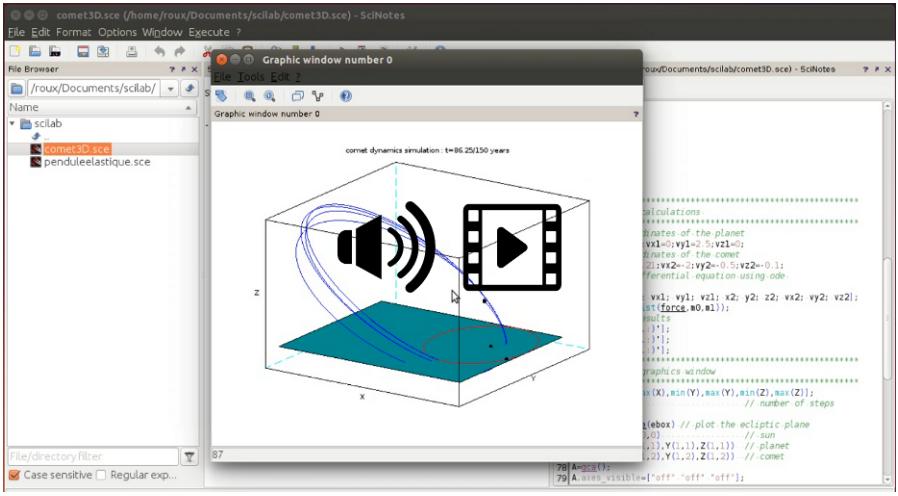
1

Preview of Scilab

Scilab is a numerical computation software that is open source, multi-platform and is meant for the theoretician as well as the engineer. It possesses a high level computer language that is well adapted to mathematical notation, as well as complementary modules targeted at scientific applications. These applications include signal and image processing, statistical analysis, mechanical simulations, fluid dynamics, numerical optimization, modeling and simulation of hybrid dynamic systems, etc. To demonstrate its capacities, we showcase an example of a comet in orbit around the Sun, following a trajectory perturbed by the presence of a planet in a stable orbit around the Sun. Despite the complexity of the problem, you can easily simulate such a system with Scilab and obtain a three dimensional animation of the movement of the different bodies. An example of this video can be seen below.

Figure 1.1 : *Simulation of a comet's orbit (vidéo)*



This animation was obtained using a program written in the Scilab language and presented in the following Example 1.1.

Example 1.1 : *Script used to visualize the trajectory of a comet around the Sun*

```
//*******************************************************
// simulation of the perturbed trajectory of a comet
//*******************************************************
//  parameterization of a sphere
function [x,y,z]=sphere(theta,phi)
   A=0.1,B=0.01
   x=A*cos(phi).*cos(theta)
   y=A*cos(phi).*sin(theta)
   z=B*sin(phi)
endfunction
// function to draw a sphere
function plot_sphere(x,y,z)
   phi=[0:0.1:2*3.15];
   theta=[2*3.15:-0.05:0];
   [dx,dy,dz]=eval3dp(sphere,theta,phi);
   surf(x+dx,y+dy,z+dz);
endfunction
// function to plot the z=0 plane
function plot_ecliptic(ebox)
   x=[ebox(1);ebox(2)]
   y=[ebox(3);ebox(4)]
   z=zeros(2,2)
   surf(x,y,z)
endfunction

// functions calculating the gravitational forces
function [u2]=force_g(t,u,mass)
   module=-G*mass*((u(1)^2+u(2)^2+u(3)^2)^(-3/2))
   u2=[module*u(1); module*u(2); module*u(3)]
endfunction

function [du]=force(t,u,mass0,mass1)
   u1=[u(1);u(2);u(3)]
   du1=[u(4);u(5);u(6)]
   u2=[u(7);u(8);u(9)]
   du2=[u(10);u(11);u(12)]
   ddu1=force_g(t,u1,mass0)
   ddu2=force_g(t,u2,mass0)+force_g(t,u2-u1,mass1)
   du=[du1;ddu1;du2;ddu2]
endfunction

// constants
G=0.04;
m0=1000;
m1=1;
dt=0.05;
T=50;
dx=0.5;
dy=0.5;
dz=0.5;
alpha=65;
Beta=150;
```

