Introduction to ODEs in Scilab

Aditya Sengupta

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

April 15th, 2010, Smt. Indira Gandhi College of Engineering

Scilab can be used to model and simulate a variety of systems, such as:

1. Ordinary Differential Equations
2. Boundary Value Problems
3. Difference Equations
4. Differential Algebraic Equations

We will deal with Ordinary Differential Equations in this talk.

Scilab can be used to model and simulate a variety of systems, such as:

1. Ordinary Differential Equations
2. Boundary Value Problems
3. Difference Equations
4. Differential Algebraic Equations

We will deal with Ordinary Differential Equations in this talk.

We will do two things:

1. Model the ODE in a way Scilab can understand.
2. Solve the system for a given set of initial values.

Modeling the system

We will model the system as a first-order equation:

$$\dot{y} = f(t, y)$$

Note: Scilab tools assume the differential equation to have been written as first order system.
Some models are initially written in terms of higher-order derivatives, but they can always be rewritten as first-order systems by the introduction of additional variables

Modeling the system

We will model the system as a first-order equation:

$$\dot{y} = f(t, y)$$

Note: Scilab tools assume the differential equation to have been written as first order system.

Some models are initially written in terms of higher-order derivatives, but they can always be rewritten as first-order systems by the introduction of additional variables

Modeling the system

We will model the system as a first-order equation:

$$\dot{y} = f(t, y)$$

Note: Scilab tools assume the differential equation to have been written as first order system.
Some models are initially written in terms of higher-order derivatives, but they can always be rewritten as first-order systems by the introduction of additional variables

Example

Let us consider the simple system:

$$\frac{dx}{dt} = sin(2t)$$

We can model this system using this code:

```
function dx = f(t, x)
    dx = sin(2*t);
endfunction
```

Example

Let us consider the simple system:

$$\frac{dx}{dt} = sin(2t)$$

We can model this system using this code:

```
function dx = f(t, x)
    dx = sin(2*t);
endfunction
```

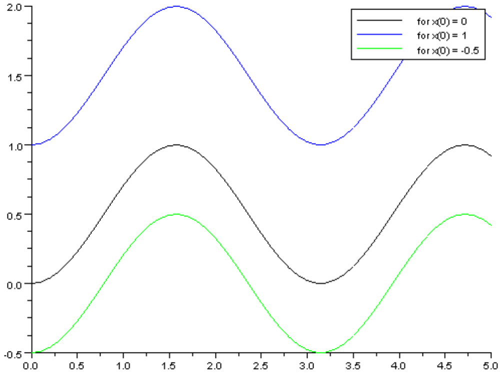
Solution

We know that the solution is *supposed* to be

$$x = -\frac{1}{2}cos(2t) + c$$

where c is a constant that depends on the initial value of the problem.

Depending on the initial value, the plot will look like this:



Solving an ODE in Scilab

The simulation tool we will use for solving ODEs in Scilab is the `ode` function
The simplest calling sequence for `ode` is:

```
y=ode(y0, t0, t, f)
```

where `y0` is the initial value at `t0` and `t` contains the points in time at which the solution is to be determined. `f` is the function corresponding to

$$\dot{y} = f(t, y)$$

For our example, we will take the initial value to be `y0 = -0.5` at `t0 = 0`.

Let us evaluate the ODE from `t = 0:0.1:5`.

The code is:

```
t0 = 0
x0 = -0.5
t = 0:0.1:5;
x = ode(x0, t0, t, f);
plot2d(t, x)
```

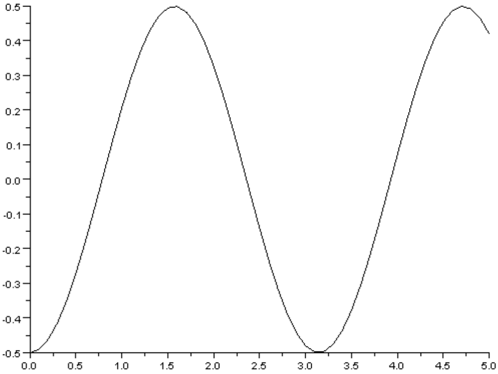
For our example, we will take the initial value to be `y0 = -0.5` at `t0 = 0`.

