2G03 Project 2020

When to start thinking about it: October 1st (see project1 lecture)
Consult with TA/Lecturer about idea: ASAP
When to start writing code: Anytime, as soon as you get a proposal approved
Written Proposal: Oct 22 or sooner – we give you detailed feedback as they come in
In class program demo: mid November
Final Write-up Due: last week of class (Dec)

Overview

Each student will complete an individual project: designing and writing a program that demonstrates the numerical solution of a scientific problem. The individual projects will be developed over the course of the term and consultation and advice will be readily available. The project consists of several parts:

1) Project proposal (short description)

Firstly the student will select a (scientifically) interesting system or situation that can be solved with computation and then write a proposal that outlines the problem, how they might solve it and how they might test their computational version of the system to see if it behaves correctly (i.e. like the real system).

It is expected that most projects will identify a set of Ordinary Differential Equations (ODEs) that can model the chosen system. Projects that do not rely on ODEs require special permission and you must approach the instructor prior to even proposing such a project. This is most other choices are too hard (e.g. PDE based projects) or too easy (e.g. linear algebra). In some areas that don't use ODEs, e.g. neural nets, there are reasonable approaches and hard ones. We will provide guidance on workable choices.

You are *strongly* encouraged to talk with lecturers in other courses to identify potential projects in research areas related to those courses. Part of the goal of 2G03 is to prepare you for computer based research work and this contact could translate into a research opportunity or a summer job.

The proposal only needs to be a page and should contain equations. A typical way to present the proposal is to describe a system you wish to model, write down the equations (or otherwise describe the system mathematically), discuss how you'll solve the mathematical model on a computer and what you will look for when you test it.

Note that it is not acceptable to use an external library that provides a complete solution to the problem in response to a very simple main program written by you or to write program that simply plots analytical solutions. Your program must actually solve the equations.

You are welcome to discuss your proposal with other students. However, the normal rules about plagiarism apply. In particular, students working in similar areas should not work on the same project. You are expected to ultimately present a project that is different from that of other students. The preliminary parts may be similar (such as the first demo versions and tests you do).

Proposals are worth participation marks, about 20% of the project grade. If you don't hand one in on time you lose marks.

Example project research areas

The following suggested areas are very broad. You should select a more specialized topic within one of the following areas for your project.

- Protein Folding / Molecular Dynamics (biochemistry)
- Natural Selection/Genetic Algorithms (evolutionary biology)
- Population growth/infections (epidemiology and populations)
- Decision Making/Brain models (psychology, neuroscience)
- Neural Networks (e.g. Hopfield model) (psychology, neuroscience)
- Economic Models/Stock Market (finance)
- Crystal growth (materials)
- Phase diagrams (thermochemistry)
- Solar system/Orbit integration (astronomy)
- Structure models for planets or stars (astrophysics)
- Models of condensed matter systems (physics)
- Electrostatics/Gravity (physics, astrophysics)
- Diffusion models (materials, physics)
- Search algorithms for genetic data (genomics)
- Data Mining (computer science, finance)
- Electrical Circuit Modeling (engineering/physics)
- Kinematics, Collisions, Projectile Motion (physics)

2) Demo Program

The goal of the project is to create a C++ program on phys-ugrad to demonstrate the solution of the model problem and show that the program approximates the behaviour of the real physical system in some way. You should do this in a directory called **project** in your home directory on phys-ugrad. You should generate a **Makefile** to make your runnable project in that directory.

We strongly recommend including some real-time plotting with your project so you can see it working and make figures for the write-up. The pgplot library is ideal for this – we will help you use it. See the example in **/home/2G03/ploty** for some basic plotting. See the example fake project in **/home/2G03/lorenz** for some ideas on how to use plotting in a project. Note that it is not required that your plots are interactive or clickable.

The best way is to make progress is to gradually improve your program – make sure it works first. When it does work on a basic case such as a simple test, save that version for your demo, e.g. **cp -r ~/project ~/demo** . This copies all the current files in your project to a new directory called **demo**. It should literally be possible to go to this directory, type **make** and then run the demo.

Demos are worth participation marks, about 20% of the project grade. If you don't have a runnable demo by the Nov. due date you lose marks.

3) Extended Program

Once you have a demo that reproduces standard results you should come up with a unique angle for your project and extend the program to explore that. Typical choices include exploring the parameter space to identify interesting behaviour and cases, adding a tweak to the equations to compensate for an obvious shortcoming of the model, adding in some extra physics that leads to a more complex behaviour or running bigger versions (such as versions with more objects, species, planets etc...).

