Numerical solution of ordinary differential equations: High order Runge-Kutta methods

André Massing

Mar 23, 2021

The Python codes for this note are given in `ode.py`.

1 Runge-Kutta Methods

In the previous lectures we introduced *Euler's method* and *Heun's method* as particular instances of the *One Step Methods*, and we presented the general error theory for one step method.

In this Lecture, we introduce a large family of the one step methods which go under the name **Runge-Kutta methods (RKM)**. We will see that Euler's method and Heun's method are instance of RKMs.

1.1 Derivation of Runge-Kutta Methods

For a given time interval $I_i = [t_i, t_{i+1}]$ we want to compute y_{i+1} assuming that y_i is given. Starting from the exact expression

$$y(t_{i+1}) - y(t_i) = \int_{t_i}^{t_{i+1}} f(t, y(t))\,\mathrm{d}t,$$

the idea is now to approximate the integral by some quadrature rule $\mathrm{Q}[\cdot](\{\xi_j\}_{j=1}^s, \{b_j\}_{j=1}^s)$ defined on I_i. Then we get

$$y(t_{i+1}) - y(t_i) = \int_{t_i}^{t_{i+1}} f(t, y(t))\,\mathrm{d}t \tag{1}$$

$$\approx \tau \sum_{j=0}^{s} b_j f(\xi_j, y(\xi_j)) \tag{2}$$

Now we can define $\{c_j\}_{j=1}^s$ such that $\xi_j = t_i + c_j\tau$ for $j = 1, \ldots, s$

Exercise 1: A first condition on b_j

Question: What value do you expect for $\sum_{j=1}^s b_j$?

A. $\sum_{j=1}^s b_j = \tau$

B. $\sum_{j=1}^s b_j = 0$

C. $\sum_{j=1}^s b_j = 1$

Answer: C.

Solution: **A**: Wrong. **B**: Wrong. **C**: Right.

In contrast to pure numerical integration, we don't know the values of $y(\xi_j)$. Again, we could use the same idea to approximate

$$y(\xi_j) - y(t_i) = \int_{t_i}^{t_i + c_j\tau} f(t, y(t))\,\mathrm{d}t$$

but then again we get a closure problem if we choose new quadrature points. The idea is now to *not introduce even more new quadrature points* but to use same $y(\xi_j)$ to avoid the closure problem. Note that this leads to an approximation of the integrals $\int_{t_i}^{t_i+c_j\tau}$ with possible nodes *outside* of $[t_i, t_i + c_j\tau]$.

This leads us to

$$y(\xi_j) - y(t_i) = \int_{t_i}^{t_i + c_j\tau} f(t, y(t))\,\mathrm{d}t \tag{3}$$

$$\approx c_j\tau \sum_{l=1}^{s} \tilde{a}_{jl} f(\xi_l, y(\xi_l)) \tag{4}$$

$$= \tau \sum_{l=1}^{s} a_{jl} f(\xi_l, y(\xi_l)) \tag{5}$$

where we set $c_j\tilde{a}_{jl} = a_{jl}$.

Exercise 2: A first condition on a_{jl}

Question: What value do you expect for $\sum_{l=1}^{s} a_{jl}$?

A. $\sum_{l=1}^{s} a_{jl} = \frac{1}{c_j}$

B. $\sum_{l=1}^{s} a_{jl} = c_j$

C. $\sum_{l=1}^{s} a_{jl} = 1$

D. $\sum_{l=1}^{s} a_{jl} = \tau$

Answer: B.

Solution: **A**: Wrong. **B**: Right. **C**: Wrong. **D**: Wrong.

Definition 1.1. *Runge-Kutta methods.*

Given b_j, c_j, and a_{jl} for $j, l = 1, \dots s$, the Runge-Kutta method is defined by the recipe

$$Y_j = y_i + \tau \sum_{l=1}^{s} a_{jl} f(t_i + c_l\tau, Y_l) \quad \text{for } j = 1, \dots s, \tag{6}$$

$$y_{i+1} = y_i + \tau \sum_{j=1}^{s} b_j f(t_i + c_j\tau, Y_j) \tag{7}$$

Runge-Kutta schemes are often specified in the form of a **Butcher table**:

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array} \tag{8}$$

If $a_{ij} = 0$ for $j \geqslant i$ the Runge-Kutta method is called **explicit**. (Why?)

Note that in the final step, all the function evaluation we need to perform have already been performed when computing Y_j.

