V - f*theta1_p;
eq2 : c*theta2_pp + b*theta1_pp - d*theta2 = 0;

sol: linsolve([eq1,eq2],[theta1_pp,theta2_pp]);

/* Theta1_pp processing */
res1 : expand(rhs(sol[1]));
eq1_theta2 : factor(part(res1,2));
eq1_theta1_p : factor(part(res1,3));
eq1_V : factor(part(res1,1));

/* Theta2_pp processing */
res2 : expand(rhs(sol[2]));
eq2_theta2 : factor(part(res2,2));
eq2_theta1_p : factor(part(res2,3));
eq2_V : factor(part(res2,1));
```

Step 14: Transfer functions and stability analysis

If we apply a Laplace transformation*, it is possible to obtain the transfer function (in terms of control variable τ)

$$W_1(s) = \frac{\theta_1(s)}{\tau(s)} \text{ and } W_2(s) = \frac{\theta_2(s)}{\tau(s)}.$$

Hence, from the relations

$$a \cdot s^2\theta_1(s) + b \cdot s^2\theta_2(s) = \tau(s)$$

$$b \cdot s^2\theta_1(s) + c \cdot s^2\theta_2(s) - d \cdot \theta_2(s) = 0$$

we have

$$W_1(s) = \frac{\theta_1(s)}{\tau(s)} = \frac{cs^2 - d}{as^2\left[\left(c - \frac{b^2}{a}\right)s^2 - d\right]} = \frac{-\frac{d}{\delta} + \frac{c}{\delta}s^2}{-\frac{ad}{\delta}s^2 + s^4}$$

and

$$W_2(s) = \frac{\theta_2(s)}{\tau(s)} = \frac{-\frac{b}{a}}{\left(c - \frac{b^2}{a}\right)s^2 - d} = \frac{-b/\delta}{-\frac{da}{\delta} + s^2}$$

Since $\frac{b^2}{a} < c$ the poles of the transfer function $W_1(s)$ are both reals and so the system are unstable.
Note: It is left as an exercise to prove the same fact when the controller is expressed in term of Voltage instead of torque applied.

**Refer to the tutorial "Introduction to Control Systems in Scilab" for more information on state-space, transfer function representation and converting methods.*

Maxima script used to obtain the transfer functions:

```
/* Maxima script for transfer functions */
eq1 : a*s^2*theta1_s + b*s^2*theta2_s = tau_s;
eq2 : b*s^2*theta1_s + c*s^2*theta2_s - d*theta2_s = 0;

sol: linsolve([eq1,eq2],[theta1_s,theta2_s]);

W1_s : rhs(sol[1])/tau_s;
W2_s : rhs(sol[2])/tau_s;
```

Step 15: Using the Scilab control toolbox

Here, in the right, we report the computation (in τ) using the Scilab control toolbox CACSD.

The state equation is

$$\dot{x} = A \cdot x + B \cdot u$$

with

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{bd}{\delta} & 0 & 0 \\ 0 & \frac{ad}{\delta} & 0 & 0 \end{bmatrix} \text{ and } B = \begin{bmatrix} 0 \\ 0 \\ \frac{c}{\delta} \\ -\frac{b}{\delta} \end{bmatrix}$$

while the observer equation is

$$y = C \cdot x + D \cdot u$$

where

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \text{ and } D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Note: It is left as an exercise to prove the same fact when the controller is expressed in term of Voltage instead of torque applied.

```
// Inverted pendulum parameters
J1 = 0.05;
l1 = 0.2;
M2 = 0.15;
l2 = 0.5;
J2 = M2*l2^2/3;
Kphi = 0.008;
N = 15;
R = 2.5;
g = 9.8;

// Derived data
a = J1 + M2*l1^2;
b = 1/2*M2*l1*l2;
c = J2;
d = 1/2*M2*g*l2;
e = N*Kphi/R;
f = N^2*Kphi^2/R;
delta = a*c-b^2;

// Define system matrix (in tau)
A = [0 0 1 0; 0 0 0 1; 0 -b*d/delta 0 0; 0 a*d/delta 0
0];
B = [0; 0; c/delta; -b/delta]
C = [1 0 0 0; 0 1 0 0];
D = [0;0];

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

Step 16: Optimal control

In optimal control theory, a linear-quadratic (***LQ***) regulator consists of feedback controller of the form

$$u = Kx$$

that minimizes the following index of quality

$$\boldsymbol{J = \int_0^{\infty} (x^T Q x + u^T R u) dt}$$

where Q is a positive semi-definitive matrix and R is a positive definite matrix. The Q matrix represents the weight associated with the state while R represents the weight associated with the input.

Under these assumptions it is possible to find that:

$$K = -R^{-1}B^T P$$

where P is the solution of the Riccati equation

$$A^T P + PA - PBR^{-1}B^T P + Q = 0$$

In order to develop this kind of controller it is necessary to check its controllability. The controllability means that for any $t_1 > t_0$ and for any states x_0 and x_1 there exists an input $u(t)$ such that the system is taken from initial state $x_0(t_0)$ to the final $x_1(t_1)$. This allows the system to converge towards the desire state since it is possible to place the poles of the system everywhere in the complex plane.

The Scilab function used to compute the controller is ***LQR***:

```
[K, X] = lqr(PSys)
```

where:

- **K :** is the Kalman gain such that $A + BK$ is stable;
- **P:** is the stabilizing solution of the Riccati equation.

The P system is obtained using the following commands:

```
Q = diag([100,100,1,1]);  // (4 states, 1 input)
R = 1;
nstates = 4;
Big = sysdiag(Q,R);        // Now we calculate C1 and D12
[w,wp] = fullrf(Big);
C1=wp(:,1:nstate);
D12=wp(:,nstate:$);    //[C1,D12]'*[C1,D12]=Big
PSys = syslin('c',A,B,C1,D12);  // The sys (cont-time)
[K,P] = lqr(PSys)
spec(A+B*K)                    // check stability
```

where ***nstate*** denotes the number of state variables.

