# Exercise #12

11. April 2023

Exercises marked with a (J) should be handed in as a Jupyter notebook.

**Problem 1.** (Numerical solution of ODEs )

(J) In this problem we will implement Euler's method, second order Taylor's method, and Heun's method, and use them to approximate the solution to the ODE,

$$y' = (1 - 2t)y, \quad y(0) = 1.$$

The exact solution to this equation is $y(t) = e^{t-t^2}$.

a) Implement Euler's method, and compute an approximation of $y(1)$, using a step size equal to 0.5.

b) Do the same using Heun's method and the second order Taylor method. The second order Taylor method is given by

$$y_{n+1} = y_n + hf(t_n, y_n) + \frac{h^2}{2} f'(t_n, y_n),$$

where $f(t, y) = y'(t)$ and thus $f'(t, y) := \frac{\partial f}{\partial t}(t, y) = y''(t)$.

c) We now want to approximate the convergence orders of these methods numerically. Recall that we defined the global error,

$$\epsilon_g := \max_i |y(t_i) - y_i|.$$

If we assume that $\epsilon_g(h) \approx Mh^p$, for some $M > 0$, we have,

$$\log\left(\frac{\epsilon_g(h_1)}{\epsilon_g(h_2)}\right) \approx p \log\left(\frac{h_1}{h_2}\right).$$

Compute the global error of the methods from a)-c) using $h_1 = 10^{-2}$ and $h_2 = 10^{-3}$, where $t_i = ih, i = 1, \ldots, \frac{1}{h}$. Use this to approximate the convergence order, $p$, for each of the three methods.

d) We can also approximate the convergence order by plotting $\log(\epsilon_g(h)) = \log(M) + p\log(h)$ versus $\log(h)$, and inspecting the slope of the function.

Plot $\log(\epsilon_g(h))$ versus $\log(h)$ for $h = 10^{-2}, 10^{-3}, 10^{-4}$ for each of the three methods.

**Solution.**

See the ipython notebook `numerical-ode-complete.ipynb`.

**Problem 2.** (System of ODEs)

Let $a > 0$, $b > 0$, $c > 0$, $d > 0$ be constants. Write the second order linear ODE,

$$\begin{aligned} ay + by' + cy'' + d &= 0, \\ y(0) &= -1, \\ y'(0) &= 1, \end{aligned}$$

as a linear system of first order ODEs, and perform one step of Euler's method (resulting in an expression that includes the constants $a, b, c, d$) with step size 1.

**Solution.**

We start by setting $w_1 = y$ and $w_2 = y'$. Inserted into the ODE, this gives

$$\begin{aligned} w_1' &= w_2 \\ w_2' &= y'' = -\frac{1}{c}(aw_1 + bw_2 + d) \end{aligned}$$

This can be expressed by

$$\boldsymbol{w}' = A\boldsymbol{w} + v,$$

where

$$\boldsymbol{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}, \quad A = \begin{pmatrix} 0 & 1 \\ -\frac{a}{c} & -\frac{b}{c} \end{pmatrix}, \quad v = \begin{pmatrix} 0 \\ -\frac{d}{c} \end{pmatrix}.$$

Also $\boldsymbol{w}(0) = (-1, 1)$. One step of Euler's method with step size 1 in this case gives

$$\boldsymbol{w}_1 = \boldsymbol{w}(0) + A\boldsymbol{w}(0) + v = \begin{pmatrix} -1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ \frac{a}{c} - \frac{b}{c} \end{pmatrix} + \begin{pmatrix} 0 \\ -\frac{d}{c} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 + \frac{1}{c}(a - b - d) \end{pmatrix}.$$

**Problem 3.** (Order of a Runge–Kutta method)

In this exercise we will study a Runge–Kutta method that is given by

$$k_1 = y_n + hf(t_n, y_n) \tag{1}$$

$$k_2 = y_n + hf(t_n + \frac{h}{3}, y_n + \frac{k_1}{3}) \tag{2}$$

$$k_3 = y_n + hf(t_n + h\frac{2}{3}, y_n + \frac{2}{3}k_2) \tag{3}$$

$$y_{n+1} = y_n + \frac{h}{4}(k_1 + 3k_3) \tag{4}$$

a) Present the method in the form of a Butcher tableau.

b) Decide the order of the method.

