Syllabus: Math 5620 Fundamentals of Computational Mathematics Spring 2020

Instructor: Joe Koebbe, Associate Professor
Office: ANSC 209
Office Hours: TTh 8:00am-9:00am, 10:00am-11:45am, 1:30pm-2:00pm
Phone: 1-435-797-2825
email: joe.koebbe@usu.edu
webpage: http://www.math.usu.edu/~koebbe

If you cannot make office hours, you can also set up an appointment. This can be done before or after class meetings or by email.

Textbook Information:

As of Spring Semester 2020 the following textbook is being used in Math 5620. The information for the textbook is:

Title: Finite Difference Methods for Ordinary and Partial Differential Equations
Author: Randall J. Leveque
Edition: First
Publisher: SIAM

USU Course Catalog Description for this Course:

MATH 5620 - Numerical Algorithms for Approximate Solutions of DE 2 credits Students solve initial value problems (IVP) and boundary value problems (BVP) in one dimension using standard methods. Topics include implicit and explicit methods, local and global error, stability, consistency and convergence, predictor-corrector methods and Runge-Kutta schemes, multi-step methods, and finite-difference methods for BVP. Prerequisite/Restriction: MATH 4610, MATH 2250, or MATH 2280 with a C- or better This listing includes updates which are effective beginning Fall 2019.

General Comments/Polocies on the Course:

This course covers the development and implementation of algorithms for the approximate solution of ordinary and partial differential equations. The course treats Initial Value Problems (IVPs) and Boundary Value Problems (BVPs) replacing derivatives of the unknown function with finite difference approximations. The course will cover the approximate solution of elliptic, parabolic, and hyperbolic differential equations. The finite difference schemes created will be analyzed for accuracy, stability and convergence of the approximations. Methods for handling boundary and initial conditions will be presented and implemented.

The course will involve significant computer programming assignments. Students must know how to write computer code in a high level programming language (e.g; Python, Fortran, C, C++, Java, and others) or work in a computational platform like Matlab or Maple. If you choose to use Matlab or Maple, you will be required to implement your own versions of the algorithms discussed in class. You can use the intrinsic routines to verify your code. For example, in Matlab there are simple expressions that will prompt the software within Matlab to compute an approximate solution of a linear system of equations. The nuts and bolts are buried deep in the software that Matlab provides. The point of this course is for each student to be able to fully implement algorithms. Using the software provided in Matlab does not allow this. There is another reason why students need to learn all the nuts and bolts that software like Matlab, Mathematica, and Maple cloak. When you are working in a job and need to ship software to your clients, Matlab, Mathematica, and Maple require expensive licensing for their part of the software. These packages are too expensive in most cases. So, you will need to write your own code.

Students will be required to obtain a Github account and post homework soltuions on this account. Students will create a private repository and invite your instructor to be a "collaborator" so that the assignments can be turned in electronically. A tutorial on how to do this will be presented in class.

Parallel algorithms for solving problems have been around for a long time. At first, these algorithms were interesting, but maintaining parallel codes can be difficult. In the past decade, the limitations of the physics associated with computing have put limitations on the speed and capacity of a single CPU. Computer companies have started to put more processors into their computers with the idea of improving performance. GPUs have been included to off load the work needed for displaying graphics and the like onto a specialized parallel processing card. Ways to take advantage of multiple cores and CPUs and the GPU onboard your computer have been developed. Students in this course will gain some knowledge of how to use these tools. In particular, concepts from the directive based languages, OpenMP and OpenACC, will be presented to show students the benefits of multicore processing the GPU processing.

Grading:

Your grade in the course will be determined by the following:

1. Homework will account for a 40% of a student's grade. Homework must be turned in on time. **Late homework will not be accepted under any circumstances.** If you must be gone, any homework must be turned in before the due date. The only exception to this rule is for a family emergency. Also, codes must be carefully documented. An example of a well documented code will be discussed in class and a version will be posted on line.

2. Two midterms will be given. Each will cover about one half of the content of the course. Each of these midterms will account for 20% or 40% of the grade earned in the course.

3. A portfolio of routines will be required of each student. The portfolio of routines must be written in the form of a software manual. The exact form of the portfolio will be discussed in class. Based on past experience the formatting will be strict in the sense that the entries in the manual will be limited to two pages. The actual format will also be restricted. Software manuals that do not meet the formatting requirements will not be graded. Part of the reason for putting strict requirements for formatting is due to the idea that in a job, you will need to follow the company formatting to have your algorithms included in bigger projects.

If students have questions about assignments and midterms, please contact me. One of the assignments will be to meet with me at least once every two weeks to discuss progress in the course. This means that 2 times each month during the semester you will need to show up at office hours or make an appointment to see me. You will need to do this 8 times during the semester.
