Math 563, Spring 2020
*Last updated: 2/27/20*

# HOMEWORK 6 (DUE WED. 3/4)

**Reading:** Section 4.1, 4.2 and 4.5 of A&P are suggested, while 4.3 provides some further detail on RK method coefficients. If you don't have access to the book, post to Piazza and excerpts can be provided.

**Code to turn in:**

- Your AB2, RK4 and RKF code.

‘

## 1. Problems

**Q1.** Implement the system version of AB2 from HW 5 and solve

$$y'' = xy, \quad y(0) = 0.355, \quad y'(0) = -0.26, \qquad x \in [-10, 10]$$

Make a plot of the solution $y(x)$ and its derivative $y'(x)$ in the given interval (note that the interval is two-sided here, not just $x > 0$!).[1]

**Q1b [new parts].** a) Check that your order is correct by testing the error in each component at $x = -10$ using the trick noted in Q1a of HW3 (for estimating $p$).

b) Obtain an estimate for $y(10)$ with a relative error of $10^{-4}$. Suppose a doubtful reader points out that for $x < 0$, the solution looks nice and bounded, but for $x > 0$, it appears to blow up, which looks suspicious. Argue for the correctness of your solution and for the estimate of $y(10)$.

---

**Q2 (AB stability).** Derive an expression for the boundary of the region of absolute stability for AB2 and plot it. Calculate the interval of absolute stability exactly.

---

**Q3 (RK error terms) [AC].** For this problem, assume that the ODE is $y' = f(y)$.

a) Derive an expression for the error in the modified Euler method from class by adding an extra term to the Taylor expansions.

b) Show that a two-stage explicit RK method cannot be third order by finding a term in $(y_{n+1} - y_n)/h$ that cannot be matched by the other side.

[1] **Note:** A correction - the solution here is related to the Airy function (which goes to zero as $x \to \pm\infty$) but has another component.

**Q4 (sacrificing order for stability in RK methods).** An RK method with $m$-stages does not need to be derived with the best possible order if other properties are needed.

A 'stabilized Runge-Kutta method' chooses coefficients to improve its stability region.

a) Consider the explicit, two stage RK methods (from class). If it is second order, what is the stability interval $(-b, 0)$? Show that if the two stage method is only first order, the coefficients can be chosen to obtain a larger stability interval (by a factor of at least two).

b) [AC] Find the method that optimizes this interval (largest possible).

c) Suppose you are solving an ODE $y' = f(t, y)$ with $|\partial f/\partial y| \approx 200$ and the solution looks like $y(t) = \sin(10t)$. You need to compute a qualitatively correct approximation. If Backward Euler takes $W_I$ time per step and method (b) takes $W_F$ time per step, when is it better to use (a) rather than the implicit method?

**Remark:** The point is that while good implicit methods are 'better' when avoiding stability constraints, it is occasionally faster to use an explicit method anyway. An equation that has a significant stability constraint, but not so much that explicit methods are impossible to use, is called **moderately stiff**.

---

**Q5 (why step size control matters).** Do Problem 4.12 in A&P (excerpt posted on Piazza along with code for the ODE function). The 'classical' RK method is the one introduced in class (with $1/6, 1/3, 1/3, 1/6$ as weights).

**Note:** This ODE comes from the 'restricted three-body problem', a special case of a notoriously ill-behaved system of ODEs; search that phrase for details on the physics.

---

**Q6.** Implement the Runge-Kutta-Fehlberg method (coefficients provided in HW section of Piazza) to solve the first order system $\mathbf{y}' = F(t, \mathbf{y})$. Use the estimate from class to implement the step size selection, replacing $|u_n - \tilde{u}_n|$ with the norm of the difference.

Your code should take as an input a tolerance `tol` or tolerances `atol` and `rtol` (absolute/relative) used for bounding $|\tau_n| <$ `tol`.

Test your function on the example from Q5 - does the adaptive stepping help?

*Note 1: use vector arithmetic to simplify the code.*

*Note 2: The size of the output array is not known in advance. It's fine to have it grow per step. Numpy arrays are not convenient to grow, while Python's 'append' does okay.*