**Solution.**

a) This is Heun's three-stage method with a Butcher tableau given by

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 \\ 2/3 & 0 & 2/3 & 0 \\ \hline & 1/4 & 0 & 3/4 \end{array}$$

b) Next we check the order conditions:

$p = 1$ $$b_1 + b_2 + b_2 = \frac{1}{4} + 0 + \frac{3}{4} = 1$$ OK

---

$p = 2$ $$b_1c_1 + b_2c_2 + b_3c_3 = \frac{1}{4} \cdot 0 + 0 \cdot \frac{1}{3} + \frac{3}{4} \cdot \frac{2}{3} = \frac{1}{2}$$ OK

---

$p = 3$ $$b_1c_1^2 + b_2c_2^2 + b_3c_3^2 = \frac{1}{4} \cdot 0^2 + 0 \cdot \frac{1}{3^2} + \frac{3}{4} \cdot \frac{2^2}{3^2} = \frac{1}{3}$$ OK

$$b_1(a_{11}c_1 + a_{12}c_2 + a_{13}c_3) + b_2(a_{21}c_1 + a_{22}c_2 + a_{23}c_3) + b_3(a_{31}c_1 + a_{32}c_2 + a_{33}c_3)$$
$$= 0 + 0 + \frac{3}{4} \cdot \frac{2}{3} \cdot \frac{1}{3} = \frac{1}{6}$$ OK

---

$p = 4$ $$b_1c_1^3 + b_2c_2^3 + b_3c_3^3 = \frac{1}{4} \cdot 0^3 + 0 \cdot \frac{1^3}{2^3} + \frac{3}{4} \cdot \frac{2^3}{3^3} = \frac{2}{9} \neq \frac{1}{4}$$ Not satisfied

We see that up to $p = 3$, the conditions are satisfied. The method is therefore of order 3.

**Problem 4.** (Lipschitz continuity)

Decide if the following functions are Lipschitz continuous for all $t, y \in \mathbb{R}$.

a) $f(t, y) = \dfrac{y}{t^2}$.

b) $f(t, y) = \dfrac{\sin(t)}{t} y$

**Solution.**

a) We compute directly,

$$|f(t,y) - f(t,x)| = \left|\frac{1}{t^2}(y-x)\right| = \frac{1}{t^2}|y-x|.$$

Hence, $f(t,y)$ cannot be Lipschitz, since we can choose $t$ arbitrarily close to $0$.

b) We have that,

$$|f(t,y) - f(t,x)| = \left|\frac{\sin(t)}{t}(y-x)\right| \leq |y-x|,$$

since $|\frac{\sin(t)}{t}| \leq 1$. Therefore, the function is Lipschitz continuous.

**Problem 5.** (Implementation of an ODE solver)

```
import numpy as np

f = lambda t,y : 2/t**2*y
t0, tend = 1, 2
y0 = 1
N = 10

y = np.zeros(N+1)
t = np.zeros(N+1)
y[0] = y0
t[0] = a

for n in range(N):
    k1 = f(t[n],y[n])
    k2 = f(t[n]+0.5*h, y[n]+0.5*h*k1)
    y[n+1] = y[n] + h*k2

print('t=',t)
print('y=',y)
```

a) There are three bugs in this code. Two that prevents it from running at all, and one which causes a completely nonsense output. Find and correct the errors.

b) Which mathematical problem does this code intend to solve numerically?

c) Which specific algorithm has been applied to the problem? No specific name is required, but present the method in the form of a Butcher tableau, and decide the order of the method.

d) Find the first two elements of the NumPy vector y, given that point a) is accomplished.

**Solution.**

```python
import numpy as np
f = lambda t,y : 2/ t**2*y
t0, tend = 1, 2
y0 = 1
N = 10
y = np.zeros(N +1)
t = np.zeros(N +1)
y[0] = y0
t[0] = t0                    #Assigning a starting time
h = (tend-t0)/N              #Need to define h
for n in range (N):
    k1 = f(t[n],y[n])
    k2 = f(t[n]+0.5* h , y[n]+0.5* h * k1)
    y[n+1] = y[n] + h*k2
    t[n+1] = t[n] + h        #Need to update timestep
print('t=',t)
print('y=',y)
```

a) The corrected version is written above, with comments for where the code is changed. The errors that made the code not run were that $t[0]$ was not set to $t_0$ but to some undefined variable $a$ and that $h$ was not defined. In addition, there were no computations of new timesteps, which made the output wrong.

b) This problem tries to solve the initial value problem

$$y' = \frac{2}{t^2}y, \qquad y(1) = 1,$$

on the interval $[1, 2]$.