Students sometimes have code problems with the extended project code so it doesn't work as planned or has a difficult to find bug. There will be class time and help to resolve these issues. If the problem is never resoled, the project can still be acceptable, especially if the demo worked. You can describe your plan in the write-up even if the code never quite did what you wanted.

4) Final Write-up

The write-up should start with a brief introduction to the area you chose and the model that your program uses from that area. You can also describe more advanced approaches (that you did not explore with code). You should include references to papers, books and/or well-regarded web-sites (e.g. scientifically sound wiki-pedia entries).

In the method section, you show the equations you solved and the kind of solver you used. In many cases it will be leap-frog or Runge-Kutta (or another method presented in lectures) and you will not need to provide very much detail. If it wasn't, give a reference to where the method came from.

In the tests section, you describe how you tested your program to be sure the model is working as it should with figures. In many cases this is the demo. For orbits this might be a simple orbit. For populations or disease models it might be reproducing a figure from the original paper or source. You should include some figures in the write-up. Ideally you clearly describe the expected answer and the origin of any discrepancies. For time evolution, you might show that the answer is the same for smaller steps forward in time over the same interval or that a conserved property does not change significantly (e.g. within round-off errors).

In the results section, you evaluate the model include any extensions you made to it. This might include changing parameters or cases or modifying the basic model a bit. For this part you need to do something different to other students. It should be working correctly at this point but may not be a good approximation of the real world. You should show figures of how the model system behaves and compare it to real world outcomes or expectations. You can be critical of the model (the original and any modifications or ideas you tried).

Finally, you should put forward conclusions that you feel are justified by the exploration you did and discuss the possible alternative approaches, future work if you had time and so on.

The final working C++ code for your project should be present in the directory **project** in your home directory. It is strongly discouraged to develop code off phys-ugrad or in any other programming language except C++. If you present a project (e.g. results and plots) without C++ code on phys-ugrad that could produce it, it may lead to problems with *academic dishonesty*. The best approach is to work direclty on phys-ugrad and get all your results by running code there. The graders will look for working code on phys-ugrad in your directory.

It is not acceptable to re-use work you or others have submitted for another course or project. You must develop a new project and write new C++ source code based on a description of the appropriate algorithm(s). Converting or using another code is considered academic dishonesty.

It *is* acceptable to use another code to test that your own code works by comparing results but you must explicitly say you did that. It is a nice way to verify. That other code could have been written by you for another course, etc...

Program Testing

For the programming part, you must demonstrate that the program works and that the results are correct. There are two parts to this: verification and validation.

Verfication indicates that the code compiles, runs and is actually doing what you say it should – i.e. no bugs. Verification typically comes first.

Validation is showing that the model, as implemented in your program, is a realistic model of a real system. If it isn't you can explore that too and critique it.

If your problem and solution describe a physical system, you must test if the numbers coming out are physically reasonable. For example, a program to integrate the orbit of the Earth should have results where the Earth's orbital period is one year. Most scientific problems have parameters. If the results for your system are supposed to vary in a particular way as you change the parameters you need to show this happens with your program. As noted below, it is acceptable to use analytical solutions, another code or other source to independently generate correct answers for comparison. A detailed reference must be given in this case. Note that your program may not be able to model the system very well. This can be ok as long as you can describe how it fails (in some cases) and have a go at explaining why.

Your write-up should include figure showing results from your program. This should include at least one figure/plot showing a test of basic correctness (verification). It should also include at least one figure (usually several) showing how the model responded to changes and showcasing the unique direction you took the project at the end. The grader is not going to run your program and try to recreate the interesting cases – you need to include plots for that!

Key steps to completing the project

1. Choose a research area or interesting problem and think up some ideas
2. **REQUIRED**: 5 minute proposal discussion with T.A. or instructor. You must tell the T.A. your chosen research area and get feedback on where to look for background material. You can also bounce ideas off the T.A.'s or the lecturer at other times.
3. **REQUIRED by Oct 22 nd**: Hand in proposal (do it earlier if you can)
4. In class feedback
5. **REQUIRED**: 10 minute feedback session with T.A. or instructor. In this session, the T.A. will help you set the programming goal for your project. Note that you are free to consult further on the program as you work on it.
6. Code up your program and test it (class time will be available for this and help to get it to work)
7. **REQUIRED by mid Nov (Date TBA)**: Demonstrate a working program in tutorial/class time. The program can be a work in progess but it must at least run
8. **FINAL HAND IN, Dec**: Hand in write-up for the program

Final Write-up and Assessment

60 % of the assessment will be a combined grade based on the quality of the final write-up, the program itself and how well you explored the model.

40 % will be a fixed participation grade for handing in a proposal and doing the in-class demo.