Numerical Methods for Differential Equations: Homework 4

Due by 2pm Tuesday 10th November. ***Please submit your hardcopy at the start of lecture.*** *The answers you submit should be attractive, brief, complete, and should include program listings and plots where appropriate. The use of "publish" in* MATLAB *is one possible approach.*

Problem 1: Step-size control Here is a simple *embedded Runge–Kutta method* consisting of forward Euler and modified Euler sharing common *intermediate stage values:*

$$\begin{aligned} a &= f(t_n, v^n), \\ b &= f(t_n + \frac{k}{2}, v^n + \frac{k}{2}a), \\ v^{n+1} &= v^n + ka, \\ \tilde{v}^{n+1} &= v^n + kb. \end{aligned}$$

Write a function that takes exactly one step of size k and returns both $\tilde{v}^{n+1}$ and v^{n+1} (from a common state v^n). Your code should work for vector v^n and f. Show the outputs of your code for the three-body problem from HW3, using $k = 0.001$.

Now we will write a code which does adaptive time-stepping:

- Estimate the *one-step error* in v^{n+1} using the difference of the two solutions.

  $$\text{Err} \approx v^{n+1} - \tilde{v}^{n+1}.$$

  (Think about why this should be so?) We want the error at each step less than a specified tolerance `ATOL`. Maybe 10^{-6} is a reasonable value.

- If the error is too large, reject the step and try again (with a smaller step).

- If the error is OK, accept the step, keeping v^{n+1}. Now take another one (possibly with a larger step), using v^{n+1} as the new initial condition (i.e., discard $\tilde{v}^{n+1}$.

- In either case, you need a new step size (to either repeat the step, or for the next one). Here's one approach. From Taylor series analysis, we expect the error to be of the form

  $$\|\text{Err}\|_\infty = Ck^{q+1} \quad (+ HOT, \text{ ignore them}),$$

  where $q = 1$ is order of the forward Euler method and C is a constant. What we want is the optimal step k_{opt} such that

  $$\text{ATOL} = Ck_{\text{opt}}^{q+1}.$$

  So solve (on paper) these equations for k_{opt}.

- Multiply your chosen k_{opt} by a safety factor of 0.8 (as rejecting steps is expensive).

Run your code on the three-body problem from HW3 Question 2 (and [1, Ch. II, pg 129]). Make a plot. Does it work better than a constant step? Count how many rejected steps and accepted steps you make.

(In practice, another algorithm would estimate an initial step size as well; this is not necessary for your implementation: use $k = 0.001$ for the first step).

Problem 2: Convergence Study This is [2, Exercise 1.2], m-files referred to are from `http://faculty.washington.edu/rjl/fdmbook/matlab`

1. Use the method of undetermined coefficients to set up the 5×5 Vandermonde system that would determine a fourth-order accurate finite difference approximation to $u''(x)$ based on 5 equally spaced points,

   $$u''(x) = c_{-2}u(x-2h) + c_{-1}u(x-h) + c_0u(x) + c_1u(x+h) + c_2u(x+2h) + O(h^4).$$

2. Compute the coefficients using the Matlab code `fdstencil.m` available from the website above and check that they satisfy the system you determined in part (a).

3. Test this finite difference formula to approximate $u''(1)$ for $u(x) = \sin(2x)$ with values of h from the array `hvals = logspace(-1, -4, 13)`.

4. Make a table of the error vs. h for several values of h and compare against the predicted error from the leading term of the expression printed by `fdstencil`. You may want to look at the m-file verb`chap1example1.m` for guidance on how to make such a table.

   Also produce a log-log plot of the absolute value of the error vs. h.

   You should observe the predicted accuracy for larger values of h. For smaller values, numerical cancellation in computing the linear combination of u values impacts the accuracy observed.

Problem 3: Smoothness

1. Using the *method of undetermined coefficients*, re-derive the "1 -2 1" rule again, being careful to work out the Big Oh error term. How smooth do you expect a function needs to be to observe this error behaviour? (that is, how many derivatives must exist and be continuous?)

2. Design and perform a numerical test that demonstrates the error term of the second-order centered second derivative "1 -2 1" rule behaves as expected with regards to smoothness of the function. Specifically, what happens if the function is just a bit less smooth then the theory requires?

You might reuse you answer here in the next exercise.

Problem 4: A collaborative test set (Required for 607E, optional for 405.)

See the `diff_matrices1d.m` code from https://github.com/cbm755/fdmatrices. Fork that repository and clone it to your local computer.

Now write at least one test function with a name like `test_conv_study_sin` (that is, starting with the lowercase characters "`test_`").

- Your test should return 'true' if the test passed, otherwise 'false'.
- It should take two arguments: `test_mine(plots, verbose)`. Both should default to false. That is, I should be able to write `test_mine()` and it will be neither verbose nor make any plots. Of course your function might not need to make any plots even if these are true.
- Your test should be reasonably fast (say max 30 seconds).
- Your test should have a reasonable name that says what it does.
- Your test should be reasonably different from others'. For example, if someone else tests `Dxx` you could test `Dxf`.
- A good idea for a test is a convergence study: pick a problem, choose one of the types of boundary conditions, and check that the error decreases appropriately (e.g., close to 4 when h goes down by 2).
- Other possible test ideas: (i) `diff_matrices1d` gives the same matrix as a small hardcoded case you worked out by-hand, (ii) it does something expected for unreasonable input (nan, inf), (iii) it gives you a matrix of the correct type (i.e., sparse not dense), or anything else you can think of.
- Submit your test as a "pull request" to the original repository. You should review one of your colleagues' merge requests and give it a "+1" (or suggest changes until you can give it a "+1")

You should occasionally fetch the latest version ("git fetch", "git pull") as the pull requests (hopefully) get merged. For your written work, submit a screenshot showing your pull request, and another screenshot showing your review of a colleague's pull request.

For bonus points, we should get "Travis CI" running continuous integration tests for us...

References

[1] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving ordinary differential equations I: Nonstiff problems*. Springer-Verlag, second edition, 1993.

[2] R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. SIAM, 2007.