Math 5300, Fall 2022 (Roos) – Homework 4 – Solutions.

Due Wednesday, Nov 2.

Important: Please submit your homework as a single compressed pdf file (< 2 MB if possible, scans of handwritten work are okay, use an appropriate app) online via Blackboard. Problem 1 must appear on top of the first page.

Only Problem 1 will be graded. The other problems are strongly recommended, but will not be graded. Problems marked with an asterisk (*) may be more challenging.

1 (Graded). (i) Determine the exact solution to the IVP

$$\begin{cases} y' = \sin(x)y \\ y(1) = 2 \end{cases}$$

(ii) Implement the 4-stage Runge-Kutta method given by the Butcher tableau

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{3} & \frac{1}{3} & & & \\ \frac{2}{3} & -\frac{1}{3} & 1 & & \\ 1 & 1 & -1 & 1 & \\ \hline & \frac{1}{8} & \frac{3}{8} & \frac{3}{8} & \frac{1}{8} \end{array}$$

Test it on the IVP from (i) on the interval $I = [1, 6]$ (so $a = 5$): Generate a scatter plot of the approximate solution for step size $h = \frac{1}{5}$ (plot each point of the approximation, do not connect the points) and also plot the curve of the exact solution (in the same plot). Output the decimal value of the global approximation error

$$|E_N| = |y(6) - y_N|$$

where $y = y(x)$ is the exact solution from (i) and y_N is the approximation that your program produced (note $N = a/h = 25$).

(iii) Confirm that the order of the method is 4 by computing the approximation error for various small step sizes, say $h = 2^{-i}$ for $i = 2, \ldots, 9$, then plotting $\log |E_N|$ against $\log(h)$ and estimating the slope of the line (as shown in class or differently).

Note: To receive full credit for (ii), (iii) it suffices to include the source code and its output including plots. Include brief comments in your source code to explain what you are doing.

Solution: (i) This ODE can be solved by separation of variables:

$$\int \frac{dy}{y} = \int \sin(x)dx$$

$$\log|y| = -\cos(x) + C$$
$$|y(x)| = e^{-\cos(x)+C}$$
$$y(x) = \pm e^C e^{-\cos(x)}$$

The constant $\pm e^C$ can be any non-zero real number. Also note that $y(x) = 0$ is a solution. The general solution can be written as

$$y(x) = Ce^{-\cos(x)}$$

with a real parameter C (different C than above). We can determine C by plugging in the initial condition $y(1) = 2$:

$$2 = Ce^{-\cos(1)}$$

Therefore

$$C = 2e^{\cos(1)}$$

and the solution to the IVP is

$$y(x) = 2e^{\cos(1)-\cos(x)}.$$

As interval of definition we can use $I = (-\infty, \infty)$.

Common mistakes:

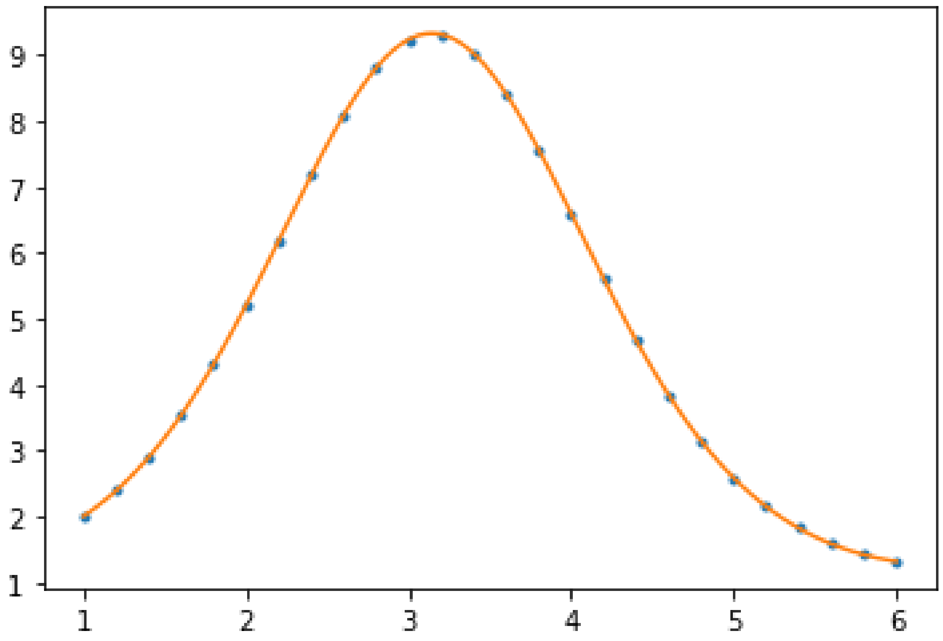
- An approximate decimal value was given for C instead of the exact value. This introduces an unnecessary error.
- The value for C was computed, but not plugged back into the general solution (no final answer given).