Therefore one often rewrite the scheme by introducing **stage derivatives**

$$k_l = f(t_i + c_l\tau, Y_l) \tag{9}$$

$$= f(t_i + c_l\tau, y_i + \tau \sum_{j=1}^{s} a_{lj}k_j) \quad j = 1, \dots s, \tag{10}$$

so the resulting scheme will be (swapping index l and j)

$$k_j = f(t_i + c_j\tau, y_i + \tau \sum_{l=1}^{s} a_{jl}k_l) \quad j = 1, \dots s, \tag{11}$$

$$y_{i+1} = y_i + \tau \sum_{j=1}^{s} b_j k_j \tag{12}$$

Exercise 3: Butcher table for the explicit Euler

Write down the Butcher table for the explicit Euler.

Solution. Define $k_1 = f(t_i, y_i) = f(t_i + 0 \cdot \tau, y_i + \tau \cdot 0 \cdot k_1)$. Then the explicit Euler step $y_{i+1} = y_i + \tau k_1 = y_i + \tau \cdot 1 \cdot k_1$, and thus the Butcher table is given by

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

Exercise 4: The improved explicit Euler method

We formally derive the **explicit midpoint rule** or **improved explicit Euler method**. Applying the midpoint rule to our integral representatio yields

$$y(t_{i+1}) - y(t_i) = \int_{t_i}^{t_{i+1}} f(t, y(t))\,\mathrm{d}t \tag{13}$$

$$\approx \tau f(t_i + \tfrac{1}{2}\tau, y(t_i + \tfrac{1}{2}\tau)) \tag{14}$$

Since we cannot determine the value $y(t_i + \frac{1}{2}\tau)$ from this system, we approximate it using a half Euler step

$$y(t_i + \tfrac{1}{2}\tau) \approx y_{t_i} + \tfrac{1}{2}\tau f(t_i, y(t_i))$$

leading to the scheme

$$y_{i+1/2} := y_i + \tfrac{1}{2}\tau f(t_i, y_i) \tag{15}$$

$$y_{i+1} := y_i + \tau f(t_i + \tfrac{1}{2}\tau, y_{i+1/2}) \tag{16}$$

a) Is this a one-step function? Can you define the increment function Φ?

Solution. Yes it is, and it's increment function is given by

$$\Phi(t_i, y_i, y_{i+1}, \tau) = f(t_i + \tfrac{1}{2}\tau, y_i + \tfrac{1}{2}\tau f(t_i, y_i))$$

b) Can you rewrite this as a Runge-Kutta method? If so, determine the Butcher table of it.

Solution. Define k_1 and k_2 as follows,

$$y_{i+1/2} := y_i + \tfrac{1}{2}\tau \underbrace{f(t_i, y_i)}_{=:k_1} \tag{17}$$

$$y_{i+1} := y_i + \tau f(t_i + \tfrac{1}{2}\tau, y_{i+1/2}) = y_i + \tau \underbrace{f(t_i + \tfrac{1}{2}\tau, y_i + \tau\tfrac{1}{2}k_1)}_{:=k_2}. \tag{18}$$

Then

$$y_{i+1} = y_i + \tau k_2 \tag{19}$$

and thus the Butcher table is given by

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

1.2 Implementation of explicit Runge-Kutta methods

Below you will find the implementation a general solver class `ExplicitRungeKutta` which at its initialization takes in a Butcher table and has `__call__` function

```
def __call__(self, y0, f, t0, T, n):
```

and can be used like this

```
# Define Butcher table
a = np.array([[0, 0, 0],
              [1.0/3.0, 0, 0],
              [0, 2.0/3.0, 0]])

b = np.array([1.0/4.0, 0, 3.0/4.0])

c = np.array([0,
              1.0/3.0,
              2.0/3.0])

# Define number of time steps
n = 10

# Create solver
rk3 = ExplicitRungeKutta(a, b, c)

# Solve problem (applies __call__ function)
ts, ys = rk3(y0, t0, T, f, Nmax)
```

The complete implementation is given here:

```
class ExplicitRungeKutta:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def __call__(self, y0, t0, T, f, Nmax):
        # Extract Butcher table
```

```
        a, b, c = self.a, self.b, self.c

        # Stages
        s = len(b)
        ks = [np.zeros_like(y0, dtype=np.double) for s in range(s)]

        # Start time-stepping
        ys = [y0]
        ts = [t0]
        dt = (T - t0)/Nmax

        while(ts[-1] < T):
            t, y = ts[-1], ys[-1]

            # Compute stages derivatives k_j
            for j in range(s):
                t_j = t + c[j]*dt
                dY_j = np.zeros_like(y, dtype=np.double)
                for l in range(j):
                    dY_j += dt*a[j,l]*ks[l]

                ks[j] = f(t_j, y + dY_j)

            # Compute next time-step
            dy = np.zeros_like(y, dtype=np.double)
            for j in range(s):
                dy += dt*b[j]*ks[j]

            ys.append(y + dy)
            ts.append(t + dt)

        return (np.array(ts), np.array(ys))
```