Step 17: Controllability proof

In order to develop a LQ regulator it is necessary to prove the system controllability and hence the possibility to allocate the eigenvalues to your preference.

Hence, it is necessary to check that

$$\text{rank}[B|AB|A^2B|A^3B] = 4$$

where

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{bd}{\delta} & -c\frac{K_\phi^2 N^2}{\delta R} & 0 \\ 0 & \frac{ad}{\delta} & b\frac{K_\phi^2 N^2}{\delta R} & 0 \end{bmatrix} \text{ and } B = \begin{bmatrix} 0 \\ 0 \\ \frac{cK_\phi N}{\delta R} \\ -\frac{bK_\phi N}{\delta R} \end{bmatrix}.$$

Using the Scilab and Maxima codes on the right, it easy to prove, that the rank is 4.

Note: It is left as an exercise to prove the same fact when the controller is expressed in term of torque applied instead of Voltage control.

Scilab script to detect controllability:

```
// Check controllability
AB = [B, A*B, A*A*B, A*A*A*B]
rank(AB)
spec(AB)
```

Maxima script used to check the controllability of the system:

```
/* Controllability check */
a0 : Kphi*N;
a1 : b*d/delta;
a2 : a*d/delta;
a5 : c/delta*Kphi*N/R
a3 : a5 * a0;
a6 : b/delta*Kphi*N/R;
a4 : a6 * a0;

A : matrix([0, 0, 1, 0], [0, 0, 0, 1], [0, -a1, -a5*a0,
0], [0, a2, a6*a0, 0]);
B : matrix([0], [0], [a5], [-a6]);
R : addcol(B,A.B,A.(A.B),A.(A.(A.B)));
detR : factor(expand(determinant(R)));
```

Step 18: LQ regulator summary

To put everything together, the development of a LQ regulator requires computing the Kalman gain matrix. The complete Scilab code for the LQ regulator is reported on the right.

The LQ regulator is computed using the Scilab function ***lqr*** which requires as input the state space representation of the system.

The ***lqr*** function computes the regulator associated to the L2 norm of z (cost function) where

$$z = C_1 x + D_{12} u$$

and

$$[x,u]^T \cdot \tilde{Q} \cdot [x,u]$$

$$\tilde{Q} = \begin{bmatrix} C_1^T \\ D_{12}^T \end{bmatrix} \cdot [C_1 \quad D_{12}].$$

The computation of C_1 and D_{12} are done using the full rank factorization using the Scilab command ***fullrf***.

```
// Inverted pendulum data
J1 = 0.005; l1 = 0.2;
M2 = 0.15;  l2 = 0.5;  J2 = M2*l2^2/3;
Kphi = 0.008; N = 15; R = 2.5;
g = 9.8;

// Derived data
a = J1 + M2*l1^2;  b = 1/2*M2*l1*l2;
c = J2;  d = 1/2*M2*g*l2;
e = N*Kphi/R;  f = N^2*Kphi^2/R;
delta = a*c-b^2;

// Define system matrix (in tau)
A = [0 0 1 0; 0 0 0 1; 0 -b*d/delta -c/delta*f 0; 0
a*d/delta b/delta*f 0];
B = [0; 0; c/delta*e; -b/delta*e]
C = [1 0 0 0; 0 1 0 0];
D = [0;0];

// Check contollability
AB = [B, A*B, A*A*B, A*A*A*B]
rank(AB)
spec(AB)

// LQ project
Q = diag([100,100,10,10]);              // Usual notations
x'Qx + u'Ru (4 states, 1 input)
R = 1;
Big = sysdiag(Q,R);                     // Now we calculate C1
and D12
nstates = 4;
[w,wp] = fullrf(Big);C1=wp(:,1:nstates);D12=wp(:,$:$);
//[C1,D12]'*[C1,D12]=Big
P = syslin('c',A,B,C1,D12);             // The plant
(continuous-time)
[K,X] = lqr(P)
spec(A+B*K)                             // check stability
```

Step 19: LQ regulator in Xcos

The LQ regulator is here described using an Xcos scheme. The values of K1, K2, K3 and K4 are given from the previous step.

In this configuration it is possible to study the system under noise perturbation.

The context configuration parameters are:

```
// Inverted pendulum parameters
J1 = 0.005;    l1 = 0.2;
M2 = 0.15;     l2 = 0.5; J2 = M2*l2^2/3;
Kphi = 0.008; N = 15;    R = 2.5; g = 9.8;

// Derived data
a = J1 + M2*l1^2;  b = 1/2*M2*l1*l2;  c = J2;
d = 1/2*M2*g*l2;   e = N*Kphi/R;      f = N^2*Kphi^2/R;
delta = a*c-b^2;

// Initial conditions
theta1 = 90*%pi/180;     // init. position
theta2 =  3*%pi/180;     // init. position
theta1_p = 0;            // init. ang. vel.
theta2_p = 0;           // init. ang. vel.

maxang = 0*%pi/180;
maxvel = 0;
initnoise = 100;

K = [10.    120.55385    7.5116657    21.523317];

K1 = K(1);
K2 = K(2);
K3 = K(3);
K4 = K(4);
```

