Assignment #6

Due Monday, 31 March 2025, at the start of class

Please read sections 5.3–5.9 and 6.1–6.4 from the textbook (R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Diff. Eqns.*, SIAM Press 2007).

Problem P24. For an autonomous ODE system $u'(t) = f(u(t))$, compute the leading term in the local truncation error of the following 2 methods. For part **(a)**, please follow the style of Example 5.9.[1] For part **(b)** follow Example 5.11, which handles scheme (5.30), the explicit midpoint rule. That is, for part **(b)** you should show the simpler fact $\tau^n = O(k^2)$, without finding the leading-order coefficient.

(a) the 2-step BDF method (5.25).

Hint. Expand around t_{n+1}, to get τ^{n+1}.

(b) the explicit trapezoid method,

$$U^* = U^n + kf(U^n)$$
$$U^{n+1} = U^n + \frac{k}{2}\Big(f(U^n) + f(U^*)\Big),$$

which is $U^{n+1} = U^n + \frac{k}{2}\Big(f(U^n) + f\big(U^n + kf(U^n)\big)\Big)$ when combined.

Problem P25. **(a)** In preparation for problem **P26** below, write two solvers

```
function [tt,zz] = fe1(f,eta,t0,tf,N)
function [tt,zz] = rk4(f,eta,t0,tf,N)
```

which implement the forward Euler scheme $U^{n+1} = U^n + kf(t_n, U^n)$ and the classical RK4 scheme (5.33), respectively, to solve the ODE IVP in (5.1) and (5.2). The first input to these solvers is `function z = f(t,u)`.[2] The other inputs are a vector of initial values `eta` $= u(t_0)$, the initial time `t0`, the final time `tf`, and the number of equal-length steps (subintervals) `N`, respectively. Note that the time step is $k = (t_f - t_0)/N$. Each solver outputs the entire trajectory, so `tt` is a 1D array of length $N+1$ starting with t_0 and ending with t_f.[3] If $\eta \in \mathbb{R}^s$ then `zz` is a 2D array with s rows and $N+1$ columns; each column i gives the solution $u(t)$ at the ith time in `tt`.

(b) Solve the following problem exactly:

$$x'' + 4x = 0, \qquad x(0) = 1, \quad x'(0) = 0.$$

[1] For the multistep midpoint rule (5.23), Example 5.9 finds leading-order term $\frac{1}{6}k^2 u'''(t_n)$.

[2] Please use the MATLAB and `scipy.integrate` variable order here, namely $f(t,u)$, and not the book order "$f(u,t)$." Numerical libraries, and their black box solvers like `ode45` or `scipy.integrate.solve_ivp`, generally use $f(t,u)$. I don't know why LeVeque swapped it, but I advise against following him.

[3] Again, this is compatible with how the MATLAB and `scipy.integrate` solvers do things.

Also write this as a first order system, needed when setting-up numerical solvers.

(c) The problem in **(b)**, for example on the interval $[t_0, t_f] = [0, 5]$, makes a good test case. Demonstrate that the final-time, absolute numerical error of each solver in **(a)** converges at the expected rate as $k \to 0$.

Hint. What is the expected rate is for each method? There is no need to compute local truncation errors yourself, but you must know their orders.

Problem P26. *This is a real application. Perhaps it will help you appreciate our abstract notation for ODE systems, vector data types in our languages, and higher-order explicit ODE schemes. This problem has an exact solution,*[4] *but it is not used here.*

Consider the problem of two massive bodies (particles) with masses m_1 and m_2. They are attracted by gravity only. They travel in a plane so their positions are given by vector-valued functions $\mathbf{x}_i(t) = (x_i(t), y_i(t))$ for $i = 1, 2$. Newton's second law and Newton's law of gravity combine to say:

$$\begin{aligned} m_1 \mathbf{x}_1'' &= -G m_1 m_2 \frac{\mathbf{x}_1 - \mathbf{x}_2}{|\mathbf{x}_1 - \mathbf{x}_2|^3} \\ m_2 \mathbf{x}_2'' &= -G m_1 m_2 \frac{\mathbf{x}_2 - \mathbf{x}_1}{|\mathbf{x}_1 - \mathbf{x}_2|^3} \end{aligned} \tag{1}$$

