Ordinary differential equations

$$\frac{dy}{dt} = f(y,t) \quad , \quad \begin{array}{l} f: \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}^N \\ y: \mathbb{R} \to \mathbb{R}^N \end{array}$$

$$\left.\begin{array}{lll} f = f(y,t) & \to & \text{non-autonomous} \\ f = f(y) & \to & \text{autonomous} \end{array}\right\} \text{system}$$

Note: It is always possible to rewrite an non-autonomous system of dimension N as autonomous by adding an extra variable.

$$f(y,t) \to \tilde{f}(\tilde{y}), \text{where} \quad \begin{array}{l} \dot{\tilde{y}}_1 = \tilde{f}(y_1, \dots, y_N, y_{N+1}) \\ \vdots \\ \dot{\tilde{y}}_N = \tilde{f}(y_1, \dots, y_N, y_{N+1}) \\ \dot{\tilde{y}}_{N+1} = 1 \end{array}$$

Next to the ODE, we require inital conditions $y_0 = (\eta_1 \;\; \eta_2 \; \dots \; \eta_N)^T$. (If we converted a non-autonomous eq. to the autonomous form η_{N+1} has to be t_0).

In the general case, the ODE will be of high order

$$y^{(n)} = \phi(y, \dot{y}, \ddot{y}, \dots, y^{(n-1)}).$$

Here y may be scalar, but a vector as well, e.g. the motion of a point-like mass:

$$\ddot{x} = \frac{F(x)}{m}, \qquad \left[x^{(2)} = \phi(x, \cancel{\dot{x}}, \ddot{x}, \cancel{t}) \right]$$

where x may be a scalar or a vector.

Every high order equation can be formulated as a system of first order ODEs, hence most numerical methods focus on the solution of such systems.

Reformulation is done in the following way:

1) $x^{(n)} = \phi(x, \dot{x}, \ldots, x^{(n-1)}) \rightarrow$ ODE of order n

2) create vector y of dimension n and formally identify:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x \\ \dot{x} \\ \ddot{x} \\ \vdots \\ x^{(n-1)} \end{pmatrix}$$

3) The derivative $\frac{dy}{dt}$ is $\frac{dy}{dt} = \begin{pmatrix} \dot{x} \\ \ddot{x} \\ x^{(3)} \\ \vdots \\ x^{(n)} \end{pmatrix}$, where we now make use of the original equation

$x^{(n)} = \phi(x, \ldots, x^{(n-1)}) = \phi(y_1, \ldots, y_n)$ to obtain the first order system

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \vdots \\ \dot{y}_{n-1} \\ \dot{y}_n \end{bmatrix} = \begin{bmatrix} y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_n \\ \phi(y_1 \ldots y_n) \end{bmatrix}.$$

<u>Mini-examples:</u> a) $\ddot{x} + x = 0 \Rightarrow y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} x \\ \dot{x} \end{pmatrix}, \frac{dy}{dt} = \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \ddot{x} \end{pmatrix} = \begin{pmatrix} y_2 \\ -y_1 \end{pmatrix},$

$$\frac{dx}{dt} = v = \frac{p}{m\gamma}$$

$$\frac{dp}{dt} = q(E + v \times B)$$

$$\Rightarrow y = \begin{pmatrix} x \\ p \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \quad \frac{dy}{dt} = \begin{pmatrix} p/m\gamma \\ q(E + \frac{p}{m\gamma} \times B) \end{pmatrix} = \begin{bmatrix} y_2/m\gamma \\ q(E + \frac{y_1}{m\gamma} \times B) \end{bmatrix}$$

$$\gamma = \sqrt{1 + \frac{p^2}{m^2c^2}} = \sqrt{1 + \frac{y_2^2}{m^2c^2}}$$

Euler methods

Assume scalar ODE $\frac{dy}{dt} = f(y,t)$. Integration gives

$$\int_{t_0}^{t_1} \frac{dy}{dt} = \int_{t_0}^{t_1} f(y(t),t)\,dt \iff y(t_1) - y(t_0) = \int_{t_0}^{t_1} f(y(t),t)\,dt$$