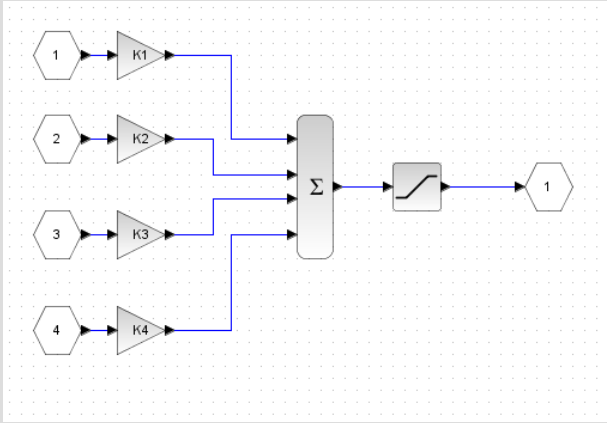


Figure 6: the LQ regulator in an Xcos scheme

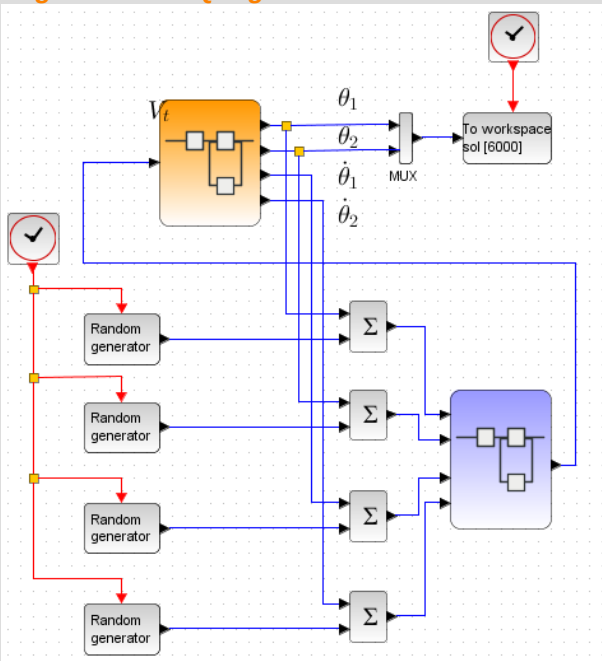


Figure 7: the entire Xcos scheme. With respect to step 10, the LQ regulator has been added in the blue subsystem.

Step 20: Results

As in step 12, we can now plot the two angles with the command:

```
plot(sol.time, sol.values*180/%pi)
```

The initial configuration is in this case

$$\begin{cases} \theta_1 &= 90° \\ \theta_2 &= 3° \\ \dot{\theta}_1 &= 0 \\ \dot{\theta}_2 &= 0 \end{cases}$$

Note that, as already explained at the beginning of step 13, this regulator works only for small perturbation of the angle θ_2 that represents the position of the vertical rod.

As an exercise, try to change the initial configuration in the context configuration parameters and see how the plot changes.

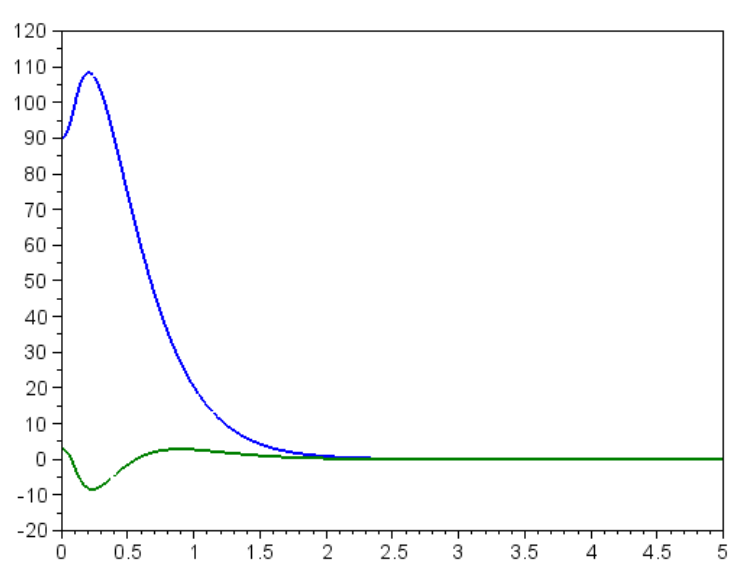


Figure 8: Angles positions at time steps

Step 21: Control for a self-erecting inverted pendulum

Now, we want to develop a new controller that takes the pendulum from its stable position to the inverted, unstable position and then maintains this configuration.

The idea is to control the moving rod, through the use of a PID regulator, such that the moving rod is in an opposite direction with respect of the pendulum. Then, when the pendulum reaches a prefixed threshold we move from the PID regulator to the LQ regulator.

The full scheme is reported on the right. This scheme is composed by the following main components:

- The ***nonlinear system block*** used to simulate the non-linear system;
- The ***LQ regulator block*** used to control the pendulum in the vertical position;
- A ***PID controller*** used to take the pendulum from its stable configuration to the unstable one;
- A ***controller switcher*** that chooses between the LQ regulator and PID controller.

The two new blocks, the PID controller and the controller switcher, will be described in the following steps.

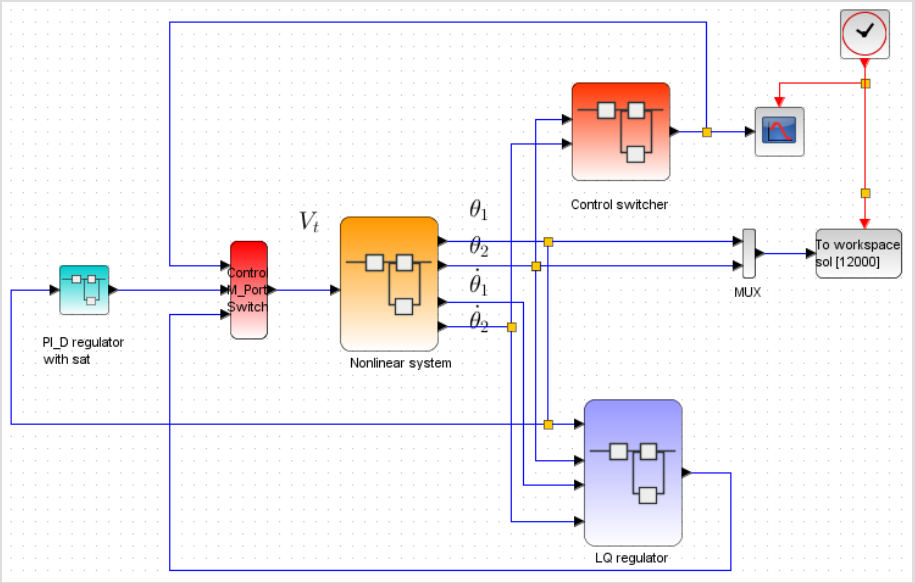


Figure 9: Self-erecting controller for the pendulum (see file inv_pend_se.zcos)

Step 22: Controller switcher

The controller switcher is based on the Scilab function ***inv_pend_mode***.

