Introduction to Scientific Computing
Lecture 8

Professor Hanno Rein

Last updated: November 5, 2018

Numerical Solutions to Ordinary Differential Equations

Standard Form

Most ODE solvers that we discuss expect the ODE in a standard form, where the left hand side is a derivate of a vector y and the right and side is a function of the same vector y. In the simplest case the vector has just one dimension and we end up with a one dimensional ODE.

As an example, we will now convert an ODE with a second order derivative of y into a system of first order derivatives in the standard vector notation. We begin with the differential equation which is *not* in standard form:

$$y'' = F(y).$$

We can rewrite this as a vector equation and first derivatives as

$$\begin{pmatrix} y' \\ y'' \end{pmatrix} = \begin{pmatrix} y' \\ F(y) \end{pmatrix}$$

and thus

$$\begin{pmatrix} y \\ y' \end{pmatrix}' = \begin{pmatrix} y' \\ F(y) \end{pmatrix}$$

or equally

$$Y' = G(Y)$$

where we introduced new variables Y and the function G. This new equation is now in standard form.

Euler Method

The simplest method to solve a differential equation of the form

$$y'(t) = F(y, t)$$

is the Euler method. We start from an initial condition consisting of two values, t_0 and $y_0 = y(t_0)$. We then integrate the equation forward in time, one timestep dt at a time. Often the timestep is also denoted by the letter h. The size of the timestep determines the precision of the scheme. The smaller the timestep the higher the accuracy but the more calculations we have to do. This statement is true not only for the Euler method but for all ODE (ordinary differential equation) solvers.

To perform an Euler step, we evaluate the derivate of the function $y(t)$ at the beginning of the timestep. For the first timestep, t is t_0, for the second timestep $t_0 + dt$, and so on. We then multiply the derivative with the timestep dt and add it to the initial value y_0. If we label subsequent steps with the index n, then the Euler method can be written as

$$y_{n+1} = y_n + dt \cdot F(y_n, t_n).$$

Note that you can think of the the function y as either a scalar function or a vector function. The Euler method (and any other ODE solver) works exactly the same in both cases. In other words, it works in multiple dimensions as well as in just one.

We now perform an error analysis on the Euler method. We do this to find out how well the Euler method does, how large the error is and how it depends on the timestep dt. So, let's go back to our generic example

$$y'(t) = F(y, t)$$

and expand the function y as a Taylor series around the $t = t_0$

$$\begin{aligned} y(t) &= y_0 + (t - t_0) \cdot \frac{\partial y}{\partial t} + \frac{1}{2}(t - t_0)^2 \cdot \frac{\partial^2 y}{\partial t^2} + \cdots \\ &= y_0 + dt \cdot F(y, t) + \frac{1}{2} dt^2 \cdot \frac{\partial^2 y}{\partial t^2} + \cdots \end{aligned}$$

The Euler method gets the first two terms right. The error after one timestep is therefor of the order

$$E \sim \frac{1}{2} dt^2 \cdot \frac{\partial^2 y}{\partial t^2}$$

plus higher order terms. The higher order terms will be small if dt is small and the dominant error term will be of order dt^2.

Suppose we want to integrate a system forward in time for some finite time T. Then the number of timesteps N depends on the size of the timestep dt:

$$N = \frac{T}{dt}$$

The smaller the timestep, the more timesteps we need to take. Thus, the error of the Euler method after a finite time T is now not of order dt^2 anymore, but of order dt. This is an important result. The Euler method is a first order method. If we reduce the timestep by a factor of two, then the error will be smaller by a factor of two. This is good because we eventually converge to the correct solution if we only make the timestep small enough. However, we might hope to find a better method that is more accurate than the Euler method and converges faster, i.e. quadratically. This is important because in most situations the evaluation of the function F is very computationally expensive and we want to minimize the number of evaluations. The Euler method has one function evaluation per timestep.

Runge-Kutta Methods

As we indicated before, we might be able to cancel out higher order terms in the expansion of y. This would lead us to a high order scheme which is better and faster than the simple Euler Methods. A class of higher order methods that is widely used is the class of Runge-Kutta Methods. These methods use the Euler method to make predictions of y during the timestep and then use this information to improve the result at the end.

The mid-point method, or second-order Runge-Kutta method, is one such method. The idea is simple. Let's first take half an Euler step, to get an estimate of the function F in the middle of the timestep. Then use this estimate to advance y from the beginning of the timestep to the end. In terms of equations, it can be written as follows (using the same notation as above):

$$\begin{aligned}
k_1 &= dt \cdot F(y_n, t_n) \\
k_2 &= dt \cdot F\left(y_n + \frac{1}{2} \cdot k_1, t_n + \frac{1}{2}dt\right) \\
y_{n+1} &= y_n + k_2
\end{aligned}$$

Let's do an error analysis for this method. The Taylor expansion of the function y to third order is

$$y(t) = y_0 + (t - t_0) \cdot \frac{\partial y}{\partial t} + \frac{1}{2}(t - t_0)^2 \cdot \frac{\partial^2 y}{\partial t^2} + \frac{1}{3}(t - t_0)^3 \cdot \frac{\partial^3 y}{\partial t^3} + \cdots .$$

An integration step of the mid-point method gives

$$\begin{aligned}
y_{n+1} &= y_n + k_2 \\
&= y_n + dt \cdot F\left(y_n + \frac{1}{2} \cdot k_1, t_n + \frac{1}{2}dt\right) \\
&= y_n + dt \cdot F\left(\underbrace{y_n + \frac{1}{2} \cdot dt \cdot F(y_n, t_n)}_{=y_{n+\frac{1}{2}}+O(dt^2)}, t_n + \frac{1}{2}dt\right) \\
&= y_n + dt \cdot \underbrace{F\left(y_n + \frac{1}{2} \cdot dt \cdot F(y_n, t_n), t_n + \frac{1}{2}dt\right)}_{=F_{n+\frac{1}{2}}+O(dt^2)}
\end{aligned}$$

We've seen this before. This is the central difference scheme from last lecture (ignoring the higher order error term). The central difference scheme was used to estimate the derivate of a function. It is symmetric with respect to a reflection around the mid-point. This makes it higher order (2nd). Here, this is what makes the mid-point method second order accurate. We now have an error of order $O(dt^3)$ after one timestep. If the error of a scheme is $O(dt^{n+1})$ after one timestep, then we call the scheme $n-$th order.

The most commonly used Runge-Kutta method is the fourth order Runge-Kutta Method, or RK4. It uses four function evaluations and can be written in a similar way as the mid-point method.

$$\begin{aligned}
k_1 &= dt \cdot F(y_n, t_n) \\
k_2 &= dt \cdot F\left(y_n + \frac{1}{2} \cdot k_1, t_n + \frac{1}{2}dt\right) \\
k_3 &= dt \cdot F\left(y_n + \frac{1}{2} \cdot k_2, t_n + \frac{1}{2}dt\right) \\
k_4 &= dt \cdot F\left(y_n + \cdot k_3, t_n + dt\right) \\
y_{n+1} &= y_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4
\end{aligned}$$

As one can show (we won't) the error after one timestep is $O(h^5)$, thus it is indeed a fourth order method.

There are many different Runge-Kutta scheme. They are usually written in block form. For example, the coefficients for RK4 can be summarized as

$$\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & & \frac{1}{2} & & \\
1 & & & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}$$

Note that the 0's are not printed in the block form. Similarly, the Euler method can be summarized as

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$$

And the midpoint method can be summarized as

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array}$$