(ii) Following the implementation seen in class:

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

def onestep(F, x0, y0, a, N, Phi):
    y = np.zeros(N+1)
    h = a/N
    x = x0
    y[0] = y0
    for j in range(N):
        y[j+1] = y[j] + h*Phi(F, x, y[j], h)
        x += h
    return y

def rk38(F, x, y, h):
    k1 = F(x,y)
    k2 = F(x+ h/3, y+1/3*h*k1)
    k3 = F(x+ 2/3*h, y-1/3*h*k1+ h*k2)
    k4 = F(x+h, y + h*k1 - h*k2 + h*k3)
    return 1/8*k1 + 3/8*k2 + 3/8*k3 + 1/8*k4

def approx_and_plot(F, x0, y0, a, y_exact, N, method):
    x, y = np.linspace(x0, x0+a, N+1), onestep(F, x0, y0, a, N,
     method)
    plt.plot(x, y, ".")
    error = np.abs(y-y_exact(x))[-1]

    x = np.linspace(x0,x0+a,int(np.ceil(a*100)))
    plt.plot(x, y_exact(x))
    return error

F = lambda x,y: np.sin(x)*y
y_exact = lambda x: 2*np.exp(np.cos(1)-np.cos(x))
x0, y0, a = 1, 2, 5
approx_and_plot(F, x0, y0, a, y_exact, 25, method=rk38)
```

Code output:

```
1.5413897263893972e-05
```



Common mistakes:

- Plot did not meet specifications (it was supposed to be a scatter plot of the approximate solution with $h = 1/5$ and a curve plot of the exact solution in the same coordinate system).
- Code did not output global error or the given global error was too large (usually indicates a bug in the code – it can happen that a method is incorrectly implemented, so does not have the correct order, but is still consistent).
- Please never write code using pen and paper. Also avoid using camera photos of your screen.

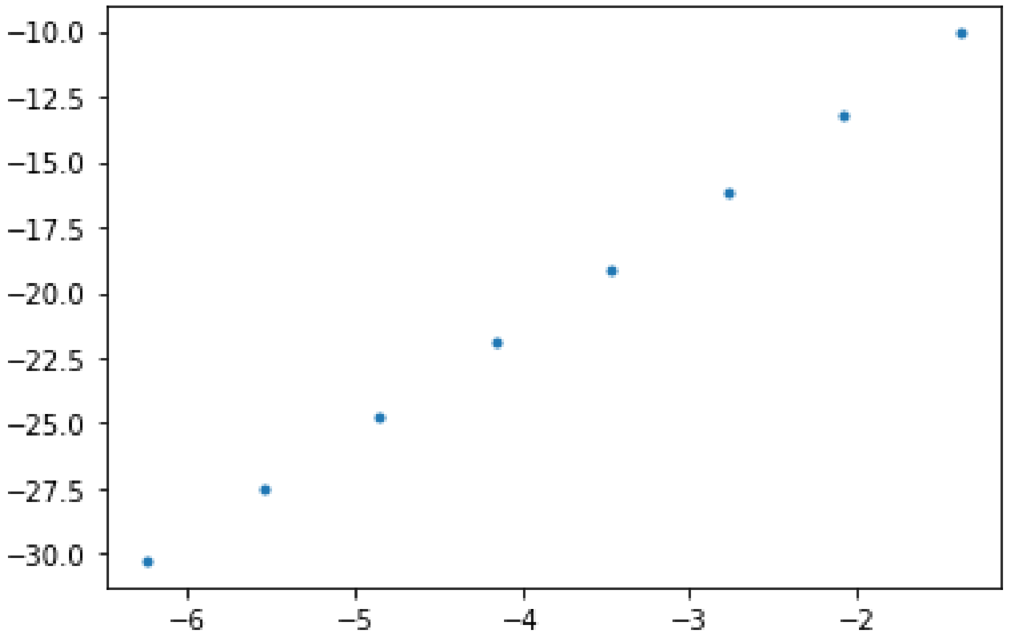
(iii) Using the code from above:

```
def get_errors(F, x0, y0, a, y_exact, N_values, method):
    errors = []
    for N in N_values:
        x, y = np.linspace(x0, x0+a, N+1), onestep(F, x0, y0, a
    , N, method)
        errors += [np.abs(y-y_exact(x))[-1]]
    return errors

N_values = np.array([a*2**i for i in range(2,9+1)])
errors = get_errors(F, 1, 2, a, y_exact, N_values, method=rk38)
print(errors)
```

We can plot the errors logarithmically as follows:

```
h_values = a/N_values
plt.plot(np.log(h_values), np.log(errors), '.')
```

The output is



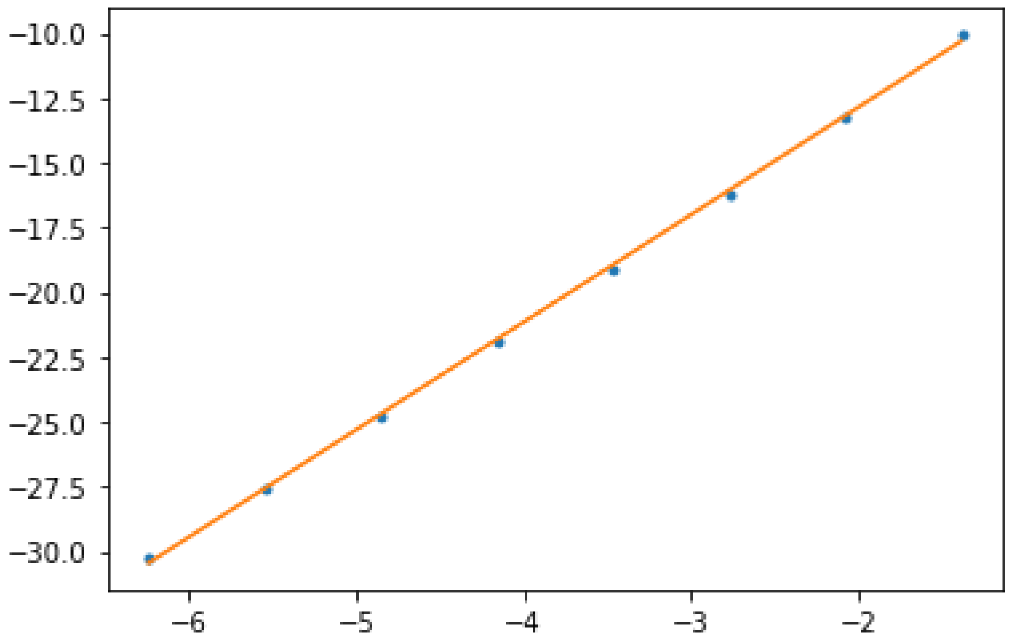
As expected, the scatter plot suggests that there is a linear relation between $\log(h)$ and the logarithm of the error. This confirms that we have used reasonable values for h (not too small, not too large). To determine the slope of the line one can use several methods (as discussed in class). One method not discussed in class is by linear regression:

```
from sklearn.linear_model import LinearRegression
x = np.log(h_values.reshape((-1,1)))
y = np.log(errors)
m = LinearRegression()
```