The idea is to move from a configuration to another checking the angle and its velocity.

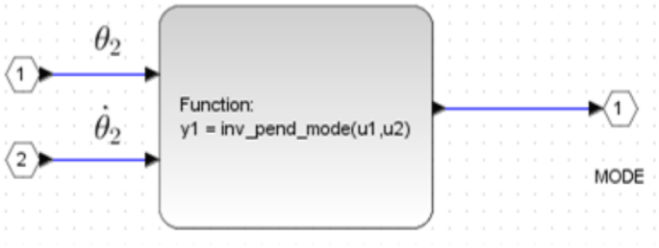


Figure 10: Controller switcher

In the first part we move θ_2 such that $0 \leq \theta_2 < 2\pi$ and then we check the angle condition. Remember that the conditions that we want to control are $\theta_2 \approx 0$ and $\theta_2 \approx 2\pi$.

```
function [mctrl]=inv_pend_mode(theta2, dtheta2);
// Select the mode of control to be applied
// 1 for trying to get vertical position
// 0 for maintain vertical position

// unwrapping of theta2
theta2u = unwrap(theta2);

mctrl = 0;
if (abs(theta2u) <= PositionThreshold) & (abs(dtheta2) <=
VelocityThreshold) then
    mctrl = 1;
end

if (abs(2*%pi - abs(theta2u)) <= PositionThreshold) &
(abs(dtheta2) <= VelocityThreshold) then
    mctrl = 1;
end
endfunction

function b=unwrap(a);
aa = abs(a);
k = floor(aa / (2*%pi));
if a > 0 then
   b = a - 2 * k * %pi;
else
   b = aa - 2 * k * %pi;
   b = 2 * %pi - b;
end
endfunction
```

Step 23: PID controller

The PID controller is used to moving rod is in an opposite way to direction of rising of the pendulum.

Several configurations of PID controllers exist in literature. Here, we use the following:

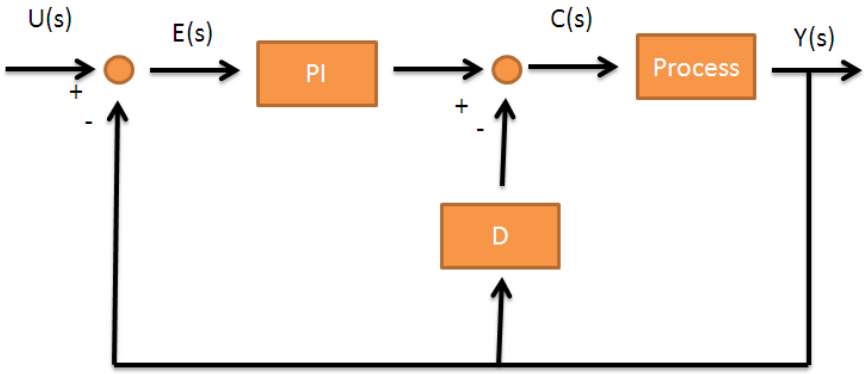


Figure 11: Scheme of our PID controller

In general, this configuration is characterized by the derivative term applied to the output. This controller is particularly useful when we prefer looking for stability than tracking the reference signal.

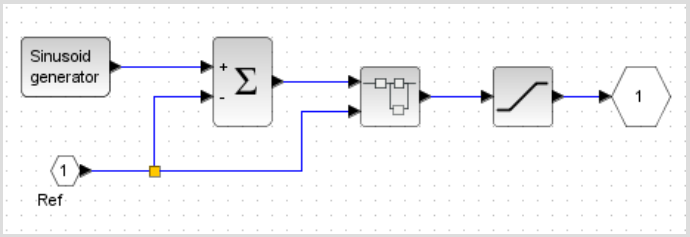


Figure 12: PID scheme

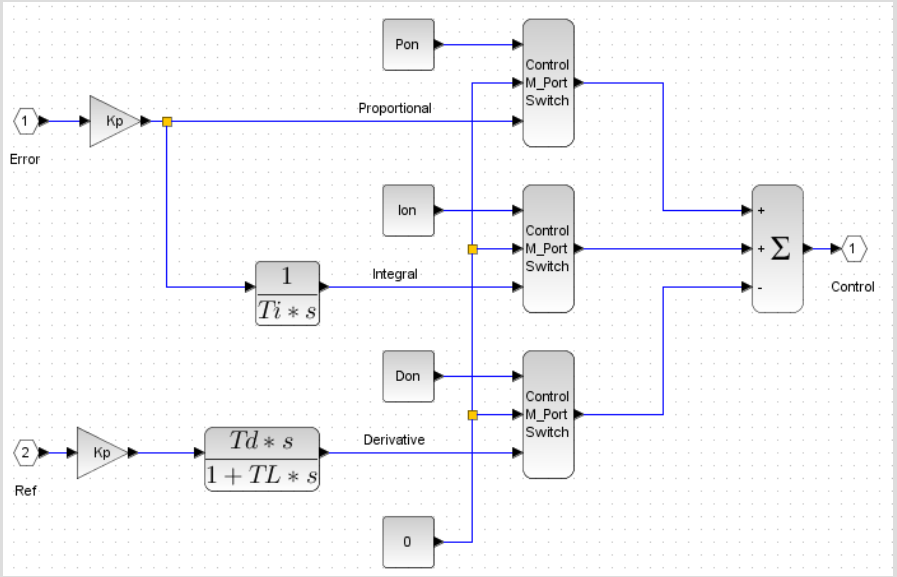


Figure 13: Basic PI_D scheme

Step 24: Numerical results

Running the Xcos scheme at step 21 we obtain a figure in which we can see the behavior of the controller. It is visible that the controller switches after six seconds.

