Some considerations regarding programming language choice

If you already have very fixed ideas on what language to use in the course, then be careful not to over-analyze the following. However, an awareness of some of the issues is almost certainly healthy. In fact, I am now convinced that for real versatility, one needs to be able to use several languages where each one has strengths for particular applications.

- *Familiarity.* Remember the goal of the course is for you to solve physical problems using computational methods rather than learn a new programming language from scratch. If you have only modest programming experience, python may be the best choice, and I will likely lean heavily on python examples.
- *Operating System.* Most scientific computing is done using a Unix like environment. You should consider which *OS* you will primarily be using (Linux, Mac, Windows) during the day, at your residence, and potentially in the computer lab.
- *Portability* is a major criterion. Will a program you write in a particular programming language be of any lasting value to you or your colleagues now and in the future? Does it need a license?
- *Compatibility* with other software packages like graphics packages, histogramming packages etc.
- *Main-stream.* Scientific programs are ways to instantiate problem solutions in language that hopefully communicates reasonably with other scientists.
- *Some possibilities*.
I mainly use C++, python, and Fortran. I tend to automate being able to repeat computational experiments with shell scripts (in bash). You also may want to continue to develop proficiency in languages directly related to your research group's methodologies. Also note that many languages have evolved substantially from their initial specifications in ways that make them much easier to use. (I just ordered a book on C++20…).
- *Compiler.* Most languages need compilers which make them platform dependent and may cost a significant amount of money. Different compilers can give different results. This is not usually a good thing, but can be a good cross-check. A good compiler is often your best friend in writing clear and correct code.
- *Scientific Library Support.* Efficient development of computational methods for research problems relies on straightforward access to tried and tested code in the form of libraries which are the de facto standard in various fields. These build on decades worth of development. You may be at a serious disadvantage if you are unable to call subroutines from for example the LAPACK library that deals with massive linear algebra problems. There are ways to do this from C and C++. Boost, Eigen and the Gnu Scientific Library (GSL) are good starting points for C/C++, and numpy etc for python.
- *Interoperability.* To some extent there are some relatively painless ways to inter-operate with different languages eg. f2c, f2py etc. with more modern compilers.
- *Procedural or Object-Oriented*? This is a difficult one. I feel a responsibility to encourage you to use modern programming languages. Badly written code in any language is to be discouraged. However for some simple tasks a focus on OOP is overkill.
- *Good Editor with language support*. (This doesn't mean Notepad!, something like emacs/gedit or vi/vim for the brave).

- *Debugger*.
- *Speed/Memory Usage.* This is very unlikely to be a major concern in this class, but may be for particular domain specific problems.
- *Clarity and Conciseness*.
- *Availability of Integrated Development Environment (IDE).* Will show some examples in class of what I mean.
- *Support for Numerical Programming.* There are a number of facets of different programming languages, and the libraries associated with them, which can often enable simple solutions to particular scientific problems. Is there sufficient support for complex variables ? Are reliable random number generators available ? Can you just multiply arrays ?
- *Available Documentation and Programming Language Learning Resources for Scientists.* Note that books which teach business type skills of how to modify phone directories or calculate the pay-roll are not talking to you as a scientist.
- *Scalability*
  How easy is it to have a program with 100 lines of code, 1000 lines of code, 10,000 lines of code. Personally I find it very annoying that the python indentation syntax for loops and conditional statements makes it non-trivial to make relatively modest changes to a chunk of code (eg. make it conditional). So I usually avoid python for bigger projects. Likewise Fortran often does not “grow” well.