EP 501 FA 2020 final examination

December 6, 2020

Instructions: This is a take-home exam (work ALL problems) to be turned in 48 hours from the time it is posted (see canvas due date), unless you have notified me of a medical issue, family emergency, or other extenuating circumstance. *Unexcused late exams will not be graded.*

I will be checking my email regularly over the next 48 hours to answer questions. If you do not understand a question, please ask for a clarification.

You MAY NOT work in groups or discuss details of the solution to these or similar problems with other students or faculty. You also MAY NOT use online "work this problem for me" resources (i.e. you must complete this problem without direct assistance of another human being). You ARE ALLOWED to use internet references, your notes, textbooks, and other resources such as physics books, differential equations books, integral tables, the TI-89, or mathematical software like Mathematica, Matlab, Maple, and similar. All normal University Honor Code rules apply, except as noted above (e.g. I am allowing use of books, notes, and internet resources). DO NOT use symbolic processing capabilities for your numerical solutions (you can, of course, use these to check results).

You must submit your completed midterm via CANVAS as a .zip file with a complete set of Python or MATLAB codes solving these question and Word, Pages, or LaTeXdocument summarizing your handwritten parts, results, and analysis. Include citations, where appropriate, to results that you use for your solutions (e.g. "Hoffman eqn. 3.70", "Wolfram website (link)", "Maple software" and so on) and make sure that your work is completely described in your written solution document (i.e. it adequately developed and explained). Please start a new page for each problem.

Note that the problems are organized in order of *increasing* difficulty. You may complete any coding tasks using either Python or MATLAB.

You must sign (below) and attach this page to the front of your solution when you submit your solutions for this exam. Electronic signatures are acceptable and encouraged. If you are typing up your solution in LaTeX, please note that the source code for this document is included in the course repository.

I, _____ *, confirm that I read the instruction above and did not discuss the solution to these problems with anyone else.*

1. (33 pts.) **This problem requires both written and coding components.**

In class and in the homework we discussed the fourth order Runge-Kutta method:.

$$y^{n+1} = y^n + \frac{1}{6}\left(\Delta y_1 + 2\Delta y_2 + 2\Delta y_3 + \Delta y_4\right) \tag{1}$$

with appropriate definitions for the Δy_i as given in the book and repeated here for convenience:

$$\Delta y_1 = \Delta t f(t^n, y^n) \tag{2}$$

$$\Delta y_2 = \Delta t f\left(t^n + \frac{\Delta t}{2}, y^n + \frac{\Delta y_1}{2}\right) \tag{3}$$

$$\Delta y_3 = \Delta t f\left(t^n + \frac{\Delta t}{2}, y^n + \frac{\Delta y_2}{2}\right) \tag{4}$$

$$\Delta y_4 = \Delta t f\left(t^n + \Delta t, y^n + \Delta y_3\right) \tag{5}$$

This problem explores the *stability* of RK4 via several different approaches using the test problem:

$$\frac{dy}{dt} = f(t, y) = -\alpha y \tag{6}$$

where α is a positive and real-valued constant.

(a) Combine the four update steps for RK4 into a single formula an use it to derive a condition describing the stability of RK4. Your condition should be of the form:

$$|G(\alpha, \Delta t)| < 1 \tag{7}$$

where G is a polynomial in the quantity Δt.

(b) Plot your gain factor G vs. $\alpha\Delta t$ and mark or otherwise identify regions of stability and instability in your plot.

(c) Note that the condition in Equation 7 can be expressed as two conditions (on account of the absolute value):

$$-1 < G(\alpha, \Delta t) < 1 \tag{8}$$

The marginal stability limits (the points where we transition from stable to unstable), then, are given by:

$$G(\alpha, \Delta t) - 1 = 0 \tag{9}$$

and

$$G(\alpha, \Delta t) + 1 = 0 \tag{10}$$

which forms a pair of root finding problems. Plot each of these marginal stability conditions and evaluate all real-valued roots using an appropriate numerical method.

(d) What is the largest time step for which RK4 will give stable results for the test ODE of Equation 6? How does that compare to RK2 stability (see class notes and RK2 scripts written in MATLAB or Python).