We will consider the Earth and the Moon in isolation as our example. Thus the constants are $m_1 = 5.972 \times 10^{24}$ kg, $m_2 = 7.348 \times 10^{22}$ kg, and $G = 6.674 \times 10^{-11}\,\mathrm{m}^3\,\mathrm{kg}^{-1}\,\mathrm{s}^{-2}$. We measure t in seconds—convert all times to seconds inside the solver!—and x_i, y_i in meters. (*Please confirm that the units are consistent in system* (1).)

(a) By using notation $v_i = x_i', w_i = y_i'$ for $i = 1, 2$, write system (1) as a first-order ODE system of dimension $s = 8$, with solution column vector $u(t) \in \mathbb{R}^8$. Use the component ordering

$$\begin{aligned} u(t) &= \begin{bmatrix} x_1(t) & y_1(t) & x_2(t) & y_2(t) & v_1(t) & w_1(t) & v_2(t) & w_2(t) \end{bmatrix}^\top \\ &= \begin{bmatrix} u_1(t) & u_2(t) & u_3(t) & u_4(t) & u_5(t) & u_6(t) & u_7(t) & u_8(t) \end{bmatrix}^\top. \end{aligned}$$

That is, write system (1) in the form of (5.1) in the book:[5] $u'(t) = f(t, u(t))$. Then implement a single function

```
function z = fearthmoon(t,u)
```

which computes the right-hand-side function $f(t, u)$ of the ODE system.

(b) Here are some initial conditions which are vaguely like what they are in reality,[6] at least if you turned off all the gravity of other bodies and start the Earth at the origin: $t_0 = 0, x_1(0) = 0, y_1(0) = 0, v_1(0) = 0, w_1(0) = 0$, $x_2(0) = 3.844 \times 10^8$ meters, $y_2(0) = 0$, $v_2(0) = 0$, $w_2(0) = 1.022 \times 10^3\,\mathrm{m\,s}^{-1}$. Use these initial conditions to generate approximate solutions with $t_f = 35$ days.

---

[4] See, for example: https://www.diva-portal.org/smash/get/diva2:630427/FULLTEXT01.pdf

[5] In fact the right side of this ODE system does not have explicit dependence on t, but, to avoid confusion in the implementation, use the MATLAB/`scipy.integrate` variable ordering.

[6] I searched "earth moon distance meters" and "mean orbital velocity moon."

Now use each of the solvers from problem **P25** with $N = 40$ and $N = 960$, i.e. daily and hourly time steps, respectively. Also use `ode45()`, or comparable dual-order, adaptive Runge-Kutta black-box solver, using the default accuracy. That is, generate five numerical solutions.

Do not, of course, show me lots of numbers. Make basic plots of the computed trajectories, i.e. the x_i, y_i values. Describe in a few words what you see, and how these results relate to the local truncation error of the schemes in **P25**.

(c) How long in days is a lunar month, using your best computation from part **(b)**?

Problem P27. **(a)** For ODE systems of form $u'(t) = Au(t) + g(t)$, where A is a square matrix and $g : \mathbb{R} \to \mathbb{R}^s$, build a BDF2 multistep solver for the initial-value problem:

```
function [tt,zz] = bdf2(A,g,eta,t0,tf,N)
```

See equation (5.25). Address the necessary linear algebra by using the default black box in your language, e.g. backslash in Matlab, or by pre-factoring.[7] Also, choose a scheme for taking the first step which preserves the order of the method.

(b) Solve exactly the same problem as in **P25(b)**, as a test case. Demonstrate that the final-time numerical error converges at the expected rate as $k \to 0$.

[7]Either will get full credit, but the latter should be faster for large s.