c) The method presented as a Butcher tableau:

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

This method is known as the explicit midpoint method. Next, we check the order conditions:

| | | |
|---|---|---|
| $p = 1$ | $b_1 + b_2 = 0 + 1 = 1$ | OK |
| $p = 2$ | $b_1 c_1 + b_2 c_2 = 0 + 1 \cdot \frac{1}{2} = \frac{1}{2}$ | OK |
| $p = 3$ | $b_1 c_1^2 + b_2 c_2^2 = 0 + 1^2 \cdot \frac{1^2}{2^2} = \frac{1}{4} \neq \frac{1}{3}$ | Not satisfied |

We see that up to $p = 2$, the conditions are satisfied. The method is therefore of order 2.

If we run the code, we get that the first two elements of $y$ are 1. and 1.19954649.

**The next exercises are optional and should not be handed in!**

**Problem 6.** (Adaptivity)

Consider the following implementation of a Runge–Kutta method:

```
import numpy as np
import matplotlib.pyplot as plt

T = 2.0
y0 = 0.0
t = 0.0
h = 0.25

def f(t,y):
    return  t*np.exp(-y)

ys = [y0]
ts = [t]

while(t+h < T):
    t, y = ts[-1], ys[-1]
    k1 = f(t,y)
    k2 = f(t+h/2, y + h*k1/2)
    k3 = f(t+3*h/4, y + 3*h*k2/4)
    k4 = f(t+h, y + h*(2*k1 + 3*k2 + 4*k3)/9)
    y = y + h*(7*k1/24 + k2/4 + k3/3 + k4/8)
    ys.append(y)
    ts.append(t + h)

plt.plot(ts, ys, 's-')
```

a) Based on the implementation above, write down the Butcher tableau for this RK method.

b) If we run the code, how many time steps will be computed?

c) Based on the tableau, determine the order of this method.

d) Considering the same initial value problem and time-step size as in Problem 1, compute the first time step using the method implemented above.
**Hint:** before you actually compute the stage derivatives $k_i$, check whether you can re-use any of the calculations done for Problem 1.

e) Consider now the combination of this method with the one from Problem 1, to create an adaptive scheme. Remember the error estimate:

$$\hat{\epsilon}_{n+1} = \left|y_{n+1} - y^*_{n+1}\right| = h\left|\sum_{i=1}^{s}(b_i - b_i^*)k_i\right|,$$

with the superscript $*$ referring to the lowest-order method among the two. Based on the calculations done so far, compute $\hat{\epsilon}_1$.

f) Comparing $\hat{\epsilon}_1$ with the tolerance tol $= 0.001$, check if the first step we computed is acceptable and, if not, compute the new time-step size $h_{\text{new}}$ based on the formula developed in class:

$$h_{\text{new}} = P\left(\frac{\text{tol}}{\hat{\epsilon}_1}\right)^{\frac{1}{p+1}} h,$$

with $P = 0.9$.

**Solution.**

a) The Butcher tableau is given by

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 3/4 & 0 & 3/4 & 0 & 0 \\ 1 & 2/9 & 3/9 & 4/9 & 0 \\ \hline & 7/24 & 1/4 & 1/3 & 1/8 \end{array}$$

b) Since $h$ is the length of each subinterval, we can compute the number of subintervals as

$$N = T/h = 2/0.5 = 8,$$

so that the number of steps is $N + 1 = 9$.

c) The order is determined by the following expressions

$$\sum_{i=1}^{s} b_i = \frac{7}{24} + \frac{1}{4} + \frac{1}{3} + \frac{1}{8} 1$$

$$\sum_{i=1}^{s} b_i c_i = \frac{7}{24} \times 0 + \frac{1}{4} \times \frac{1}{2} + \frac{1}{3} \times \frac{3}{4} + \frac{1}{8} \times 1 = \frac{1}{2}$$

$$\sum_{i=1}^{s} b_i {c_i}^2 = \frac{7}{24} \times (0)^2 + \frac{1}{4} \times \left(\frac{1}{2}\right)^2 + \frac{1}{3} \times \left(\frac{3}{4}\right)^2 + \frac{1}{8} \times (1)^2 = \frac{6}{16} \neq \frac{1}{3}.$$

The method is second-order consistent ($p = 2$).

