EP 501 FA 2020 midterm examination

October 16, 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).

Include citations, where appropriate, to results that you use for your solutions (e.g. "Jackson eqn. 3.70", "Wolfram website", "Maple software" and so on) and make sure that your work is completely described by your solution (i.e. it adequately developed and explained). Please start a new page for each problem. You must submit your completed midterm via CANVAS as a .zip file with Python or MATLAB codes and Word, Pages, or LaTeXtext.

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. **This problem requires both written (or typeset) and coding components.**

   In class we compared the efficiency of Gaussian elimination vs. Jacobi iteration for linear systems of various size in the `benchmark` scripts (see `./linear_algebra/benchmark.(py,m)` in the course code repository for the language of your choice). In this problem we will explore a special method for solving tridiagonal systems that is even more efficient than these two options - the Thomas algorithm.

   (a) Write a MATLAB or Python function `tridiag()` that solves a tridiagonal system of equations using the Thomas algorithm. Verify your solution by applying it to the iterative test problem for HW2 in the EP501 Assignments repository in `./HW2/iterative_testproblem.mat` file (this problem is actually tridiagonal). If you are using Python, see the `./python_basics/load_matlab_file.py` script in the EP501 Python repository for an example of how to import MATLAB data into Python.

   (b) Produce a new version of the `benchmark.(py,m)` script that runs timed tests for all three of: Gaussian elimination, Jacobi iteration, and your new `tridiag` solver. Use the same tridiagonal problem for testing each of the solvers (as in the existing `benchmark.(py,m)` script). Store the solve times for each method and each system size in separate arrays and plot times vs. number of unknowns for each method on a single graph.

   (c) For what system sizes does Gaussian elimination perform better than Jacobi iteration (on your computer)?

   (d) Explain why the Thomas algorithm is more efficient for large numbers of unknowns. Try to be as quantitative as possible by discussing the number of operations requires to execute each algorithm/iteration. The textbook may be of use.

2. **This problem requires a substantial handwritten component which should be scanned or typeset and submitted with your midterm (the problem also requires coding).**

   In plasma physics we encounter the following homogeneous system of equations describing waves in a magnetohydrodynamic (MHD) plasma:

$$\begin{bmatrix} v^2 - C_s^2 \sin^2\theta - C_A^2 & 0 & -C_s^2 \sin\theta\cos\theta \\ 0 & v^2 - C_A^2\cos^2\theta & 0 \\ -C_s^2\sin\theta\cos\theta & 0 & v^2 - C_s^2\cos^2\theta \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = 0 \tag{1}$$

   represented compactly as:

$$\underline{\underline{A}}\;\underline{u} = 0 \tag{2}$$

   In this system v is the wave velocity, θ is the angle of propagation (with respect to the local magnetic field), $u_{x,y,z}$ are the components of the plasma drift. The remaining constants are the adiabatic sound speed C_s and the Alfvén speed C_A:

$$C_s \equiv \sqrt{\frac{\gamma p}{\rho}}; \qquad C_A \equiv \sqrt{\frac{B^2}{\mu_0 \rho}} \tag{3}$$

   where γ is the adiabatic index, p is the plasma thermal pressure, ρ is the plasma mass density, and B is the magnitude of the local magnetic field.

   (a) Under what condition does this homogeneous system of equations have a nontrivial solution (viz. $\underline{u} \neq 0$)?

   (b) Derive a polynomial for the unknown wave speed v using the condition you found in part a of this problem.

   (c) How many roots (known as wave modes) does your characteristic polynomial have?

(d) Two of the roots for this system are $v = \pm C_A \cos\theta$. Plug one of these roots back into the matrix equation and use it to determine which component(s) of drift, i.e. $u_{x,y,z}$, can be nonzero for this wave mode (the shear Alfvén mode).

(e) Given typical parameters for the solar wind plasma: $\gamma = 5/3$, $\rho = 1.67 \times 10^{-21}$ [kg/m^3], $p = 1.38 \times 10^{-11}$ [Pa], and $B = 10^{-9}$ [T], compute values for the sound and Alfvén speeds in the plasma.

(f) Use an *exact* Newton method (i.e. use analytically computed derivatives) to find numerical values for all roots using $\theta = \pi/4$ for the angle of propagation. Make sure you select a sensible convergence criteria given the coefficients for this problem and treat all parameters except for the unknown roots for v to be constant for purposes of developing derivatives needed to implement Newton's method.

3. **You will need to write a Python or MATLAB script to solve this question. Be sure to output all quantities requested by this problem and included written/typed responses explaining any formulae you have derived and used in your code.**

This problem concerns polynomial division and its use to develop root finding methods for high-order polynomials.

(a) Write a Python or MATLAB function that *analytically* solves a quadratic equation of the form:

$$ax^2 + bx + c = 0 \tag{4}$$

given the column vector of coefficients $[a, b, c]^\mathsf{T}$. Test your code on the quadratic:

$$2x^2 - 6x + 4 = 0, \tag{5}$$

and compare against roots that you find by hand via factorization or quadratic formula. Show your work.

(b) Write a polynomial division algorithm (the classic approach is discusssed on pgs. 194-195 of the textbook) capable of dividing a given polynomial $P_n(x)$ (of order n and defined by a set of coefficients) by a given divisor $(x - N)$. I.e. find $Q_{n-1}(x)$ such that:

$$P_n(x) = (x - N)Q_{n-1}(x) + R \tag{6}$$

Q_{n-1} is the polynomial left once you divide $(x - N)$ out of P_n. Test your code by using it to divide out a factor of $(x - 5)$ from the polynomial:

$$x^5 - 15x^4 + 85x^3 - 225x^2 + 274x - 120 = 0 \tag{7}$$

Note: for the rest of this problem the remainder R ca be set to zero since we will be dividing out known roots.

(c) Use an *approximate* Newton's method (using an approximate derivative) to find *one* root of Eqn. 7

(d) Use your synthetic division algorithm to factor the root found in part c out of Eqn. 7 to produce a lower-order polynomial (i.e. $Q_{n-1}(x)$). The result will be a set of coefficients for a fourth order polynomial, denoted Q_4.

(d) Write a program to repeat the "deflation" process described above (parts c and d) to find all of roots of Eqn. 7. This can be done by taking Q_{n-1} (which has the first root you've found divided out) and using approximate Newton's method to find one of its roots. I.e. find a root N for Q_{n-1} and then use it to findQ_{n-2} such that: $Q_{n-1} = (x - N)Q_{n-2}$. Repeat this procedure until you have divided out three roots such that you are left with a second order polynomial Q_2. Once you have Q_2 use your second degree polynomial solver from part a to find the final two roots (this procedure is referred to as polynomial deflation). You can code your script to work specifically for this fifth order polynomial example - it does not have to be general enough to work with a polynomial of arbitrary degree (this would require a bit more programming).

Hint/Suggestion: If you are using MATLAB you will likely find the function ***`polyval()`*** *to be useful in your Newton method since you can input coefficients for a polynomial of arbitrary degree. This prevents the need to hardcode a separate objective function for each Newton iteration needed. You can simply use* polyval *with your new set of coefficient each time you divide out a factor. If using Python the analogous function is* ***`numpy.poly1d()`****. It is probably easiest to do this problem with the approximate Newton's method since you do not need to manually recompute derivatives once you divide out a factor and produce a new polynomial.*