Stiffness, ODE23s, ODE15s

Let's see the important notion of stiffness by running ode45, the primary MATLAB ODE solver, on our flame example.

The differential equation is

$$y' = y^2 - y^3$$

```
F = @(t,y) y^2-y^3
```

We need to choose a fairly -- an extremely small initial condition to the minus sixth.

```
y0=1.e-6
```

The final value of t is 2 over y0

```
tfinal = 2/y0
```

And we are going to impose a modest accuracy requirement

```
opts = odeset('reltol',1.e-5)
```

Now let's run ode45 with its default output.

```
ode45(F,[0,tfinal],y0,opts)
```

We can see it takes very small steps around $y = 1$.

Stiffness is an efficiency issue. It's doing what we asked for. It's meeting the accuracy requirements, but it's having to take very small steps to do it.

Let's try another ODE solver -- ode23

```
ode23(F,[0,tfinal],y0,opts)
```

It also taking very small steps for the same reason.

Let's see a new solver -- ode23s. 's' for stiffness.

```
ode23s(F,[0,tfinal],y0,opts)
```

This was designed to solve stiff problems. It just a few steps to get to the final result.

Before we see how ode23 works, let's try to define stiffness firstly. It's a qualitative notion that does not have a precise mathematical definition. It depends upon the problems, but also on the solver and the accuracy requirements.

Stiffness: A problem is stiff if the solution being sought varies slowly, but there are nearly solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results.

Implicit Euler's Method

$$y_{n+1} - hf(t_{n+1}, y_{n+1}) = y_n$$

This formula, it defines y_{n+1}, but does not tell us how to compute it. We have to solve this equation for y_{n+1}. And we are not going to go into detail about how we actually do, because it involves something like a Newton method that would requires knowing the derivative, or an approximation to the derivative of f.

The stiff solver, ode23s, using an implicit second-order formula and an associated third-order error estimator. It evaluates the partial derivatives of f with respect to the t, y at each step, so that's expensive. It's efficient at crude error tolerances, like graphic accuracy. And it has relatively low overhead.

By way of comparison, the stiff solver ode15s can be configured to use either the Variable Order Numerical Differentiation Formula, NDF, or the related to backward differentiation formula BDF. Neither case it saves several values of the function over previous steps. The order varies automatically between one and five, it evaluates the partial derivatives less frequently and it see efficient at higher tolerances than 23s.