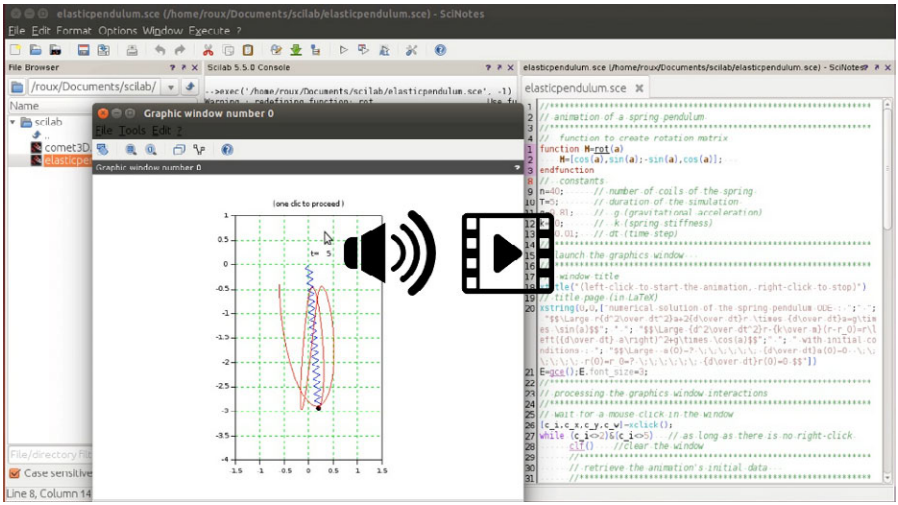
```
//*********************************************************
// trajectory calculations
//*********************************************************
// initial coordinates of the planet
x1=5;y1=0;z1=0;vx1=0;vy1=2.5;vz1=0;
// initial coordinates of the comet
x2=6;y2=6;z2=0.21;vx2=-2;vy2=-0.5;vz2=-0.1;
//solve the differential equation using ode
t=[0:dt:T];
u0=[x1; y1; z1; vx1; vy1; vz1; x2; y2; z2; vx2; vy2; vz2];
u=ode(u0,0,t,list(force,m0,m1));
// retrieve results
X=[u(1,:)',u(7,:)'];
Y=[u(2,:)',u(8,:)'];
Z=[u(3,:)',u(9,:)'];

//*********************************************************
// launch the graphics window
//*********************************************************
ebox=[min(X),max(X),min(Y),max(Y),min(Z),max(Z)];
N=length(t);                        // number of steps
drawlater()
plot_ecliptic(ebox) // plot the ecliptic plane
plot_sphere(0,0,0)                  // sun
plot_sphere(X(1,1),Y(1,1),Z(1,1))  // planet
plot_sphere(X(1,2),Y(1,2),Z(1,2))  // comet
A=gca();
A.axes_visible=["off" "off" "off"];
A.rotation_angles=[alpha Beta];
A.data_bounds=ebox;
drawnow()

//*********************************************************
// main loop creates the graphical animation
//*********************************************************
for k=1:5:N
   Beta=Beta+k/300;                        //  view angle
   realtimeinit(0.05)                     // unit of time
   drawlater()              // open the graphical buffer
   clf()                   // erase the graphical buffer
   plot_ecliptic(ebox)        // plot on ecliptic plane
   param3d1(X(1:k,:),Y(1:k,:),...          // display the
   list(Z(1:k,:),[5,2]))                  // trajectories
   plot_sphere(0,0,0)                          // the sun
   plot_sphere(X(k,1),Y(k,1),Z(k,1))        // the planet
   plot_sphere(X(k,2),Y(k,2),Z(k,2))         // the comet
   title('comet dynamics simulation : t='+msprintf(...
   '%2.2f',t(k))+'/'+string(T)+' years')         // title
   xinfo(string(t(k)))                    // display time
   A=gca();                // resize the graphics window
   A.axes_visible=["off" "off" "off"];
   A.rotation_angles=[alpha Beta];    // rotate pt of vue
   A.data_bounds=ebox;
   drawnow()                 // display graphical buffer
   realtime(k)            // pause to adjust display rate
end
```

This example provides a small preview of Scilab's capabilities pertaining to graphics as well as numerical computation. With a small degree of skill, you will rapidly be able to go beyond this level. For example, you may then create programs that allow the user to interact with Scilab's graphical interface and create his/her own animations such as those shown in the video below.

Figure 1.2 : *Simulation of a pendulum hung from a spring (video)*



In this example, we simulate the trajectory of a pendulum hung from a spring (of stiffness k), oscillating freely after being released from a given position with no initial velocity. The pendulum's motion is modeled through a system of differential equations involving two variables:

$$\begin{cases} r \times \frac{d^2a}{dt^2} + 2\frac{dr}{dt} \times \frac{da}{dt} = g \times \sin(a) \\ \frac{d^2r}{dt^2} - \frac{k}{m}(r - r_0) = r \times \left(\frac{da}{dt}\right)^2 + g \times \cos(a) \end{cases}$$

where:

- a is the angle of the pendulum with respect to the vertical
- r is the length of the spring that constitutes the pendulum

and with the initial conditions: $a(0) = a_0$, $\frac{da(0)}{dt} = 0$, $r(0) = r_0$, $\frac{dr(0)}{dt} = 0$.

In order to study a system like this that is very sensitive to initial conditions, it is important to be able to easily modify the initial position and observe the effect of each parameter on the movement. With Scilab, it is possible to create a graphical interface that simplifies the input of parameters and lets the user focus on the analysis of results. With the script shown in Example 1.2, you can modify the initial position of the pendulum with a simple mouse click in the graphics window and study its movement.