Let us evaluate the ODE from `t = 0:0.1:5`.

The code is:

```
t0 = 0
x0 = -0.5
t = 0:0.1:5;
x = ode(x0, t0, t, f);
plot2d(t, x)
```

You should get a graph that looks like this:



Higher Order Derivatives

When we have ODEs formulated in terms of higher order derivatives, we need to rewrite them as first-order systems. We do this by using variables to fill in the intermediate order derivaties.

For example, let us consider the system:

$$\frac{d^2y}{dt^2} = sin(2t)$$

whose one solution we can easily guess to be $y = -(1/4)sin(2t)$

Higher Order Derivatives

When we have ODEs formulated in terms of higher order derivatives, we need to rewrite them as first-order systems. We do this by using variables to fill in the intermediate order derivaties. For example, let us consider the system:

$$\frac{d^2 y}{dt^2} = sin(2t)$$

whose one solution we can easily guess to be $y = -(1/4)sin(2t)$

We convert the second order equation into two first order equations:

$$dy/dt = z$$

$$dz/dt = sin(2t)$$

Therefore, we have the ode in the form:

$$dx/dt = f(t,x)$$

where dx and x are vectors:

$$x = [z; sin(2t)]$$

$$dx = [dy/dt; dz/dt]$$

We then proceed to replace z, dy/dt, and dz/dt with vector components x(2), dx(1), and dx(2)

We convert the second order equation into two first order equations:

$$dy/dt = z$$

$$dz/dt = sin(2t)$$

Therefore, we have the ode in the form:

$$dx/dt = f(t, x)$$

where dx and x are vectors:

$$x = [z; sin(2t)]$$

$$dx = [dy/dt; dz/dt]$$

We then proceed to replace z, dy/dt, and dz/dt with vector components x(2), dx(1), and dx(2)

We convert the second order equation into two first order equations:

$$dy/dt = z$$

$$dz/dt = sin(2t)$$

Therefore, we have the ode in the form:

$$dx/dt = f(t, x)$$

where dx and x are vectors:

$$x = [z; sin(2t)]$$

$$dx = [dy/dt; dz/dt]$$

We then proceed to replace z, dy/dt, and dz/dt with vector components x(2), dx(1), and dx(2)

We model the system thus:

```
function dx = f(t, x)
    dx(1) = x(2)
    dx(2) = sin(2*t)
endfunction
```

and simulate the ODE thus:

```
t = 0:0.01:4*%pi;

y=ode([0; -1/2], 0, t, f);
// Note the importance of giving correct starting values. Try to put alternate starting values and see the difference.

plot2d(t',[y(1,:)' y(2, :)'])
// The curve in black is the final solution. The other curve is for illustration-- to show the intermediate step.
```

We model the system thus:

```
function dx = f(t, x)
    dx(1) = x(2)
    dx(2) = sin(2*t)
endfunction
```

and simulate the ODE thus:

```
t = 0:0.01:4*%pi;

y=ode([0; -1/2], 0, t, f);
// Note the importance of giving correct starting values. Try to put alternate starting values and see the difference.

plot2d(t',[y(1,:)' y(2, :)'])
// The curve in black is the final solution. The other curve is for illustration - to show the intermediate step.
```

Root Finding

Sometimes- we just want to simulate a differential equation up to the time that a specific event occurs.

For example, an engine being revved until it reaches a particular speed- after which the gear is to be changed.

For such circumstances, we need to define a quantity that signals the occurance of the event.

Root Finding

Sometimes- we just want to simulate a differential equation up to the time that a specific event occurs.

For example, an engine being revved until it reaches a particular speed- after which the gear is to be changed.

For such circumstances, we need to define a quantity that signals the occurance of the event.

Root Finding

Sometimes- we just want to simulate a differential equation up to the time that a specific event occurs.
For example, an engine being revved until it reaches a particular speed- after which the gear is to be changed.
For such circumstances, we need to define a quantity that signals the occurance of the event.