The integral may be approximated as:

$$\int_{t_0}^{t_1} f(y,t)\,dt \simeq \begin{cases} f(y(t_0),t_0)\,\Delta t \\ f(y(t_1),t_1)\,\Delta t \end{cases}, \text{ where } \Delta t = t_1 - t_0$$

This yields the two Euler methods:

Forward Euler: $\quad y(t_1) = f(y(t_0),t_0)\Delta t + y_0$

Backward Euler: $\quad y(t_1) = f(y(t_1),t_1)\Delta t + y_1$

Forward Euler is an explicit method, $y(t_1)$ may be directly calculated upon knowledge of $y(t_0)$. Backward Euler is an implizit method, since it only gives an implicit equation for $y(t_1)$.

The order of the error we make in every step can be inferred from another way of looking at the problem.

In the case of Euler forward:

$$y(t_0+\Delta t) = y(t_0) + \Delta t\, y'(t_0) + \frac{1}{2}\Delta t^2 y''(t_0) + O(\Delta t^3)$$

$$\iff \frac{y(t_0+\Delta t) - y(t_0)}{\Delta t} = y'(t_0) + O(\Delta t)$$

$$\iff \frac{y(t_1) - y(t_0)}{\Delta t} = f(y(t_0),t_0) + O(\Delta t)$$

$$\iff y(t_1) = \Delta t\, f(y(t_0),t_0) + y(t_0) + O(\Delta t^2)$$

From here we draw the conclusions:

a) $$\frac{dy}{dt} = \frac{y(t+\Delta t) - y(t)}{\Delta t} + \mathcal{O}(\Delta t)$$

b) In each step Δt we make a local truncation error

$$\tfrac{1}{2}\Delta t^2 y''(t_0) + \mathcal{O}(\Delta t^3) = \mathcal{O}(\Delta t^2)$$

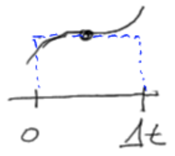
For Euler backward we find $\frac{dy}{dt} = \frac{y(t)-y(t-\Delta t)}{\Delta t} + \mathcal{O}(\Delta t)$ and the same order of error for each step.

$$\left[\begin{aligned} &y(t-\Delta t) = y(t) - \Delta t f'(t) + \tfrac{1}{2}\Delta t^2 f''(t) + \mathcal{O}(\Delta t^3) \\ \Leftrightarrow\ &\frac{y(t)-y(t-\Delta t)}{\Delta t} = f'(t) - \tfrac{1}{2}\Delta t^2 f''(t) + \mathcal{O}(\Delta t^3)\end{aligned}\right]$$

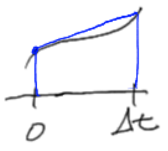
Integration from $t_0 = 0$ to $t_{end} = T$ will take a number of small steps. Assuming a constant step-size Δt, we have to take $N = \frac{L}{\Delta t}$ steps. In each step we have a local error $\mathcal{O}(\Delta t^2)$, but after N steps we may have accumulated $N\mathcal{O}(\Delta t^2) = \frac{L}{\Delta t}\mathcal{O}(\Delta t^2) = \mathcal{O}(\Delta t)$. Thus, both Euler methods are said to be of order Δt.

To increase accuracy, obviously the local error has to be reduced. Two ways to do this are the mid-point and the trapezoiadal rule, respectively.

Mid-point: $$\int_0^{\Delta t} f(y,t)\,dt \approx f\left(y(\Delta t/2), \frac{\Delta t}{2}\right)\Delta t$$



Trapezoidal: $$\int_0^{\Delta t} f(y,t)\,dt \approx \tfrac{1}{2}\Big[f(y(0),0) + f(y(\Delta t),\Delta t)\Big]\Delta t$$



4
23.11.15

This results in:

$$y(t+\Delta t) = y(t) + f\left(y\left(t+\tfrac{\Delta t}{2}\right), \tfrac{\Delta t}{2}\right)\Delta t \quad \text{(Midpoint)}$$

$$y(t+\Delta t) = y(t) + \tfrac{1}{2}\left[f(y(t+\Delta t), t+\Delta t) + f(y(t), t)\right]\Delta t \quad \text{(Trapezoidal)}$$

