Numerical solution of ODEs: adaptive algorithms and Matlab's ODE solvers

So far, all the programs we have written perform time-stepping with a fixed step size h. However, the numerical methods implemented in modern software packages are mostly *adaptive* algorithms where, at each step, h is adjusted based on an estimate of the error at that step.

The error estimation formulas that adaptive algorithms rely on are usually obtained by comparing expressions for the error (derived from Taylor series expansions) associated with different methods.

For example, Matlab's built-in Runge–Kutta solver `ode23` uses an error estimator obtained by comparing a second-order Runge–Kutta error formula with a third-order one (hence the "23" in the function name). The method (see `rk3bs.m`) has three stages:

$$
\begin{aligned}
s_1 &= f(t_n,\, u_n)\\
s_2 &= f\big(t_n + \tfrac{h}{2},\, u_n + \tfrac{h}{2}s_1\big)\\
s_3 &= f\big(t_n + \tfrac{3h}{4},\, u_n + \tfrac{3h}{4}s_2\big)\\
u_{n+1} &= u_n + \frac{h}{9}\big(2s_1 + 3s_2 + 4s_3\big)
\end{aligned}
$$

Then, the approximate slope at the right endpoint is computed: $s_4 = f(t_n + h,\, u_{n+1})$, and the error estimator is given by

$$e_{n+1} = \frac{h}{72}\big(-5s_1 + 6s_2 + 8s_3 - 9s_4\big)$$

If e_{n+1} is within the specified tolerance, then the step is considered successful, the value of u_{n+1} is accepted, and Matlab proceeds to the next step; if e_{n+1} is not within the tolerance, then h is decreased and the step is repeated.

To get a rough idea of how adaptive time-stepping algorithms are coded, look at `ode23smp.m`, which is a simplified version of Matlab's built-in ODE-solving function `ode23`. As with all adaptive codes, instead of a step size h it takes a tolerance as input. Thus, `ode23smp` is called as follows:

```
[t,u]=ode23smp(f,tspan,u0,tol);
```

In fact, `ode23smp` takes a *pair* of tolerances as input: the *absolute* tolerance and the *relative* tolerance. The default absolute tolerance is 10^{-6}, and the default relative tolerance is 10^{-3}. To enter tolerances other than these, define `tol=odeset('RelTol',...,'AbsTol',...)` before calling `ode23smp`.

Matlab's ODE-solving functions

Matlab has a collection of built-in functions for approximating the solutions to ODEs; all are adaptive algorithms. The most frequently used ones are the following:

`ode23` explicit one-step three-stage Runge–Kutta method due to Bogacki and Shampine (error estimate based on comparing 2nd-order and 3rd-order Runge–Kutta formulas)

`ode45` explicit one-step four-stage Runge–Kutta method (error estimate based on comparing 4th-order and 5th-order Runge–Kutta formulas); the "workhorse" ODE solver

`ode113` explicit multistep predictor–corrector method that combines Adam–Bashforth and Adams–Moulton algorithms

`ode15s` implicit multistep method for solving "stiff" problems

Each ODE-solving function takes *input* of the form `(f,tspan,u0,options,...)` where:

`f` is a function with at least two input arguments `t,u` which outputs a column vector of the same length as `u` (`f` may also have additional input arguments that are parameters)

`tspan` is a row vector `[t0 T]` of initial and final times, or a row vector `[t0:increment:T]` of times at which to record solution values (NOTE: these are *not* the actual t_n's that Matlab uses for time-stepping, which are determined adaptively based on the error estimate at each step)

`u0` is a vector of initial values of `u`

`options` consists of optional arguments—such as absolute tolerance, relative tolerance, events—that should be specified in a statement of the form `options=odeset('RelTol',1e-4,'AbsTol',1e-7,...)` (the default relative tolerance is 10^{-3}, and the default absolute tolerance is 10^{-6})

There may be further input arguments after `options`, which are parameters of the ODE system.

The *output* of an ODE-solving function is of the form `[tout,uout,...]` where:

`tout` is a column vector of t-values at which the approximated solution value is recorded

`uout` is a matrix whose nth row contains the components of the computed solution corresponding to the nth row of `tout`

There may be further output arguments if you want to locate special "events".

If no output arguments are given, Matlab will plot the different solution components as functions of time while the solution is being computed.