(e) Numerically solve the given ODE with RK4 using time steps slightly above and below your derived stability criteria and show plots for each simulation that demonstrate that it behaves as your analysis predicts for these two choices of time step.

2. (34 pts.) **This problem requires both written and coding components.**

Taylor series expansions were our preferred method for deriving finite difference approximations to derivatives. In class we derived these by repeatedly generating expansions until we were able to solve for the desired derivative at the desired level of accuracy. There is, however, a more "organized" process for deriving derivatives of arbitrary order - this problem explores one such process.

(a) First and second derivative approximations were generated in class via Taylor expansions (see notes). Use Taylor series to generate a *third* derivative (d^3f/dx^3) finite difference formula approximation for a function $f(x)$ specified on some grid: $f(x_i) = \{f_i\}$ having step size Δx. There are several such approximations - generate specifically one involving the function values $f_{i+2}, f_{i+1}, f_i, f_{i-1}$.

(b) Write a script to test your third derivative formula on the function:

$$f(x) = \cos x \tag{11}$$

and plot your result alongside the analytical third derivative of this function (which you compute by hand). Ignore boundary adjacent points (i.e. only compute the third derivative on "interior" grid points). Use a 100 point grid covering the region $0 \leq x \leq 2\pi$.

(c) Generally an Nth derivative requires N Taylor series expansions in order to solve for the desired derivatives in terms of the gridded function values $\{f_i\}$. For example, the fourth derivative d^4f/dx^4 can be derived from the four Taylor series for $f_{i+2}, f_{i+1}, f_{i-1}, f_{i-2}$. Generate Taylor series for these quantities in terms of the derivatives at the ith grid point (e.g. $f'(x_i)$ and so on).

(d) Use your Taylor series equations to formulate a matrix system of equations for the unknowns f_i', f_i'', f_i''', and $f_i^{(4)}$ (viz. the derivatives up to fourth order at the ith grid point), express your system in the form:

$$\begin{bmatrix} f_{i-2} \\ f_{i-1} \\ f_{i+1} \\ f_{i+2} \end{bmatrix} = +f_i \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix} \begin{bmatrix} f_i'\Delta x \\ f_i''\Delta x^2 \\ f_i'''\Delta x^3 \\ f_i^{(4)}\Delta x^4 \end{bmatrix} + \mathcal{O}(\Delta x^5) \tag{12}$$

Where the M_{jk} entries are obtained from the Taylor series. In compact matrix form this can be represented as:

$$\underline{\Delta f} = \underline{\underline{M}}\ \underline{f'} \tag{13}$$

where:

$$\underline{\Delta f} = \begin{bmatrix} f_{i-2} - f_i \\ f_{i-1} - f_i \\ f_{i+1} - f_i \\ f_{i+2} - f_i \end{bmatrix}, \qquad \underline{f'} = \begin{bmatrix} f_i'\Delta x \\ f_i''\Delta x^2 \\ f_i'''\Delta x^3 \\ f_i^{(4)}\Delta x^4 \end{bmatrix} \tag{14}$$

(e) Write a MATLAB or Python script to numerically invert this system using either Gauss-Jordan elimination or LU factorization to compute the $\underline{\underline{M}}^{-1}$ so that you have a system describing the derivatives as a function of the function data (differences) at various grid points:

$$\underline{f'} = \underline{\underline{M}}^{-1}\ \underline{\Delta f} \tag{15}$$

Once this is solved for $\underline{f'}$, one may solve for the desired derivative by hand as needed by dividing through by Δx^4 and combining f_i terms. Derive formula for the fourth derivative using your numerically computed values for $\underline{\underline{M}}^{-1}$.

(f) Write a function to compute the fourth derivative using the formula derived in part e and use it to differentiate the test function (Equation 11), over interior grid points of your domain. Plot the result alongside the analytical fourth derivative that you compute by hand.

