Slides from FYS3150/4150 Lectures, introduction to the course

Morten Hjorth-Jensen[1,2]

[1]Department of Physics, University of Oslo
[2]National Superconducting Cyclotron Laboratory, Michigan State University

Fall 2014

Overview of week 34

- Monday: First lecture: Presentation of the course, aims and content
- Monday: Second Lecture: Introduction to C++ programming and numerical precision.
- Tuesday: Numerical precision and C++ programming, continued and exercises for first week
- Numerical differentiation and loss of numerical precision (chapter 3 lecture notes)
- Computer lab: Thursday and Friday. First time: Thursday and Friday this week, Presentation of hardware and software at room FV329 first hour of every labgroup and solution of first simple exercises.

Lectures and ComputerLab

- Lectures: Monday (4.15pm-6pm) and Tuesday (12.15pm-2pm) only this and next week. Thereafter weeks 38 and 39, and weeks 43 and 44 and finally weeks 47 and 48.
- Weekly reading assignments needed to solve projects.
- First hour of each lab session used to discuss technicalities, address questions etc linked with projects.

- Detailed lecture notes, exercises, all programs presented, projects etc can be found at the homepage of the course.
- Computerlab: Thursday (9am-7pm) and Friday (9am-7pm) room FV329.
- Weekly plans and all other information are on the official webpage.
- Final written exam December 12, 9am (four hours).

Course Format

- Several computer exercises, 5 compulsory projects. Electronic reports only.
- Evaluation and grading: The last project (50
- The computer lab (room FV329)consists of 16 Linux PCs, but many prefer own laptops. C/C++ is the default programming language, but Fortran2008 and Python are also used. All source codes discussed during the lectures can be found at the webpage of the course. We recommend either C/C++, Fortran2008 or Python as languages.

ComputerLab

day	teacher
Thursday 9am-1pm	Anders, Morten L., Håvard, MHJ
Thursday 1pm-5pm	Anders, Morten L., Håvard, MHJ
Friday 9am-1pm	Anders, Morten L., Håvard, MHJ
Friday 1pm-5pm	Anders, Morten L., Håvard, MHJ

Topics covered in this course

- Numerical precision and intro to C++ programming
- Numerical derivation and integration
- Random numbers and Monte Carlo integration
- Monte Carlo methods in statistical physics
- Quantum Monte Carlo methods
- Linear algebra and eigenvalue problems

- Non-linear equations and roots of polynomials
- Ordinary differential equations
- Partial differential equations
- Parallelization of codes
- Programming av GPUs (optional)

Syllabus FYS3150

Linear algebra and eigenvalue problems, chapters 6 and 7.

- Know Gaussian elimination and LU decomposition
- How to solve linear equations
- How to obtain the inverse and the determinant of a real symmetric matrix
- Cholesky and tridiagonal matrix decomposition

Syllabus FYS3150

Linear algebra and eigenvalue problems, chapters 6 and 7.

- Householder's tridiagonalization technique and finding eigenvalues based on this
- Jacobi's method for finding eigenvalues
- Singular value decomposition
- Qubic Spline interpolation

Syllabus FYS3150

Numerical integration, standard methods and Monte Carlo methods (chapters 4 and 11).

- Trapezoidal, rectangle and Simpson's rules
- Gaussian quadrature, emphasis on Legendre polynomials, but you need to know about other polynomials as well.
- Brute force Monte Carlo integration
- Random numbers (simplest algo, ran0) and probability distribution functions, expectation values
- Improved Monte Carlo integration and importance sampling.

Syllabus FYS3150

Monte Carlo methods in physics (chapters 12, 13, and 14).

- Random walks and Markov chains and relation with diffusion equation
- Metropolis algorithm, detailed balance and ergodicity
- Simple spin systems and phase transitions
- Variational Monte Carlo
- How to construct trial wave functions for quantum systems

Syllabus FYS3150

Ordinary differential equations (chapters 8 and 9).