We can even plot the trend of the two angles θ_1 and θ_2 as we did in the previous steps.

Using the command:

```
plot(sol.time, sol.values*180/%pi)
```

we can plot the positions of the angles θ_1 and θ_2. Even in this plot we may see that, after 6 seconds, the rod is taken into its unstable vertical position and then controlled perfectly.

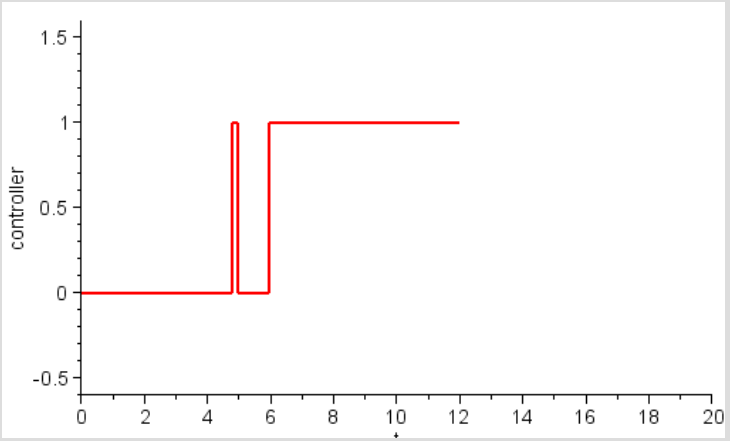


Figure 14:Switch of the controller

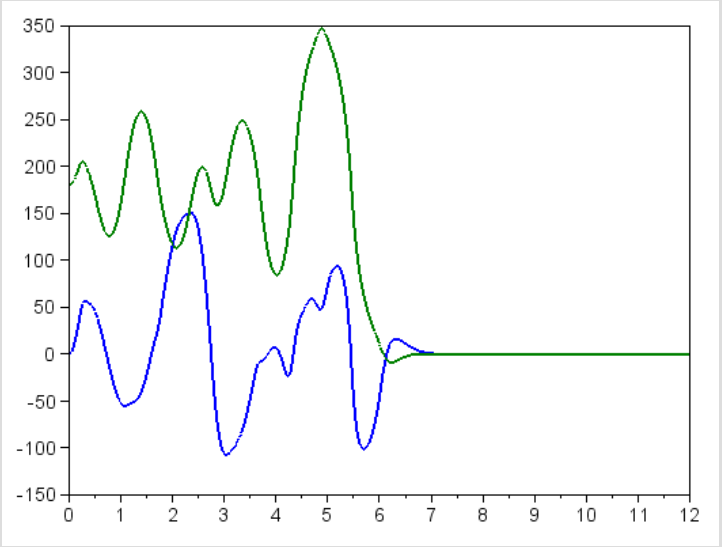


Figure 15: Angles positions at time steps

Step 25: Animations in Scilab

Plotting the animation of the moving pendulum, from its stable state to its vertical position, would be useful in order to understand the entire dynamic of our system. Scilab has the capability to plot animations and in these last steps of the tutorial we will show how to use this capability.

First of all, we need to develop a new Scilab function that, starting for the two angles θ_1 and θ_2 (in radians) as input arguments, plots the pendulum for this configuration. Then, the idea is to store, in the ***user_data*** field of the figure, the handles of the two graphical objects and then update the new coordinates of the rod points. The Scilab code on the right reports only part of the source code of the ***inv_pend_anim.sci*** function.

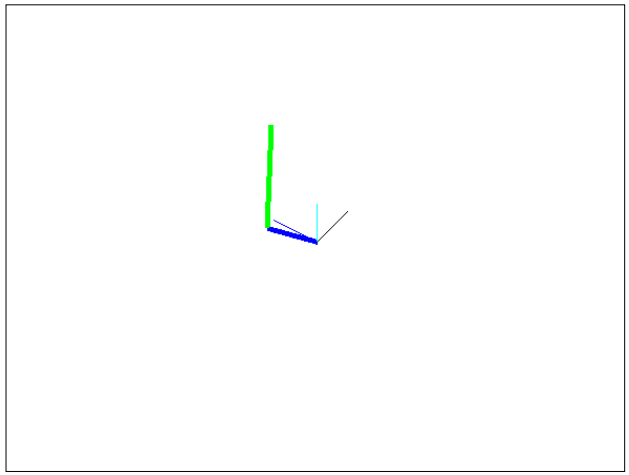


Figure 16: A single frame of the animation. This plot is obtained with the command inv_pend_anim(1.8, 0)

```
function [y1_deg, y2_deg]=inv_pend_anim(theta1, theta2)

y1_deg = theta1*180/%pi;
y2_deg = theta2*180/%pi;

// Model data should be inserted from a mask
l1 = 0.2;
l2 = 0.5;
L = 0.4

f = findobj("Tag", "ANIM_ROD");

if f == [] then
    //Initialization
    //download source code from Openeering.com
else
    scf(f);
end

// Modify coordinates
f = findobj("Tag", "ANIM_ROD");
drawlater();
e1 = f.user_data.e1;
xv1 = [0; l1*cos(theta1)];
yv1 = [0; l1*sin(theta1)];
zv1 = [0; 0];
// xsegs(xv1,yv1,zv1);
e1.data = [xv1, yv1, zv1];

e2 = f.user_data.e2;
xv2 = [l1*cos(theta1); l1*cos(theta1) -
l2*sin(theta2)*cos(%pi/2 - theta1)];
yv2 = [l1*sin(theta1); l1*sin(theta1) +
l2*sin(theta2)*sin(%pi/2 - theta1)];
zv2 = [0; l2*cos(theta2)];
// xsegs(xv2,yv2,zv2);
e2.data = [xv2, yv2, zv2];
drawnow();

endfunction
```

Step 26: Using the animation function in Xcos

To use the animation function as a block in Xcos, it is sufficient to open one of the previous Xcos scheme and add a "***SCIFUNC_BLOCK_M***".