In Scilab we use the `ode_root` function, which is called thus:

```
[y, rd] = ode("root", y0, t0, t, f, ng, g)
```

where `g` is a function that becomes zero valued when the constraining event occurs and `ng` is the size of `g`.
`rd` is a vector that contains the stopping time as its first element.

In Scilab we use the `ode_root` function, which is called thus:

```
[y, rd] = ode("root", y0, t0, t, f, ng, g)
```

where `g` is a function that becomes zero valued when the constraining event occurs and `ng` is the size of `g`.
`rd` is a vector that contains the stopping time as its first element.

Example

Let us consider the example of the engine that is revved. We wish to constrain the revving of the engine till it reaches a certain point.

We build a first order approximation of an engine using the following code (call it engine.sci):

```
function d_revs = engine(t, revs)
    d_revs = %e^(-revs)
endfunction
```

We can simulate the behaviour of the engine when it is unconstrained using the following code:

```
exec engine.sci
revs = ode(0, 0, 0:0.1:10, engine);
plot2d(0:0.1:10, revs)
```

Example

Let us consider the example of the engine that is revved. We wish to constrain the revving of the engine till it reaches a certain point. We build a first order approximation of an engine using the following code (call it engine.sci):

```
function d_revs = engine(t, revs)
    d_revs = %e^(-revs)
endfunction
```

We can simulate the behaviour of the engine when it is unconstrained using the following code:

```
exec engine.sci
revs = ode(0, 0, 0:0.1:10, engine);
plot2d(0:0.1:10, revs)
```

Example

Let us consider the example of the engine that is revved. We wish to constrain the revving of the engine till it reaches a certain point. We build a first order approximation of an engine using the following code (call it engine.sci):

```
function d_revs = engine(t, revs)
    d_revs = %e^(-revs)
endfunction
```

We can simulate the behaviour of the engine when it is unconstrained using the following code:

```
exec engine.sci
revs = ode(0, 0, 0:0.1:10, engine);
plot2d(0:0.1:10, revs)
```

We then write the constraining function (call it gearbox.sci):

```
function stop = gearbox(t, revs)
    stop = 1.5 - revs //We choose to stop the engine when
        the revs reach the value 1.5 (You can choose any
        other value)
endfunction
```

We then simulate the behaviour of the engine when it is constrained as above.

```
exec engine.sci
exec gearbox.sci
[revs, stop_time]= ode("root", 0, 0, 0:0.1:10, engine, 1,
    gearbox);
plot2d([0:0.1:stop_time(1), stop_time(1)], revs)
```

We then write the constraining function (call it gearbox.sci):

```
function stop = gearbox(t, revs)
    stop = 1.5 - revs //We choose to stop the engine when the revs reach the value 1.5 (You can choose any other value)
endfunction
```

We then simulate the behaviour of the engine when it is constrained as above.

```
exec engine.sci
exec gearbox.sci
[revs, stop_time]= ode("root", 0, 0, 0:0.1:10, engine, 1, gearbox);
plot2d([0:0.1:stop_time(1), stop_time(1)], revs)
```

Compare the two graphs- can you see where the simulation was halted in the second case?

Linear Systems

Since they appear so often, there are special functions for modeling and simulating linear systems.
For instance, you can create a linear system thus:

```
s = poly(0, 's')
sys = syslin('c', 1/(s+1))
```

and simulate it thus:

```
t = 0:0.1:10;
y = csim('step', t, sys);
plot2d(t, y);

z = csim(sin(5*t), t, sys);
plot2d(t, z);

bode(sys, 0.01, 100);
```

Linear Systems

Since they appear so often, there are special functions for modeling and simulating linear systems.

For instance, you can create a linear system thus:

```
s = poly(0, 's')
sys = syslin('c', 1/(s+1))
```

and simulate it thus:

```
t = 0:0.1:10;
y = csim('step', t, sys);
plot2d(t, y);

z = csim(sin(5*t), t, sys);
plot2d(t, z);

bode(sys, 0.01, 100);
```

Now try to:

- Model and simulate your own systems
- Use the help command to find more options

Thanks