- Euler's method and improved Euler's method, truncation errors
- Runge Kutta methods, 2nd and 4th order, truncation errors
- How to implement a second-order differential equation, both linear and non-linear. How to make your equations dimensionless.
- Boundary value problems, shooting and matching method (chap 9).

Syllabus FYS3150

Partial differential equations, chapter 10.

- Set up diffusion, Poisson and wave equations up to 2 spatial dimensions and time
- Set up the mathematical model and algorithms for these equations, with boundary and initial conditions. Their stability conditions.
- Explicit, implicit and Crank-Nicolson schemes, and how to solve them. Remember that they result in triangular matrices.
- How to compute the Laplacian in Poisson's equation.
- How to solve the wave equation in one and two dimensions.

Overarching aims of this course

- Develop a critical approach to all steps in a project, which methods are most relevant, which natural laws and physical processes are important. Sort out initial conditions and boundary conditions etc.
- This means to teach you structured scientific computing, learn to structure a project.
- A critical understanding of central mathematical algorithms and methods from numerical analysis. In particular their limits and stability criteria.
- Always try to find good checks of your codes (like solutions on closed form)
- To enable you to develop a critical view on the mathematical model and the physics.

And, there is nothing like a code which gives correct results!!

- J. J. Barton and L. R. Nackman,*Scientific and Engineering C++*, Addison Wesley, 3rd edition 2000.
- B. Stoustrup, *The C++ programming language*, Pearson, 1997.
- H. P. Langtangen INF-VERK3830 http://heim.ifi.uio.no/~hpl/INF-VERK4830/
- D. Yang, *C++ and Object-oriented Numeric Computing for Scientists and Engineers*, Springer 2000.
- More books reviewed at http:://www.accu.org/ and http://www.comeaucomputing.com/booklist/

Other courses in Computational Science at UiO

Bachelor/Master/PhD Courses.

- INF-MAT4350 Numerical linear algebra
- MAT-INF3300/3310, PDEs and Sobolev spaces I and II
- INF-MAT3360 PDEs
- INF5620 Numerical methods for PDEs, finite element method
- FYS4411 Computational physics II (Parallelization (MPI), object orientation, quantum mechanical systems with many interacting particles), spring semester
- FYS4460 Computational physics III (Parallelization (MPI), object orientation, classical statistical physics, simulation of phase transitions, spring semester
- INF3331 Problem solving with high-level languages (Python), fall semester
- INF3380 Parallel computing for problems in the Natural Sciences (mostly PDEs), spring semester

Extremely useful tools, strongly recommended

and discussed at the lab sessions the first week.

- GIT for version control (see webpage)
- ipython notebook
- QTcreator for editing and mastering computational projects (for C++ codes, see webpage of course)
- Armadillo as a useful numerical library for C++, highly recommended
- Unit tests, see also webpage
- Devilry for handing in projects

A structured programming approach

- Before writing a single line, have the algorithm clarified and understood. It is crucial to have a logical structure of e.g., the flow and organization of data before one starts writing.
- Always try to choose the simplest algorithm. Computational speed can be improved upon later.

- Try to write a as clear program as possible. Such programs are easier to debug, and although it may take more time, in the long run it may save you time. If you collaborate with other people, it reduces spending time on debuging and trying to understand what the codes do. A clear program will also allow you to remember better what the program really does!

A structured programming approach

- The planning of the program should be from top down to bottom, trying to keep the flow as linear as possible. Avoid jumping back and forth in the program. First you need to arrange the major tasks to be achieved. Then try to break the major tasks into subtasks. These can be represented by functions or subprograms. They should accomplish limited tasks and as far as possible be independent of each other. That will allow you to use them in other programs as well.

- Try always to find some cases where an analytical solution exists or where simple test cases can be applied. If possible, devise different algorithms for solving the same problem. If you get the same answers, you may have coded things correctly or made the same error twice or more.