To connect the block with the Scilab function ***inv_pend_anim.sci*** it is sufficient to specify the number and type of input and output ports and the function name as reported in the following figures.

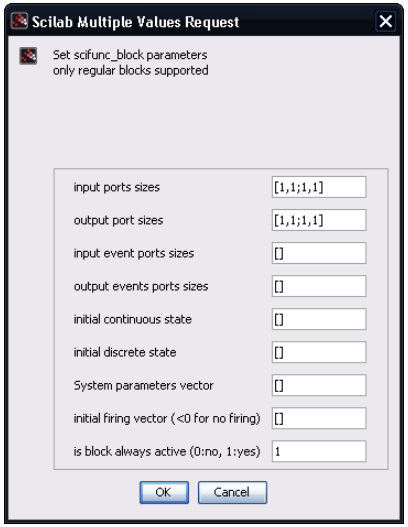


Figure 17: Specifying input and output ports

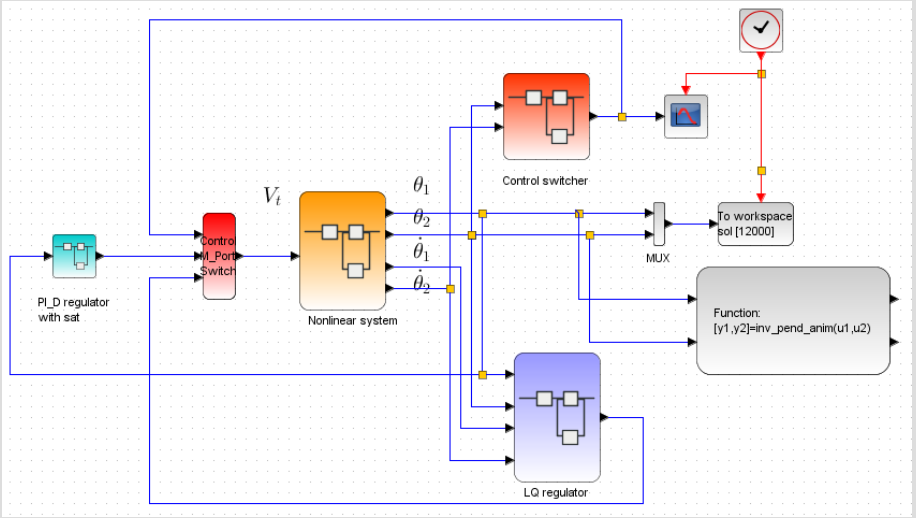


Figure 19: The Xcos scheme with the animation block (inv_pend_se_anim.zcos)

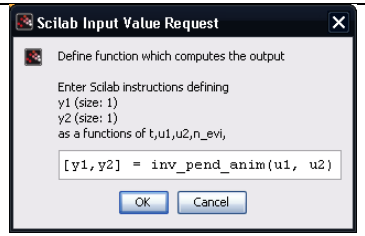


Figure 18: Specifying the Scilab command

Step 27: Running the animation

Before running the Xcos scheme, remember to load the function adding the following command ***exec("inv_pend_anim.sci",-1)*** in the set context menu.

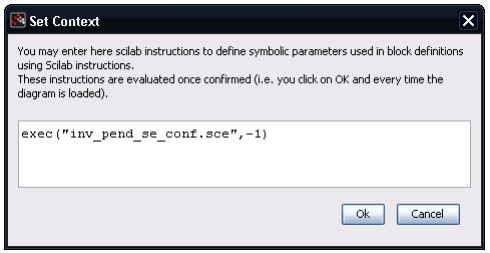
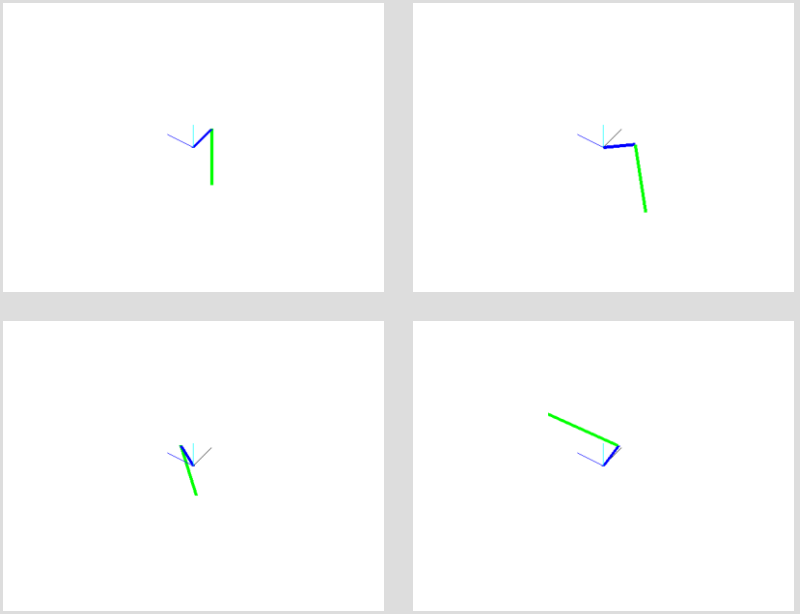


Figure 20: set context window

The animation results are reported on the right.