While for the trapezoidal rule it is clear that it is an implicit method, it the midpoint rule has two variations. How to evaluate $f\left(y\left(t+\frac{\Delta t}{2}\right), t+\frac{\Delta t}{2}\right)$?

$$\text{Explicit:} \quad y\left(t+\tfrac{\Delta t}{2}\right) \approx y(t) + \tfrac{\Delta t}{2} f(y(t), t)$$

$$\text{Implicit:} \quad y\left(t+\tfrac{\Delta t}{2}\right) \approx \tfrac{1}{2}\left[y(t) + y(t+\Delta t)\right]$$

Introducing the notation $t_n = n\Delta t$, $y(t_n) = y_n$, we have

$$y_{n+1} = y_n + \tfrac{1}{2}\Delta t\left[f(y_n, t_n) + f(y_{n+1}, t_{n+1})\right] \quad \text{(Trapezoidal)}$$

$$y_{n+1} = y_n + \Delta t \; f\left(y_n + \tfrac{\Delta t}{2} f(y_n, t_n), t_{n+1/2}\right) \quad \text{(explicit midpoint)}$$

$$y_{n+1} = y_n + \Delta t \; f\left(\frac{y_n + y_{n+1}}{2}, t_{n+1/2}\right) \quad \text{(implicit midpoint)}$$

These three methods have a local error of $\mathcal{O}(\Delta t^3)$ and thus a global error $\mathcal{O}(\Delta t^2)$.

All of the above methods are single-step methods, since starting from the knowledge of y_n and the ODE itself, we can compute y_{n+1}. For multi-step methods values y_{n-1}, $y_{n-2}, \dots$ have to be known in order to determine y_{n+1}.

Runge-Kutta methods

RK methods are the most popular single-step methods.

Idea: Evaluate $f(y,t)$ not only at left or right boundary of interval $[t, t+\Delta t]$, but use also intermediate evaluations.

A m-stage RK method has the form

$$\vec{y}_{n+1} = \vec{y}_n + \Delta t \sum_{i=1}^{m} b_i \vec{k}_i ,$$

$$\vec{k}_i = f\left(\vec{y}_n + \Delta t \sum_{j=1}^{m} a_{ij} \vec{k}_j , t_n + c_i \Delta t \right) , \quad i = 1, \dots, m$$

The parameters b_i, c_i, a_{ij} are determined in such a way, that for given m we obtain the largest order p.
To find the coefficients, the numerical solution $\vec{y}_{n+1}$ and the analytical solution $\vec{y}_{ana}(t+\Delta t)$ are Taylor expanded.
Comparison of the coefficients leads to nonlinear equations for the parameters. Usually, the solution to the equations is not unique. Each solution is a different RK method.

Setting $c_1 = 0$, $a_{ij} = 0$ for $j \geq i$ results in an explicit method.
For the calculation of $\vec{k}_i$ only previous values $\vec{k}_1, \vec{k}_2, \dots, \vec{k}_{i-1}$ are used.

Example: $p = m = 1$, $b_1 = 1$, $b_i = 0$ $(i > 1)$, $c_i = 0$, $a_{ij} = 0$

$$\Rightarrow \quad \vec{y}_{n+1} = \vec{y}_n + \Delta t \, \vec{k}_1 , \quad \vec{k}_1 = f(\vec{y}_n, t_n)$$

This is the Euler forward method.

The popular fourth-order RK method (RK4) is

$$\vec{k}_1 = f(\vec{y}_n, t_n)$$
$$\vec{k}_2 = f(\vec{y}_n + \frac{\Delta t}{2}\vec{k}_1, \; t_n + \frac{\Delta t}{2})$$
$$\vec{k}_3 = f(\vec{y}_n + \frac{\Delta t}{2}\vec{k}_2, \; t_n + \frac{\Delta t}{2})$$
$$\vec{k}_4 = f(\vec{y}_n + \Delta t \vec{k}_3, \; t_n + \Delta t)$$

$$\vec{y}_{n+1} = \vec{y}_n + \frac{\Delta t}{6}(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4) + \mathcal{O}(\Delta t^5)$$

In general the coefficients are most easily summarized in a Butcher tableau. For an explicit m-stage method it has the form

$$\begin{array}{c|ccccc} c_1 & & & & & \\ c_2 & a_{21} & & & & \\ \vdots & & & & & \\ c_m & a_{m1} & a_{m2} & & a_{m,m-1} & \\ \hline & b_1 & b_2 & \dots & b_{m-1} & b_m \end{array} \qquad \begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 2/6 & 2/6 & 1/6 \end{array}$$

Order p can not be obtained with less than p stages. RK4 is the highest order for which $m=p$. Methods with higher order have $m>p$.