Example 1.1. *Implementation and testing of the improved Euler method.*

We implement the **improved explicit Euler** from above and plot the analytical and the numerical solution. Finally, we determine the convergence order.

```
# Define Butcher table for improved Euler
a = np.array([[0,   0],
              [0.5, 0]])
b = np.array([0, 1])
c = np.array([0, 0.5])

# Create a new Runge Kutta solver
rk2 = Explicit_Runge_Kutta(a, b, c)

t0, T = 0, 1
y0 = 1
lam = 1
Nmax = 10

# rhs of IVP
f = lambda t,y: lam*y

# the solver can be simply called as before, namely as function:
ts, ys = rk2(y0, t0, T, f, Nmax)

plt.figure()
plt.plot(ts, ys, "c--o", label="$y_{\mathrm{heun}}$")

# Exact solution to compare against
y_ex = lambda t: y0*np.exp(lam*(t-t0))
```

```python
# Plot the exact solution (will appear in the plot above)
plt.plot(ts, y_ex(ts), "m-", label="$y_{\mathrm{ex}}$")
plt.legend()

# Run an EOC test
Nmax_list = [4, 8, 16, 32, 64, 128]

errs, eocs = compute_eoc(y0, t0, T, f, Nmax_list, rk2, y_ex)
print(errs)
print(eocs)

# Do a pretty print of the tables using panda

import pandas as pd
from IPython.display import display

table = pd.DataFrame({'Error': errs, 'EOC' : eocs})
display(table)
```

1.3 Order conditions for Runge-Kutta Methods

The convergence theorem for one-step methods gave us some necessary conditions to guarantee that a method is convergent order of p:

"consistency order p" + "Increment function satisfies a Lipschitz condition" $\Rightarrow$ "convergence order p.

"local truncation error behaves like $\mathcal{O}(\tau^{p+1})$" + "Increment function satisfies a Lipschitz condition" $\Rightarrow$ "global truncation error behaves like $\mathcal{O}(\tau^{p})$"

It turns out that for f is at least C^1 with respect to all its arguments then the increment function Φ associated with any Runge-Kutta methods satisfies a Lipschitz condition. Thus the next theorem

Theorem 1.1. *Order conditions for Runge-Kutta methods.*

A Runge–Kutta method has consistency order p if and only if all the conditions up to and including p in the table below are satisfied.

p	conditions
1	$\sum b_i = 1$
2	$\sum b_i c_i = 1/2$
3	$\sum b_i c_i^2 = 1/3$
	$\sum b_i a_{ij} c_j = 1/6$
4	$\sum b_i c_i^3 = 1/4$
	$\sum b_i c_i a_{ij} c_j = 1/8$
	$\sum b_i a_{ij} c_j^2 = 1/12$
	$\sum b_i a_{ij} a_{jk} c_k = 1/24$

where sums are taken over all the indices from 1 to s.

Proof. Without proof.

Example 1.2. *Applying order conditions to Heun's method.*

Apply the conditions to Heun's method, for which $s = 2$ and the Butcher tableau is

$$\begin{array}{c|cc} c_1 & a_{11} & a_{12} \\ c_2 & a_{21} & a_{22} \\ \hline & b_1 & b_2 \end{array} = \begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}.$$

The order conditions are:

$p = 1$ $\qquad b_1 + b_2 = \dfrac{1}{2} + \dfrac{1}{2} = 1$ $\qquad$ OK

$p = 2$ $\qquad b_1 c_1 + b_2 c_2 = \dfrac{1}{2} \cdot 0 + \dfrac{1}{2} \cdot 1 = \dfrac{1}{2}$ $\qquad$ OK

$p = 3$ $\qquad b_1 c_1^2 + b_2 c_2^2 = \dfrac{1}{2} \cdot 0^2 + \dfrac{1}{2} \cdot 1^2 = \dfrac{1}{2} \neq \dfrac{1}{3}$ $\qquad$ Not satisfied

$$\begin{aligned} b_1(a_{11}c_1 + a_{12}c_2) + b_2(a_{21}c_1 + a_{22}c_2) &= \frac{1}{2}(0 \cdot 0 + 0 \cdot 1) + \frac{1}{2}(1 \cdot 0 + 0 \cdot 1) \\ &= 0 \neq \frac{1}{6} \end{aligned}$$

Not satisfied

The method is of order 2.

Theorem 1.2. *Convergence theorem for Runge-Kutta methods.*

Given the IVP $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y}), \boldsymbol{y}(0) = \boldsymbol{y}_0$. Assume $f \in C^1$ and that a given Runge-Kutta method satisfies the order conditions from Theorem 1.1 up to order p. Then the Runge-Kutta method is convergent of order p.

Proof. Without proof.