```
m = m.fit(x, y)
print("Slope of fitted line: %f"%m.coef_)
print("R^2 on training data: %f"%m.score(x, y))
plt.plot(x, y, '.', x, m.predict(x))
```

We use linear regression to fit the best line to the data (the standard algorithm is to minimize mean squared error). The output is

```
Slope of fitted line: 4.151299
R^2 on training data: 0.999569
```



Since we know the order of the method has to be an integer, this provides good evidence that the order of this method is 4 (which can be proved mathematically).

Common mistakes:

- Attempts at handwritten sketches of the error plot were not sufficient (it would be in theory, if you did it with sufficient accuracy).
- Using only two points for the slope comparison is sometimes not sufficient evidence (especially not when the plot is missing).
- Some ran the code for several h values manually and used Excel to produce the logarithmic plot and fit a line through it. This works here, but is not good practice (it's better to write a loop over h). (No points deducted.)

2. Consider the second-order linear ODE

$$y'' = -y + 2y'$$

(i) Determine the general solution.
(ii) Determine the solution with initial conditions $y(0) = 1, y'(2) = -1$.
(iii) Determine all solutions (that is, the general solution) of the inhomogeneous equation

$$y'' = -y + 2y' + 3.$$

Solution:

(i) This is a second-order linear ODE with constant coefficients and its characteristic polynomial is

$$P(u) = u^2 - 2u + 1 = (u-1)^2$$

The polynomial has a zero of multiplicity 2 at $u = 1$. Thus, the general solution is given by

$$y(x) = C_1 e^x + C_2 x e^x.$$

(ii) Plugging in the initial condition $y(0) = 1$ we get the equation

$$1 = C_1 e^0 + C_2 \cdot 0 \cdot e^0 = C_1,$$

so $C_1 = 1$. To plug in the other initial condition, we first need to compute

$$y'(x) = e^x + C_2 e^x + C_2 x e^x = e^x + C_2(1+x)e^x$$

Plugging in the initial condition $y'(2) = -1$ we get the equation

$$-1 = e^2 + C_2(1+2)e^2,$$

so

$$C_2 = -\tfrac{1}{3}(1 + e^{-2})$$

and

$$y(x) = e^x - \tfrac{1}{3}(1 + e^{-2}) x e^x.$$

(iii) We first need to determine a particular solution to the inhomogeneous equation. From looking at the inhomogeneity we guess that the constant function $y(x) = 3$ is a solution, which is indeed the case. By the superposition principle, the general solution to the inhomogeneous equation is therefore given by

$$y(x) = C_1 e^x + C_2 x e^x + 3.$$

3. (Extra Credit!) Extend your code from the in-class programming session in such a way that it covers also systems of first-order ODEs. Test your code on the system

$$y_1' = y_2, y_2' = -4y_1$$

with initial conditions $y_1(0) = 1$, $y_2(0) = 2$ (first determine the exact solution).

Produce graphs that show the exact solution beside the numerical solutions using methods A,B,C,D from Exercise 3.52. Produce graphs that demonstrate the different orders of methods A,B,C,D empirically using the given system (double-logarithmic error plots).

Comments:

- Submissions without pictures will not receive any credit.
- The problems asks for an implementation that works for systems of arbitrary size (i.e. arbitrarily many equations), but if you only implement it for systems of two equations to cover the given example, you may still receive some credit.
- To determine the exact solution of the system, you may want to recognize it as a linear second-order equation in disguise.

4. Recall the definition of linear multistep methods from class.

(i) Define reasonable notions of truncation error, consistency and order for linear multistep methods (review the corresponding notions for one-step methods). (You can compare with the actual definitions in Süli-Mayers, Ch. 12.6).

(ii) Prove that the truncation error converges to zero for the linear two-step Adams-Bashforth method

$$y_{n+2} - y_{n+1} = h(\tfrac{3}{2}F_{n+1} - \tfrac{1}{2}F_n)$$

and show that it has order 2.

Comment: For the solutions, see Süli-Mayers, Ch. 12.6.