AMSC/CMSC 663-664 Advanced Scientific Computing
Fall 2018

Description
AMSC/CMSC 663-664 is a two-semester project course in which each student will identify and carry out a scientific computing project with focus on:
- Understanding of scientific computing algorithms related to the project
- Code Development, including
    - Modularity, portability, memory management
    - Post-processing, restarting, and writing to databases
    - Interactivity
    - Scientific visualization
    - Documentation and version management tools
    - Debugging and profiling tools
    - Validation
- Verification and unit testing using test problems
- Time permitting, additional considerations may be given, for example implementation of parallel algorithms: OpenMP, MPI, GPU programming; Masking communication costs, load balancing, granularity; parallel numerical linear algebra.

Logistics
Lectures: Tu/Th 2:00-3:15pm.
Location: MATH B0425
Instructors:
Tom Goldstein: AVW 3141 (pi)
Jacob Bedrossian: MATH 4413 and CSIC 4129

Prerequisites
You *must* have taken both AMSC/CMSC 660 and 661, or equivalent.

Course requirements
The first semester will have the following deliverables: proposal document, proposal presentation, midterm presentation, final presentation, final report. You will also deliver code and documentation for your project.

Your faculty advisor is required to participate in a meeting with 663 supervisors, and must attend your final presentation.

Grading
This course does NOT have an “A for effort” grading policy. Projects will be graded on the quality of code, documentation, presentation, and writing. Projects that do not meet high standards of quality will not receive an “A”, and in some circumstances will not receive a passing grade.

I**mportant dates and grading events**
Aug 30-Sept 11: Personal presentations
Sept 24: Proposal report and advisor meetings must be completed:
Sept 25 - Oct 11: Proposal presentations
Oct 23-25: Code reviews
Oct 30 - Nov 8: Midterm update presentations
Nov 13 - 15: Code review
Nov 27th - done: final presentations
Dec 10: Final report

Requirements for deliverables

Personal presentation (15 mins):

- Where are you from?
- What are you interested in professionally?
- What are you interested in personally?
- What do you hope to get out of this class?

Proposal document and presentation should include:

- Background on the problem: why is it important, what is the state of the art (with citations)?
- Project Goals: what are you hoping to achieve?
- Approach: How will you achieve these goals? What components will need to be implemented to get there?
- Describe specific algorithms and how they will be implemented
- Describe hardware/software platform you target. What programming language?
- How will it be *documented* and *distributed*? Github? User guide?
- Validation methods: complete suite of unit tests!
- Deliverables: what specific components/code/data/visualizations will you generate?
- Milestones, and a rough timeline of when you expect to accomplish them?

The oral presentation should last no more than 30 minutes including questions and discussions. A sample presentation document is available. You will be evaluated on the quality of your presentation skills, and clarity of your slides.

The proposal document should be at most 5 pages, and fewer is acceptable.

Midterm update (15 mins):

- Brief overview of project
- Brief overview of approach, goals, milestones
- Update on current status and accomplishments
- Figures or code snippets or demos to show what your code can do
- Explanation and description of any changes/updates to approach

Final Presentation and report (30 mins):

- Build on the proposal and midterm presentations.
- Overview of proposal content
- Detailed description of what has been accomplished and changes to the approach
- Description of what did not get accomplished and why
- Overview of deliverables, such as code, data, code documentation and distribution
- Description of plans for the next semester

Your code must be...

- Well documented, with a user guide (if appropriate)
- Well commented. This includes having descriptive headers and usage information at the top of every file, and having important description information within the code body. Code should also be clearly written, and "self-documenting" when appropriate.
- Well organized, clean, loosely coupled, and extensible
- Portable to different systems
- Thoroughly unit tested and validated
- Distributed using online tools, such as Gitlab, Github, etc...

Academic integrity

Be aware of the UMD code of academic integrity, found here http://www.president.umd.edu/policies/iii100a.html. All sources must be properly and clearly cited. Text or code can never be copied verbatim from the sources, this is plagiarism and a serious offense. You can, of course, import and use third-party libraries in your code (provided you give full and due credit and discuss the third party in your reports and presentations).