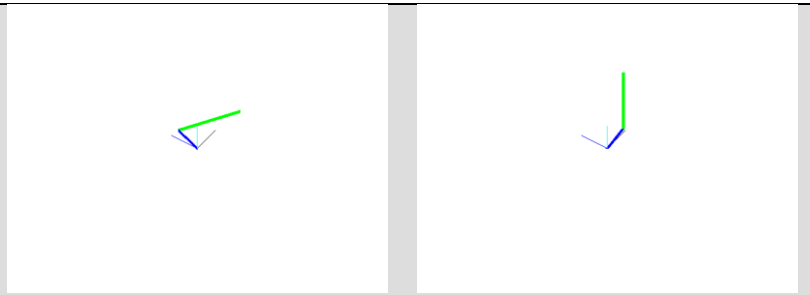


Figure 21: Simulation results: Some images
The entire animation is available for download

Step 28: Exercise 1

Try to implement an unwrapping function that maps, using only Xcos blocks, an input signal into the interval $[0, 2\pi]$ and having π as mean value (it maps 0 into π).

On the right we propose an idea of the scheme and its result.

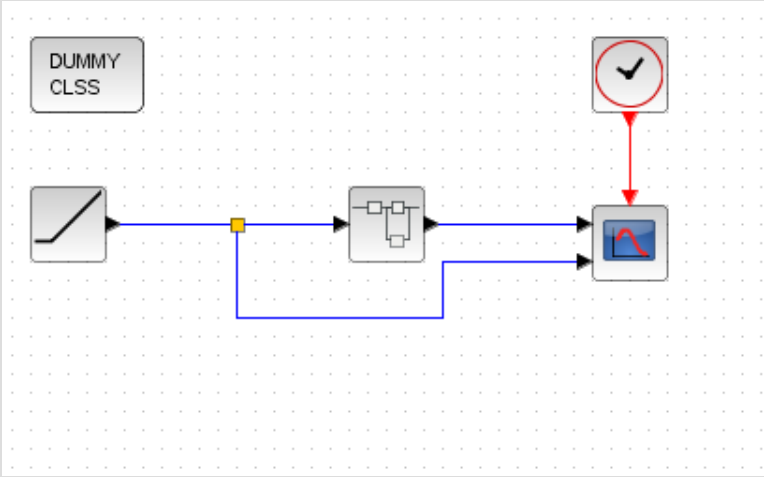


Figure 22: Xcos System (ex1.zcos)

Hints: Use the quantization block.

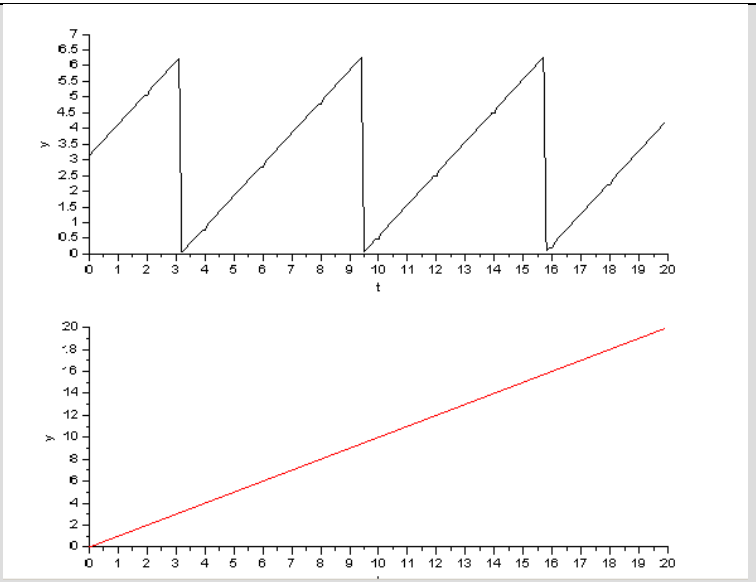


Figure 23: Plots and Results

Step 29: Exercise 2

Starting from noised angle measures try to develop a recovery procedure for the recovery of the derivative, i.e. if we kwon the angle at each time step we want to know the velocity.

The idea is to use a high-pass filter with a unity gain. Try to complete the following scheme adding the appropriate block in the empty block. Solution is given in ex2.zcos.