Other RK methods in Butcher notation:

$$\begin{array}{c|c} 1 & 0 \\ \hline & 1 \end{array}$$ Euler forward, $\mathcal{O}(\Delta t)$

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$ Euler backward, $\mathcal{O}(\Delta t)$

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1/2 & 1/2 \end{array}$$ Heun's method, $\mathcal{O}(\Delta t^2)$

$$\begin{array}{c|cc} 0 & & \\ 1/2 & 1/2 & \\ \hline & 0 & 1 \end{array}$$ explicit mid-point $\mathcal{O}(\Delta t^2)$

$$\begin{array}{c|cc} 0 & & \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array}$$ implicit mid-point $\mathcal{O}(\Delta t^2)$

Dense output

With the RK methods above we obtain solutions y_n, $n=1,\dots,N$, but we might be interested in the solution at time t^*. t^* might even be not known in advance, it may be implicitly dependent on the computed solution via a function $g(y(t^*), t^*) = 0$ (for example t^* is defined as the point in time for which one component of $\vec{y}$ is zero and we need to know the other components).

In this case the solution at a point in the interval $[t_n, t_{n+1}]$ can be approximated by an interpolating function.

In general $\quad \vec{y}_{n+\theta} = \vec{y}_n + \Delta t \sum_{i=1}^{m^*} b_i(\theta)\,\vec{k}_i \quad , 0 \le \theta \le 1, \quad m^* \ge m.$

In the case of RK4 an interpolation of order 3 can be obtained by

$$b_1(\theta) = \theta - \frac{3\theta^2}{2} + \frac{2\theta^3}{3}, \quad b_2(\theta) = b_3(\theta) = \theta^2 - \frac{2\theta^3}{3}, \quad b_4(\theta) = -\frac{\theta^2}{2} + \frac{2\theta^3}{3}.$$

Here we have $m^* = m$, which results in a lower order of interpolation.

Richardson extrapolation

We may increase the order of the global error by one if we compute each time-step twice using the same integration scheme. If the order of the method is p we find:

$$1 \text{ step of size } \Delta t: \; \vec{y}(x_n) = \vec{y}_n + C\Delta t^{p+1} + \mathcal{O}(\Delta t^{p+2})$$

$$2 \text{ steps of size } \frac{\Delta t}{2}: \; \vec{y}(x_n) = \tilde{\vec{y}}_n + 2C\left(\frac{\Delta t}{2}\right)^{p+1} + \mathcal{O}(\Delta t^{p+2})$$

(Here $\vec{y}(x_n)$ is the true solution and we suppose small enough Δt for C to be the same in both cases).

Now we multiply eq. $\textcircled{2}$ by 2^p and subtract $\textcircled{1}$

$$\textcircled{2} \quad 2^p \, \vec{y}_n(x_n) = 2^p \, \tilde{\tilde{\vec{y}}}_n + c\Delta t^{p+1} + \mathcal{O}(\Delta t^{p+2})$$

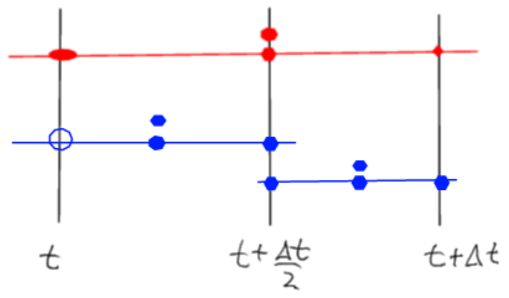
$$\textcircled{2} - \textcircled{1} \quad (2^p - 1)\vec{y}_n(x_n) = 2^p \, \tilde{\tilde{\vec{y}}}_n - \vec{y}_n + \mathcal{O}(\Delta t^{p+2})$$

This results in

$$\vec{y}_n(x_n) = \frac{2^p \, \tilde{\tilde{\vec{y}}}_n - \vec{y}_n}{2^p - 1} + \mathcal{O}(\Delta t^{p+2}),$$

which is one order better than the original expressions.

