Stiff problems

Matlab's workhorse ODE solver `ode45` does a good job of computing approximate solutions for a wide range of ODE systems. However, for certain problems it works very inefficiently. Matlab has several ODE-solving functions (`ode15s`, `ode23s`, `ode23t`, `ode23tb`—all implicit methods) that are specially designed to deal efficiently with these so-called "stiff" problems.

Examples

Simple model of flame growth

Suppose you light a match. This produces a ball of flame which grows rapidly until it reaches a certain size; it then remains at this critical size as the amount of oxygen being consumed by combustion inside the ball balances the amount available through the surface of the ball. A simplified scaled model of the process is

$$\frac{du}{dt} = u^2 - u^3, \qquad u(0) = u_0$$

where u represents the radius of the ball of flame relative to the critical radius. The u^2 and u^3 terms come from the surface area and volume. We consider the solution of this problem over a length of time inversely proportional to u_0.

For $u_0 = 0.01$, 0.001 and 0.0001, use `ode45` with relative tolerance 10^{-4} to compute the solution from $t = 0$ to $t = 2/u_0$. What do you observe?
Try computing the solution with $u_0 = 0.0001$ using `ode23s` and `ode15s` instead.

Gear's example

The linear first-order scalar ODE $u'(t) = \lambda\big(u - g(t)\big) + g'(t)$ with initial condition $u(0) = u_0$ has an exact solution given by $u(t) = \big(u_0 - g(0)\big)e^{\lambda t} + g(t)$. Taking $\lambda = -100$ and $g(t) = t$, the ODE becomes $u'(t) = -100(u - t) + 1$, and the exact solution is $u(t) = u_0\, e^{-100t} + t$. What do you expect the solution for $t > 0$ to look like?
Use `ode45` with default tolerances to compute the solution from $t = 0$ to $t = 100$, starting from the initial condition $u(0) = 1$. How many time steps did the code use? What is the average step size?
Compute the solution using `ode15s` instead, and compare the number of steps and average step size.

Robertson's chemical reaction model

Robertson proposed the following equations to model a certain reaction between three chemicals X, Y and Z:

$$\begin{aligned}
\frac{dx}{dt} &= -0.04x + 10^4 yz \\
\frac{dy}{dt} &= 0.04x - 10^4 yz - 3 \times 10^7\, y^2 \\
\frac{dz}{dt} &= 3 \times 10^7\, y^2
\end{aligned}$$

The dependent variables x, y and z represent the concentrations of chemicals X, Y and Z.

Use `ode45` with default tolerances to compute the solution from $t = 0$ to $t = 3$, starting from the initial conditions $x(0) = 1$, $y(0) = 0$, $z(0) = 0$. Plot the graph of $y(t)$. How many time steps did the code use? What is the average step size?
Compute the solution using `ode15s` instead, and compare the number of steps and average step size.

From these examples it can be seen that stiff problems often arise when there are *multiple time scales* in the physical process being modelled by the differential equation(s).

Detecting events

Sometimes we may need to determine the time(s) of occurrence of some special "event" in the solution of an ODE system; often we would even want to terminate the computation once this event is detected. For example, if you have an ODE system that models the trajectory of an object moving under the influence of gravity, you would be interested in the time at which the object hits the ground, and you would not want to keep computing the (spurious) solution after that time.
With the Matlab ODE solvers, you can turn on event detection by specifying

```
'Events',@<function name>  in  options=odeset(...)
```

`<function name>` refers to a function that defines the event. It has three output arguments:

`value`	is a formula whose zeros correspond to occurrences of the event
`isterminal`	is set to 1 if you want the computation is to terminate at the first zero of `value`; set it to 0 if you want the computation to continue until the end of `tspan`, detecting any further zeros of `value` if they exist
`direction`	specifies the direction of zero-crossing in order for a zero of `value` to be registered as an event: `direction=1` means that `value` must be increasing through zero; `direction=-1` means that `value` must be decreasing through zero; `direction=0` allows for any type of zero

With event detection turned on, the output of the ODE solver will take the form `[tout,uout,te,ue]`, where `te` is an array of event times detected and `ue` contains the corresponding u-values.

Examples

A pursuit problem
Let $\mathbf{R}(t)$ describe the path taken by a rabbit running in a meadow. A fox chases the rabbit in such a way that its velocity is directed along the vector from its current position to the position of the rabbit, with magnitude being a constant times the speed of the rabbit. The path $\mathbf{F}(t)$ of the fox is then determined by the ODE

$$\frac{d}{dt}\mathbf{F}(t) = k\|R'(t)\|_2 \frac{\mathbf{R}(t) - \mathbf{F}(t)}{\|\mathbf{R}(t) - \mathbf{F}(t)\|_2}$$

This is in fact a system of two ODEs, because each of $\mathbf{F}(t)$ and $\mathbf{R}(t)$ have two components.
Suppose that

$$\mathbf{R}(t) = \begin{pmatrix} \sqrt{1+t}\ \cos t \\ \sqrt{1+t}\ \sin t \end{pmatrix} \qquad \text{and} \qquad \mathbf{F}(0) = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$$

If $k = 0.8$, does the fox catch the rabbit? What about if $k = 1.1$?

Period of a periodic solution
Some ODE systems have periodic solutions, such that $\mathbf{u}(t+p) = \mathbf{u}(t)$ for some non-zero constant p and all values of t. The minimal positive value of p is called the period. For the simple harmonic oscillator $mx'' + kx = 0$, we can solve the equation exactly and find that all nontrivial solutions are periodic with period $\frac{2\pi}{\sqrt{k/m}}$. For nonlinear systems such as the predator–prey equations

$$\begin{aligned} \frac{dV}{dt} &= rV - aVP \\ \frac{dP}{dt} &= -sP + baVP \end{aligned}$$

the periods of periodic solutions are usually difficult or impossible to determine exactly, but they can be estimated by using event detection with a Matlab ODE solver.