d) Notice that you don't need to compute $k_1$, $k_2$ and $k_3$ again, since they will be the same as for Ralston's method (Problem 1). So all you have to do is evaluate $k_4$, then use all the $k_i$'s to finally compute the first step. The fourth stage derivative is

$$k_4 = f\left(t_n + h, y_n + h\left[\frac{2}{9}k_1 + \frac{1}{3}k_2 + \frac{4}{9}k_3\right]\right) = f(0.48, 0.108843) = 0.48\mathrm{e}^{-0.108843} \approx 0.4305\,.$$

Then:

$$\begin{aligned} y_1 &= y_0 + h\left[\frac{7}{24}k_1 + \frac{1}{4}k_2 + \frac{1}{3}k_3 + \frac{1}{8}k_4\right] \\ &= y_0 + h\left[\frac{7}{24} \times 0 + \frac{1}{4} \times 0.24 + \frac{1}{3} \times 0.330202 + \frac{1}{8} \times 0.4305\right] \approx 0.1074622 \end{aligned}$$

e) $\hat{\epsilon}_1 = |0.108843 - 0.1074622| \approx 0.001381$.

f) Since $\hat{\epsilon}_1 \approx 0.001381 > 0.001$, we must recompute the first step with a new time-step size:

$$h_{\text{new}} = \gamma \left(\frac{\text{tol}}{\hat{\epsilon}_1}\right)^{\frac{1}{p+1}} h = 0.9 \left(\frac{0.001}{0.001381}\right)^{\frac{1}{2+1}} 0.48 \approx 0.388\,.$$

**Problem 7.** (Stability)

For an ODE $y'(t) = f(t, y)$, the so-called *explicit midpoint method* is given by

$$y_{n+1} = y_n + hf(t_n + 0.5h,\ y_n + 0.5hf(t_n, y_n))\,.$$

Consider, in particular, the linear autonomous equation where $f(y) = \lambda y$, with $\lambda \in \mathbb{C}$.

a) Write down the Butcher tableau for this Runge–Kutta method.

b) Determine the stability function $R(z)$ of the explicit midpoint method, that is, the function that allows us to write $y_{n+1} = [R(h\lambda)]\, y_n$.

c) Determine the stability region $\mathcal{S}$.

d) For $\lambda = -20$, what is the interval of time-step sizes $h$ for which we get a stable solution?

**Solution.**

a) We can reinterpret the expression for $y_{n+1}$ as a Runge–Kutta algorithm, via

$$\begin{aligned} k_1 &= f(t_n, y_n) = f(t_n + c_1 h, y_n + a_{11} h k_1 + a_{12} h k_2) \\ k_2 &= f(t_n + 0.5h, y_n + 0.5hk_1) = f(t_n + c_2 h, y_n + a_{21} h k_1) \\ y_{n+1} &= y_n + hk_2 = y_n + h(b_1 k_1 + b_2 k_2)\,, \end{aligned}$$

that is, $c_2 = 0.5$, $a_{21} = 0.5$, $b_2 = 1$ and the other entries are 0. We therefore get the following tableau:

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

b) We rewrite the expression for the explicit midpoint method:

$$\begin{aligned} y_{n+1} &= y_n + h\lambda(y_n + 0.5h\lambda y_n), \\ &= y_n + h\lambda y_n + 0.5h^2\lambda^2 y_n, \\ &= (1 + h\lambda + 0.5h^2\lambda^2)\, y_n\,. \end{aligned}$$

The stability function is then

$$R(h\lambda) = 1 + h\lambda + 0.5h^2\lambda^2.$$

c) The stability region is defined by setting $z = \lambda h$ and requiring $|R(z)| \leq 1$, that is,

$$\mathcal{S} = \left\{ z \in \mathbb{C} : |1 + z + 0.5z^2| \leq 1 \right\}.$$

d) For $\lambda \in \mathbb{R}$, we can write

$$1 + z + 0.5z^2 = \frac{(z+1)^2}{2} + \frac{1}{2} \geq \frac{1}{2} \quad \text{for all} \quad z \in \mathbb{R},$$

so we know that $R(\lambda h) > 0$, that is, $|R(\lambda h)| = R(\lambda h)$. Thus, the inequality we need to solve to have stability is simply $R(\lambda h) \leq 1$. For $\lambda = -20$, this means

$$1 - 20h + 0.5(-20h)^2 \leq 1\,.$$

Since the time-step size $h$ is always larger than zero, we can divide both sides of the inequality by $20h$ to get

$$-1 + 10h \leq 0\,,$$

that is, $h \leq 0.1$. The method will therefore be stable for any $h$ in the interval $(0, 0.1]$.