How many evaluations of $f(\vec{y}, t)$ are needed in case of RK4 using Richardson extrapolation?



4 evaluations @ Δt step-size

7 evaluations @ 2 steps à $\frac{\Delta t}{2}$

(○ is the same as ●)

In total 11 evaluations are required to achieve $\mathcal{O}(\Delta t^5)$ using RK4.

Embedded RK methods

m-stage RK method of order p:

$$\vec{y}_{n+1} = \vec{y}_n + \Delta t \sum_{i=1}^{m} b_i \vec{k}_i + \mathcal{O}(\Delta t^{p+1})$$

A second method of different order $\hat{p}$ may share the $\vec{k}_i$:

$$\hat{\vec{y}}_{n+1} = \vec{y}_n + \Delta t \sum_{i=0}^{m} \hat{b}_i \vec{k}_i + \mathcal{O}(\Delta t^{\hat{p}+1})$$

If $\hat{\vec{y}}_{n+1}$ and $\vec{y}_{n+1}$ share the majority of the $\vec{k}_i$, then we obtain two results of different orders, where the more precise result demands little extra effort to compute.

We may use $\|\Delta\| = \|\vec{y}_{n+1} - \hat{\vec{y}}_{n+1}\|$ as a local error estimate.

A well known embedded method is the Dormand-Prince method, which combines a 4th and a 5th order RK.

Butcher Tableau for Dormand-Prince 4(5) method.

	0							
	1/5	1/5						
	3/10	3/40	9/40					
	4/5	44/45	−56/15	32/9				
	8/9	19372/6561	−25360/2187	64448/6561	−212/729			
	1	9017/3168	−355/33	46732/5247	49/176	−5103/18656		
	1	35/384	0	500/1113	125/192	−2187/6784	11/84	
$b_i \longrightarrow$		35/384	0	500/1113	125/192	−2187/6784	11/84	0
$\hat{b}_i \longrightarrow$		5179/57600	0	7571/16695	393/640	−92097/339200	187/2100	1/40

Once we have results of different accuracy (either obtained via extrapolation or from embedded methods), we have the possibility to control the step-size Δt in every step. In places where $f(\vec{y},t)$ is changing slowly we may take larger steps and in the case of rapid variations we reduce the step-size.

Assume two numerical solutions of order p and $p+1$:

$$\vec{y}_{n+1} = \vec{y}(t_{n+1}) + \vec{c}\,\Delta t^{p+1} + \mathcal{O}(\Delta t^{p+2})$$

$$\hat{\vec{y}}_{n+1} = \vec{y}(t_{n+1}) + \mathcal{O}(\Delta t^{p+2})$$

We may use $\mathcal{X}(\Delta t) = \|\vec{y}_{n+1} - \hat{\vec{y}}_{n+1}\|$ as an approximation to the error of the low order method.

We specify an error ε which we are willing to tolerate, what we need to find is a step-size Δt_{new} which we have to use to achieve $\mathcal{X}(t_{new}) < \varepsilon$.

$$\frac{\mathcal{X}(\Delta t)}{\underbrace{\mathcal{X}(\Delta t_{new})}_{=\varepsilon}} = \frac{\Delta t^5}{\Delta t_{new}^5} \Rightarrow \Delta t_{new} = \frac{\Delta t}{\mathcal{X}(\Delta t)^{1/5}}\,\varepsilon^{1/5}$$

Since Δt_{new} will be only used in the next step, a safety-factor $q \simeq 0.1 - 0.9$ is usually used, i.e.

$$\Delta t_{new} = q\,\Delta t\left(\frac{\varepsilon}{\mathcal{X}(\Delta t)}\right)^{1/5}.$$