(g) Extend your code from part f so that it can (in principle) be used to find formulas for derivatives of arbitrary order. To improve accuracy use a centered stencil, e.g. for an nth order derivative use function values $\left\{f_{i-\lceil n/2\rceil} \dots f_i \dots f_{i+\lceil n/2\rceil}\right\}$ from grid point indices $i-\lceil n/2\rceil \dots i \dots i+\lceil n/2\rceil$. Use your code to derive and develope formulas for the fifth and sixth derivatives and write these in your solution (you do not need to implement these derivatives, just use your program to derive their formulae).

3. (33 pts.) **This problem requires both written and coding components.**

Parabolic partial differential equations of the form:

$$\frac{\partial f}{\partial t} - \lambda \frac{\partial^2 f}{\partial x^2} = 0, \tag{16}$$

are often difficult to deal with due to the highly restrictive stability condition applying to forward difference (in time) methods like a forward Euler approach (note the right-hand side is evaluated at the n time level):

$$\frac{f^{n+1} - f^n}{\Delta t} = \lambda_i \left[\frac{\partial^2 f}{\partial x^2}\right]_i^n. \tag{17}$$

In practice methods based on forward differencing in time are often unstable for reasonable time steps. Irrespective of the time differencing used, the spatial derivative can be any reasonable finite difference approximation; for this problem we use a second order accurate centered difference (here specified at an arbitrary mth time level):

$$\left[\frac{\partial^2 f}{\partial x^2}\right]_i^m \approx \frac{f_{i+1}^m - 2f_i^m + f_{i-1}^m}{\Delta x^2} \tag{18}$$

Because of overly restrictive stability criteria for forward in time differences, backward time difference formulas (BDFs) are more commonly used for parabolic equations. This problem explores several commonly-used BDFs that will be applied to solve Equation 16.

(a) The simplest BDF is just a backward Euler method:

$$\frac{f_i^{n+1} - f_i^n}{\Delta t} = \lambda_i \left[\frac{\partial^2 f}{\partial x^2}\right]_i^{n+1}, \tag{19}$$

where now the right-hand side is evaluated at the $n+1$ time level (contrast with Equation 17). Such approaches are *unconditionally stable* for the linear test problem we are using. Because the right-hand side is evaluated at the $n+1$ time you will need to solve a tridiagonal system of equations *at each time step*. Develop, starting from equation 19 the system of equations corresponding to the backward Euler (in time) method.

(b) Write a MATLAB or Python script that solves Equation 16 using a backward Euler in time method with a centered in space derivatives (Equation 18). You may use built-in MATLAB or Python utilities for the matrix solution or you can leverage your tridiagonal solver from the midterm or the Gaussian elimination tools from the repositories. Note that both repositories have an example of the Crank-Nicolson method (in MATLAB or Python) which is quite similar to backward Euler and may serve as a useful template. Plot your results in a manner similar to what is done for the example script provided. Use the parameters:

$$\lambda = 2 \tag{20}$$

$$\Delta x = \frac{1}{64} \tag{21}$$

$$\Delta t = 5\frac{\Delta x^2}{2\lambda} \tag{22}$$

$$0 \leq x \leq 1 \tag{23}$$

$$0 \leq t \leq 1024 \frac{1}{\lambda \frac{2\pi}{(2\Delta x)^2}} \tag{24}$$

(c) The backward Euler method is only first order accurate in time. A second order in time BDF approach can be developed using a second order accurate backward difference for the time derivative:

$$\left[\frac{\partial f}{\partial t}\right]_i \approx \frac{3f_i^{n+1} - 4f_i^n + y_i^{n-1}}{2\Delta t} = \lambda_i \left[\frac{\partial^2 f}{\partial x^2}\right]_i^{n+1} \tag{25}$$

Develop a numerical solution to Equation 16 based on this approach. Note that you will need to store data for f_i at two previous time levels (n and $n-1$) as opposed to the backward Euler approach which only requires prior data from the n time level. Develop and write down your system of equations that would need to be solved at each time step.

(d) Make a version of your parabolic solver from part b that implements the second order BDF derived in part c and run it on the same test problem. Plot your results and compare them against the backward Euler approach and analytical solution.