

Quick Guide to Numerically Solving Systems of ODEs

In modelling and other endeavors it is common to express some relationship using a system of ordinary differential equations (ODEs). It is also common, that in trying to make a realistic relationship one produces a system which is unsolvable by ordinary means. If possible we prefer an analytic (closed form) solution, but often that is impossible. One way to ‘solve’ these unsolvable ODEs is to approximate the solution numerically.

1. Systems of ODEs: Before we get into the details of the numerics, let’s start with some information about systems. A single first order ODE can be written as

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0,$$

where f is a given function and t_0 and y_0 are given initial values. If we have a system of n ODEs, they can be written as

$$\frac{dy^i}{dt} = f_i(t, y^1, y^2, \dots, y^n), \quad y^i(t_0) = y_0^i, \quad i = 1, \dots, n.$$

If we write y for the vector (y^1, y^2, \dots, y^n) , $f(t, y)$ for the vector value $(f_1(t, y), f_2(t, y), \dots, f_n(t, y))$, and y_0 for the vector (y_0^1, \dots, y_0^n) , we can rewrite the system in the compact form

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0,$$

which is exactly the same as for a single ODE!

2. Higher Order ODEs: If we have a single, m th order ODE, we can write it as

$$\frac{d^m y}{dt^m} = f(t, y, y', y'', \dots, y^{(m-1)}), \quad y(t_0) = y_0^1, \dots, y^{(m-1)}(t_0) = y_0^m.$$

Note: only initial value problems can be written this way, if you have a boundary value problem, then other methods need to be used. This ODE can be converted into a system of m first order ODEs, by introducing the new variables $y^1 = y$, $y^2 = y'$, \dots , $y^m = y^{(m-1)}$. We then have

$$\frac{dy^k}{dt} = y^{k+1}, \quad k = 1, \dots, m-1$$

and

$$\frac{dy^m}{dt} = f(t, y^1, y^2, \dots, y^m),$$

with the obvious initial conditions.

3. Numerical Notation: The point of these last two sections is that when considering initial value problems involving ODEs, we can, without loss of generality, assume we are considering a system of the form:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

And thus we will do so.

Assume we want to know the solution $y(t)$ on an interval $[t_0, t_0 + T]$. Instead of finding the function, the numerical approach will estimate values of the function at particular points. Let $h = T/N$ for some large integer N . h is called the time step and is sometimes written as Δt . Let $t_i = t_0 + ih$ and then our goal is to determine values $\{u_i\}$ which approximate the true values $\{y_i = y(t_i)\}$. Note if we are solving a system then y_i and u_i are vectors. To summarize

$$h = T/N, \quad t_i = t_0 + ih, \quad u_i \approx y_i = y(t_i), \quad i = 0, 1, \dots, N.$$

4. Numerical Methods: Almost all numerical methods work the same way: given some information about y up to time t_i , predict what goes on in the interval $[t_i, t_{i+1})$ to determine u_{i+1} . A simple example of this is *Euler's Method*:

$$u_0 = y_0, \quad u_{i+1} = u_i + hf(t_i, u_i), \quad i = 1, \dots, N - 1.$$

In this case, we use $f(t_i, u_i)$ as a prediction of the behavior of y in the interval $[t_i, t_{i+1})$. As simple as this method is, it is also not very accurate. One can show that

$$|u_N - y_N| = \mathcal{O}(h),$$

using the ‘big Oh’ notation. A more popular (and better) method is the 4th-Order Classic Runge-Kutta Method (usually just called the Runge-Kutta method, although R-K is just a type of method). It is more complicated looking, but is fairly easy to implement. It starts with $u_0 = y_0$ and then to go from u_i to u_{i+1} you perform the following calculations:

$$\begin{aligned} K_1 &= f(t_i, u_i) \\ K_2 &= f\left(t_i + \frac{h}{2}, u_i + \frac{h}{2}K_1\right) \\ K_3 &= f\left(t_i + \frac{h}{2}, u_i + \frac{h}{2}K_2\right) \\ K_4 &= f(t_i + h, u_i + hK_3) \\ u_{i+1} &= u_i + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \end{aligned}$$

Note that if f is vector-valued then so is K_i , thus some care must be taken during the computer implementation of this method. For this method, we have

$$|u_N - y_N| = \mathcal{O}(h^4).$$

5. Adaptive Methods: There are many other methods of the form listed above, there are also methods called Adams or Multistep which are of a different form, but work on basically the same principle. A more practical method is an *adaptive* method, where a basic method is modified so that the step size h is modified from step to step so as to keep the total error within some user given tolerance. The idea behind an adaptive method is that given u_i , compute two approximations u_{i+1} and v_{i+1} by different methods with v being more accurate than u . Then the difference between u_{i+1} and v_{i+1} can be used to estimate the error in u_{i+1} . Since the local error is of the form Ch^p for some C and p , we can then figure out what size of h to use to make $Ch^p < \delta$ where δ is a given tolerance. The actual implementation of an adaptive scheme is challenging as one has to balance the needs for accuracy and for speed. Many routines are based on pairs of Runge-Kutta type methods, call Runge-Kutta-Fehlberg. The MATLAB routines `ode45` and `ode23` are such routines. You can also find such routines at `netlib` (www.netlib.org).