Computational Physics and Quantum Mechanical Systems

Morten Hjorth-Jensen[1,2]

Department of Physics, University of Oslo, Norway[1]

Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University, USA[2]

May 16-20, 2016

Topics covered in this course

This course aims at giving you an overview of important numerical methods for solving quantum mechanical problems. In particular, in this course we will focus on

- Quantum Monte Carlo methods applied to few-electron systems like quantum dots, electrons confined to move in two or three dimensions. The system we will study will contain up to 20 interacting electrons in an oscillator trap.
- Linear algebra and eigenvalue problems with applications to the same physical systems, simple two-electron systems and Hartree-Fock theory.
- Object orientation, code optimization and High-performance computing aspects

The main programming language will be c++, interspersed with Python examples in the slides.

Overarching aims of this course

While developing the mathematical tools and a numerical project we aim at giving you an insight in how to:

- 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.
- To teach you structured scientific computing and learn to structure a project using for example object orientation.
- To give you a critical understanding of central mathematical algorithms and methods from numerical analysis. In particular their limits and stability criteria.
- To show you how to use version control software to keep track of different versions of your normal project, how to verify and validate your codes and how to make your science reproducible.

Learning outcomes

We hope we can give you some useful hints and guidance on

- how to develop a thorough understanding of how computing is used to solve scientific problems
- getting to know some central algorithms used in science
- developing knowledge of high-performance computing elements: memory usage, vectorization and parallel algorithms
- understanding approximation errors and what can go wrong with algorithms
- getting experience with programming in a compiled language (Fortran, C, C++)
- getting experience with test frameworks and procedures
- being able to critically evaluate results and errors
- understanding how to increase the efficiency of numerical algorithms and pertinent software
- understanding tools to make science reproducible and to develop a sound ethical approach to scientific problems

Why is computing comptence important?

Definition of computing

With computing I will mean solving scientific problems using computers. It covers numerical, analytical as well as symbolic computing. Computing is also about developing an understanding of the scientific process by enhancing algorithmic thinking when solving problems.

Computing competence, what is it?

Computing competence has always been a central part of the science and engineering education. Traditionally, such competence meant mastering mathematical methods to solve science problems - by pen and paper.
Today you are expected to use all available tools to solve scientific problems; computers primarily, but also pen and paper.
I will use the term/word algorithms in the broad meaning: methods (for example mathematical) to solve science problems, with and without computers.

What is computing competence about?

Computing competence is about

1. derivation, verification, and implementation of algorithms
2. understanding what can go wrong with algorithms
3. overview of important, known algorithms
4. understanding how algorithms are used to solve mathematical problems
5. reproducible science and ethics
6. algorithmic thinking for gaining deeper insights about scientific problems

Continuous verse discrete

Algorithms involving pen and paper are traditionally aimed at what we often refer to as continuous models.
Application of computers calls for approximate discrete models.
Much of the development of methods for continuous models are now being replaced by methods for discrete models in science and industry, simply because much larger problem classes can be addressed with discrete models, often also by simpler and more generic methodologies. However, verification of algorithms and understanding their limitations requires much of the classical knowledge about continuous models.

Shift from classical mathematics to modern computing?

1. The impact of the computer on mathematics is tremendous: science and industry now rely on solving mathematical problems through computing.
2. Computing increases the relevance in education by solving more realistic problems earlier.
3. Computing through programming is excellent training of creativity.
4. Computing enhances the understanding of abstractions and generalization.
5. Computing decreases the need for special tricks and tedious algebra, and shifts the focus to problem definition, visualization, and "what if" discussions.

The result is a deeper understanding of mathematical modeling. Not only is computing via programming a very powerful tool, it also a great pedagogical aid. For the mathematical training, there is one major new component among the arguments above: understanding

Key principle in scientific modeling

The power of the scientific method lies in identifying a given problem as a special case of an abstract class of problems, identifying general solution methods for this class of problems, and applying a general method to the specific problem (applying means, in the case of computing, calculations by pen and paper, symbolic computing, or numerical computing by ready-made and/or self-written software). This generic view on problems and methods is particularly important for understanding how to apply available, generic software to solve a particular problem.

