# Numerical solutions

Here I want to briefly discuss numerical solutions to an ordinary differential equation. This is a big subject and, in a way, I don't want this section to suggest that you implement your own numerical solvers. A lot of folk have put a lot of work into numerical solvers, `numpy` in `Python` and `DifferentialEquations.jl` in `Julia` contain incredibly well writen and speedy numerical solvers; I am sure `Matlab` can also do this well though since this is 2023 and not 2003 I don't actually know what `Matlab` is doing. Anyway, the point is you should always use numerical libraries when possible, they are robust and highly optimized, but it is useful to have some idea what is happening.

The main thing that is happening is the Taylor expansion. We have seen before that

$$y(x+h) = y(x) + \dot{y}(x)h + \text{stuff that has } h^2 \text{ or higher} \tag{1}$$

So say you have a first order differential equation

$$\dot{y} = F(y,t) \tag{2}$$

with an intial condition $y(0) = y_0$ and you want to solve it but can't do that analytically, maybe because the equation is complicated and non-linear or maybe because it gets input you can't anticipate in advance or whatever. On a computer you can divide time up into little $\delta$ sized pieces

$$t = n\delta \tag{3}$$

You then define

$$y_n = y(t_n) \tag{4}$$

and using the Taylor expansion you try to write $y_n$ in terms of $y_{n-1}$, then you can start at $y_0$, the initial condition, and work out all the $y_n$. The simplest approximation is called the Euler approximation, where you stop the Taylor expansion at $\delta$:

$$y_n = y(n\delta) = y[(n-1)\delta+\delta] = y_{n-1}+\delta\dot{y}[(n-1)\delta] = y_{n-1}+F(y_{n-1},t_{n-1})\delta \tag{5}$$

From the Taylor expansion we know this approximation has errors of size $\delta^2$, but it does provide a simple way to approximate the solution to the

differential equation. The smaller $\delta$ the more accurate the approximation is, but the slower the calculation.

In fact, the Euler approximation is rarely used, a common approach is the fourth order Runge-Kutta integrator. It isn't obvious from the formula how this works, in fact, what Runge-Kutta did was they found from the Taylor expansion a formula where the coefficients of the higher order terms cancel up to $\delta^5$; in short I amn't showing the workings which are straight forward but very long, but the Runge-Kutta method works a bit like the Euler method except four different approximations like the Euler method are used and arranged so that the errors we know from the Taylor expansion, startin with the $\ddot{y}\delta^2/2$ all cancel up to $\delta^5$. The approximation is

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\delta \tag{6}$$

where

$$\begin{aligned} k_1 &= F(y_n, t_n) \\ k_2 &= F(y_n + k_1\delta/2, t_n + \delta/2) \\ k_3 &= F(y_n + k_2\delta/2, t_n + \delta/2) \\ k_3 &= F(y_n + k_3\delta, t_n + \delta) \end{aligned} \tag{7}$$

However, as I said, if possible, don't ever implement this yourself, use a library.

The fourth order Runge-Kutta is by no means the only integrator, it is something of a stardard approach, but when it comes to complicated differential equation, different equations have different challenges and so different methods are appropriate.

# Summary

To solve a first order differential equation numerically time is divided up into tiny steps and the Taylor expansion is derivate a formula to approximate the value at one step from the value at previous steps. The most straight-forward formula like this is the Euler approximation:

$$y_{n+1} = y_n + \delta F(y_n, t_n) \tag{8}$$

but the fourth-order Runge-Kutta method is the standard. There are other methods appropriate for equations with different properties, but, in practice,

while it is useful to have an understanding of how a method you use works, it is best to take the implementation from a library.