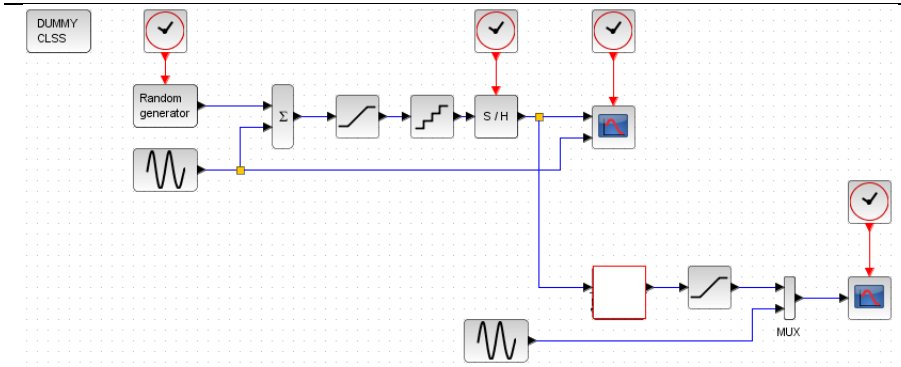


Figure 24: Example of recovery strategy for the derivative of a signal

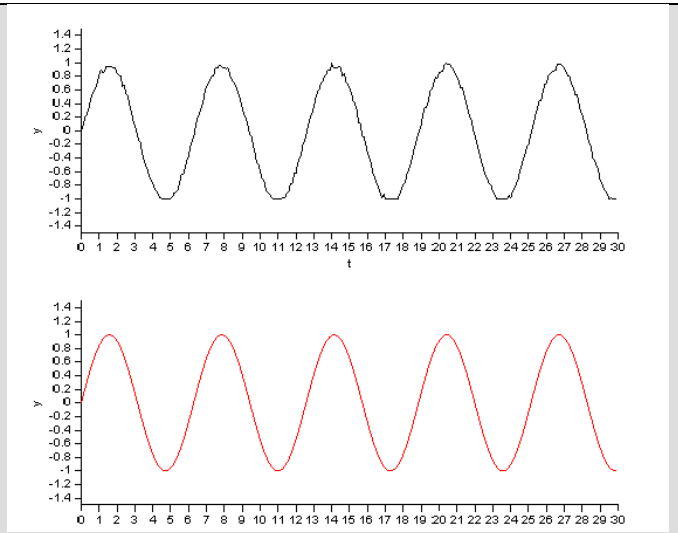


Figure 25: Noise input and original signal

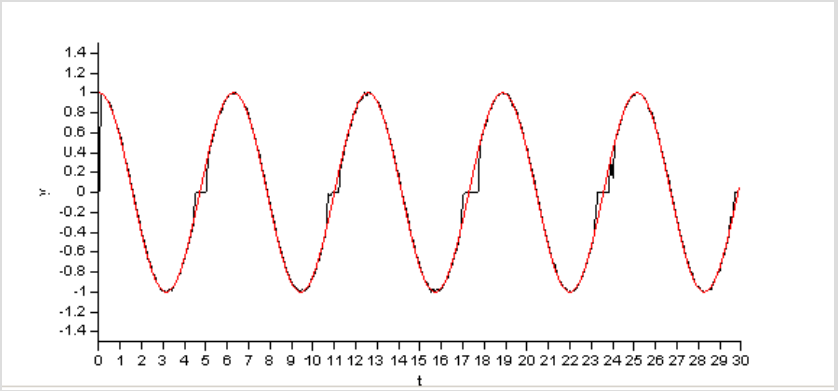


Figure 26: Example of recovery

Step 30: Concluding remarks and References

In this tutorial we have presented a modeling approach for the control of an inverted pendulum in Scilab/Xcos. The regulator is developing using the Control System Toolbox available in Scilab known as CACSD while the simulation is done using Xcos.

All the symbolic manipulations of the expressions and equations have been obtained using Maxima; the software is free and available at [3].

It is trivial that the system is only a study version which can be improved in several ways, for example improving the PID parameters or development a new strategy algorithm.

1. Scilab Web Page: Available: www.scilab.org
2. Openeering: www.openeering.com
3. Maxima: http://maxima.sourceforge.net/

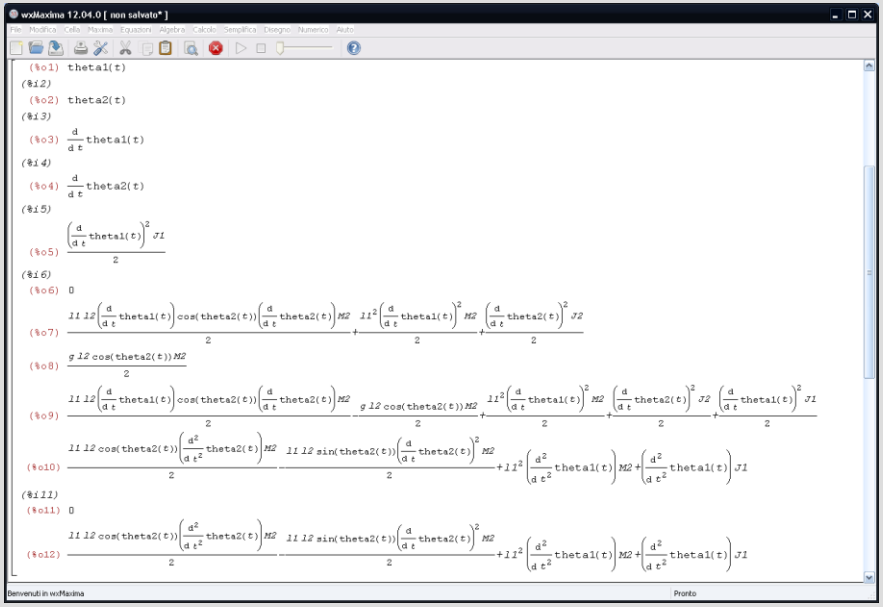


Figure 27: Maxima graphical user interface

Step 31: Software content

```
---------------
Main directory
```

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

www.openeering.com

Thank you for your attention,

Manolo Venturin, Silvia Poles

```
--------------
Animation.png                : Animation of the inv. pend.
ex1.zcos                     : Solution of the ex. 1
ex2.zcos                     : Solution of the ex. 2
inv pend.zcos                : Inv. pend. in Xcos
inv pend anim.sci            : Inv. pend. animation
inv_pend_anim.zcos           : Inv. pend. with animation in Xcos
inv_pend_ctrl.zcos           : Inv. pend. with regulator
inv_pend_ctrl_anim.zcos      : Inv. pend. with anim. and regulator
inv pend ganim.sce           : Post-processing animation
inv pend mode.sci            : Inv. pend. mode function
inv pend nl scilab.sce       : Inv. pend. in Scilab
inv_pend_se.zcos             : Inv. pend. self-erecting
inv_pend_se_anim.zcos        : Inv. pend. self-erecting with anim.
inv_pend_se_conf.sce         : Inv. pend. self-erecting conf.
inv pend se data             : Inv. pend. self-erecting solut.
invpend_check.sce            : Inv. pend. development
license.txt                  : The license file
system_maxima.txt            : Maxima scripts
```