ASTR 4470/5470: Computational Astronomy/Astrophysics Final Projects

Goals

The final project is intended to give you some experience writing a larger, interconnected code. We mostly have focused on writing a series of small functions for homework problems and some of the research we do is like this – short analysis and plotting scripts to analyze the output of more complex codes or observational data. But, it is also often the case that one needs to write larger codes to solve a problem, to perform more sophisticated data analysis, or to synthesize output from different software tools. In other cases, one needs to add functionality to large, sophisticated code bases. The goal of the final project is to give everyone some experience with coding on a somewhat larger scale and with less structure than a typical homework set provides. It will also give us some experience with designing the architecture for the code: How will the code interface with the user? What form will the outputs take? What analysis and plotting functionality will be provided? How will the code be organized among different files?

Another key point to understand is that building a code is more than just writing a code that works. One needs to make it easy to use and document it so that others (and your future self) can use it. Hence, documenting and structuring the code is a requirement of the project.

What is Required

The final project should represent a substantial effort – roughly equivalent to 3.5 homework assingments! All final projects must fulfill the following requirements:

- A written project proposal that outlines an initial plan and API for the code. This includes doing some research on what the components of the code will be and how they work together
- The code must have a convenient user interface – either via an input file that is read in or using command line interface
- The code must produce multiple well defined output files, including plots. These can be output text or binary files along with code used to generate plots from these files or the output can be the plotting files (as .png, .pdf, etc.) themselves.
- The code must be well organized and distributed over multiple files. For example, in C these can be a number of header files and .c files implementing different sets of related functions which are compiled with a Makefile provided by you. For Python, this could be a module or set of module(s) supplemented by scripts and/or notebooks.
- The code must have a dedicated GitHub repository **used for development**
- The code must be documented via (**undergrads and grads**) a README file with basic instructions for running (and building, if needed) the code and (**grad students only**) a GitHub wiki (implemented in markdown) with more detailed information on inputs, outputs, and usage
- The project must have a clearly scientific goal, even if the application is relatively simple and already well understood.
- You must devise and implement at least two (**three for grad students**) non-trivial tests of the code. These can be comparisons to analytic solutions or results from other (reliable) codes.
- The project must be a novel effort. I do not want you to recycle a code you have already written! If, however, you are interested in significant effort to extend an existing code (written by yourself or others), you are welcome to discuss the possibility with the instructor. Your update would still need to be subject to all of the above requirements.

What is optional

There are also several optional choices for the project that nevertheless deserve consideration:

- You are welcome to write the code in either Python or C or a mix of both. For example, you could write a C code that produces output files that are analyzed or plotted with Python scripts.
- Other languages are potential options as well, but this **needs to be approved by the instructor**.
- The code can be focused either on data analysis, or theoretical computations (e.g. numerical simulation).

Choosing a Project

Students are welcome to submit projects subject to the above requirements. So if there is a project that fits in well with your current research, that would be wonderful. If you are not currently doing any research projects that would be suitable or are otherwise uncertain, I have several suggestions for projects that can incorporate algorithms learned in class that I believe to be achievable on the time frame of the project. These include:

- Writing your own hydro code using finite volume methods.
- Writing your own MCMC-based data analysis tookit
- Writing your own N-body integrator
- Writing your own Monte Carlo radiation transfer code
- Writing a Fourier analysis code to look for periodic behavior in light curves

Recommendations

These are not requirements, but I would like to make several recommendations to avoid stress and frustration:

- You are welcome to write the code in languages other than Python but students are encouraged to use Python unless you are very comfortable with other codes
- If you are unsure about a good idea for a project that meets the above requirements, use one of the suggestions that I provide above
- Although I applaud ambition, start with a clear, focused project with achievable goals. If development proceeds faster than expected, then think about expanding the capabilities of your code.
- We will have multiple weeks to complete the project – make sure you are making continuous progress. We will be using Thursday's class for check-ins. Do not put things off until the last week!

Rubric

The final project will be graded according to the following rubric. Note that the numbers not in parentheses represent the weighting for graduate students and the numbers in parentheses represent the weighting for undergraduates. In other words, graduate student projects places more weight on testing and documentation whereas the undergraduate course emphasizes getting the code to work

Percentage	Requirement
25% (25%)	**Writing a clear, reasonable, and focused project proposal.** The proposal should address: the goal of the research project, including what scientific calculations and/or analysis the code will perform; the primary methods/algorithms to be employed in the code; how the code will be broadly structured and implemented; inputs and outputs; at least two possible tests that will be performed. This should be approximately two-three pages of text and equations (if needed to describe the algorithm), with additional pages as needed for figures and references.

35% (45%) **Generating a working code.** The code should run and perform the core functions outlined in the proposal. Does the code have an effective I/O interface that allows one to change input parameters and data? Are the outputs correct and reliable?

15% (10%) **Quality of documentation.** If I follow the instructions, will the code run as documented. Does the documentation cover modifying the input and explain the outputs. If there are tools for examining or further analyzing the outputs, is there usage explained clearly?

15% (10%) **Implementation of the code tests.** Were the three required code tests implemented? Can the user run them? Do they test the code effectively? Does at least one of them test a specific subset of the code?

10% (10%) **Usability, extensibility, and execution.** Is the code easy to use? Is the code compartmentalized and organized well? Is it extensible – written in a way that new methods can be implemented or functionality can be extended. Did the student use the GitHub repo effectively (frequent commits, etc.)? Is the code sufficiently commented? Does the product indicated the student put a substantial effort into the project.