Example 1.2 : *Script used to modify the pendulum's initial position*

```
//*****************************************************************
// animation of a spring pendulum
//*****************************************************************
//  function to create rotation matrix
function M=rot(a)
    M=[cos(a),sin(a);-sin(a),cos(a)];
endfunction
//  constants
n=40;       // number of coils of the spring
T=5;        // duration of the simulation
g=9.81;     //  g (gravitational acceleration)
k=10;       //  k (spring stiffness)
dt=0.01;    // dt (time step)

//*****************************************************************
// launch the graphics window
//*****************************************************************
//  window title
xtitle("(left-click to start the animation, right-click to stop)")
// title page (in LaTeX)
titlepage(["numerical solution of the spring pendulum ODE : ";" "; "$
$\Large r{d^2\over dt^2}a+2{d\over dt}r \times {d\over dt}a=g\times
 \sin(a)$$";
" "; "$$\Large {d^2\over dt^2}r-{k\over m}(r-r_0)=r\left({d\over dt}
 a\right)^2+g\times \cos(a)$$";" "; " with initial conditions : "; "$$
\Large  a(0)=? \;\;\;\;\;\; {d\over dt}a(0)=0  \;\;\;\;\;\; r(0)=r_0=?
 \;\;\;\;\;\; {d\over dt}r(0)=0 $$"])

//*****************************************************************
// processing the graphics window interactions
//*****************************************************************
// wait for a mouse click in the window
[c_i,c_x,c_y,c_w]=xclick();
while (c_i<>2)&(c_i<>5)   // as long as there is no right-click
      clf()    //clear the window
      //*************************************************************
      // retrieve the animation's initial data
      //*************************************************************
      // window title
      xtitle("(one click to initialize pendulum position a(0) and
 r(0) )")
      // parameterize the window's Axes handle
      plot(0,0,'.k');A=gca();A.x_location="origin";
```

```
A.y_location="origin";A.auto_scale="off";A.isoview="on";
A.data_bounds=[-1 -1; 1,0];xgrid(3)
// retrieve the pendulum's initial position coordinates :
[c_i,x,y,c_w]=xclick();
// compute initial values :
a=sign(x)*abs(atan(x/y));a0=a;da=0;  // compute angle a(0)
l=sqrt(x^2+y^2);r=l;,dr=0;           //  compute r(0)
// adapt the window's size to the pendulum's maximum size :
A.data_bounds=[-1.5,-max(4*l,4);1.5,max(l,0.5)];
//*********************************************************
// loop creates the animation
//*********************************************************
for t=0:dt:T
     //*****************************************************
     // compute new positions
     //*****************************************************
     // solve the differential equation for a and r using
     // Euler's method
     dda=-(g*sin(a)+2*dr*da)/r;
     ddr=r*(da)^2-k*(r-l)+g*cos(a);
     da=da+dt*dda;
     dr=dr+dt*ddr;
     a=a+dt*da;
     r=r+dt*dr;
     // compute the spring's line representation
     springr=linspace(0,r,n)';        // the spring stretches
     // coordinates transverse to spring's axis -> /\/\/\
     springa=[0;(-1).^[0:n-3]';0]*(l/10);
     //rotate the spring's picture by the angle a
     x=[x;r*sin(a)];
     y=[y;-r*cos(a)];
     M=-rot(-a);
     N=[springr,springa]*M;
     springy=N(:,1);springx=N(:,2);
     //*****************************************************
     // display the pendulum
     //*****************************************************
     drawlater()  // write to the graphics buffer
     clf()        // clear the window
     plot(springx,springy) //display the pendulum's spring
     xstring(0,0.1,["t=" string(t)]) // elapsed time
     // pendulum bob :
     xfarc(r*sin(a)-0.05,-r*cos(a)+0.05,0.1,0.1,0,360*64)
     // resize the graphics window
     A=gca();A.data_bounds=[-1.5,-max(4*l,4);1.5,max(l,0.5)];
     A.auto_scale="off";A.isoview="on";
     A.axes_visible=["off" "off" "off"];
     drawnow()            // display the graphics buffer
     realtime(t);           // delay display
end
//*********************************************************
// choose a new animation or exit program
//*********************************************************
xtitle("(one clic to proceed )")    // window title
plot(x,y,'-r')                       // display trajectory
A=gca();A.isoview="on";xgrid(3); // display grid (green)
```

```
        // waiting for a mouse click in graphics window :
        [c_i,x,y,c_w]=xclick();
        clf();                          // choose a new operation
        xtitle("(left-click to start the animations, right-click to
    stop)")
        plot(0,0,'.k');A=gca();A.x_location="origin";
        A.y_location="origin";
        // waiting for a mouse click in the window :
        [c_i,x,y,c_w]=xclick();
end
```

These two examples require knowledge of basic Scilab concepts that we are going to introduce in the rest of this book. Once you are done reading, you will be fully able to recreate them yourself. We will demonstrate this in the last chapter Two Case Studies: a Pendulum and Comet Orbit.

Caution › *To make sure the above scripts are properly executed, use Scilab version 5.4.1 or later.*