Computing competence is central to solving scientific problems

Definition of computing

Computing competence represents a central element in scientific problem solving, from basic education and research to essentially almost all advanced problems in modern societies. Computing competence is simply central to further progress. It enlarges the body of tools available to students and scientists beyond classical tools and allows for a more generic handling of problems. Focusing on algorithmic aspects results in deeper insights about scientific problems.

Today's project in science and industry tend to involve larger teams. Tools for reliable collaboration must therefore be mastered (e.g., version control systems, automated computer experiments for reproducibility, software and method documentation).

Modeling and computations

In this course we will try to focus on how to

- Introduce Research based teaching from day one
- Trigger further insights in math and other disciplines
- Validation and verification of scientific results, with the possibility to emphasize ethical aspects as well. Version control is central.
- Introduce good project handling practices from day one.

Symbolic calculations and numerical calculations in one code

The following simple example here illustrates many of the previous points. Python offers an extremely versatile programming environment, allowing for the inclusion of analytical studies in a numerical program. Here we show an example code with the **trapezoidal rule** using **SymPy** to evaluate an integral and compute the absolute error with respect to the numerically evaluated one of the integral $4\int_0^1 dx/(1+x^2) = \pi$:

```
from math import *
from sympy import *
def Trapez(a,b,f,n):
    h = (b-a)/float(n)
    s = 0
    x = a
    for i in range(1,n,1):
        x = x+h
        s = s+ f(x)
    s = 0.5*(f(a)+f(b)) +s
    return h*s

#  function to compute pi
def function(x):
     return 4.0/(1+x*x)
```

Error analysis

The following extended version of the trapezoidal rule allows you to plot the relative error by comparing with the exact result. By increasing to 10^8 points one arrives at a region where numerical errors start to accumulate.

```
from math import log10
import numpy as np
from sympy import Symbol, integrate
import matplotlib.pyplot as plt
# function for the trapezoidal rule
def Trapez(a,b,f,n):
    h = (b-a)/float(n)
    s = 0
    x = a
    for i in range(1,n,1):
        x = x+h
        s = s+ f(x)
    s = 0.5*(f(a)+f(b)) +s
    return h*s
#  function to compute pi
def function(x):
     return 4.0/(1+x*x)
# define integration limits
a = 0.0;  b = 1.0;
# find result from sympy
# define x as a sumbol to be used by sumpy
```

Integrating numerical mathematics with calculus

The last example shows the potential of combining numerical algorithms with symbolic calculations, allowing you thereby to

- Validate and verify your algorithms.
- Include concepts like unit testing. It gives you the possibility to test and validate several or all parts of the code.
- Validation and verification are then included *naturally* and you can develop a better attitude to what is meant with an ethically sound scientific approach.
- The above example allows you to test the mathematical error of the algorithm for the trapezoidal rule by changing the number of integration points. You get to think error analysis from day one.

Additional benefits: A structured approach to solving problems

In this process we easily bake in

1. How to structure a code in terms of functions
2. How to make a module
3. How to read input data flexibly from the command line
4. How to create graphical/web user interfaces
5. How to write unit tests (test functions or doctests)
6. How to refactor code in terms of classes (instead of functions only)
7. How to conduct and automate large-scale numerical experiments
8. How to write scientific reports in various formats (LaTeX, HTML)

Additional benefits: A structure approach to solving problems

The conventions and techniques outlined here will save you a lot of time when you incrementally extend software over time from simpler to more complicated problems. In particular, you will benefit from many good habits:

1. New code is added in a modular fashion to a library (modules)
2. Programs are run through convenient user interfaces
3. It takes one quick command to let all your code undergo heavy testing
4. Tedious manual work with running programs is automated,
5. Your scientific investigations are reproducible, scientific reports with top quality typesetting are produced both for paper and electronic devices.