Computational Physics Lectures: How to write a scientific project, with examples

Morten Hjorth-Jensen[1,2]

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

2016

Overarching aims first

An essential part of this course is to enable you to do science via numerical experiments and develop projects which allow you to study complex systems. The aim is to enhance what we call algorithminc thinking.

Algorithm : A finite set of unambiguous instructions that, given some set of initial conditions, can be performed in a prescribed sequence to achieve a certain goal.

What do we mean with computing?

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

And this competence is about:

- derivation, verification, and implementation of algorithms
- understanding what can go wrong with algorithms
- overview of important, known algorithms
- understanding how algorithms are used to solve complicated problems
- reproducible science and ethics

- algorithmic thinking for gaining deeper insights about scientific problems

All these elements (and many more) will hopefully aid you in maturing and gaining a better understanding of the scientific process *per se*. **Writing good reports is a central element in achieving such insights**.

The standard situation we meet on a daily basis

The standard situation we meet at an almost daily basis:

- Theory+experiment+simulation is almost the norm in research and industry
- To be able to model complex systems with no simple answers. Solve real problems.
- Emphasis on insight and understanding of fundamental principles and laws in the Sciences.
- Be able to visualize, present, discuss, interpret and come with a critical analysis of the results, and develop a sound ethical attitude to own and other's work.
- Enhance reasoning about the scientific method

Again, a proper presentation of obtained results via good scientic reports, aids in including all the above aspects.

Practicalities

When working on the projects, we recommend strongly that you form teams of two to three participants. You **must** have a *github account* where we can monitor your progress and give you appropriate feedback.

Furthermore, when setting up your git repository for a given numerical project, you should create a folder where selected benchmarks are placed. These benchmarks could represent a calculation with specific input parameters. This makes your work reproducible, and allows us to see that your programs reproduce selected benchmarks. Furthermore, developing a habit of producing benchmarks, allows you to keep track of the results produced by different versions of your codes.

If you have not used version control before now, it is thus time to do so. Proper version control is central to a good ethical scientific conduct. We do require that you use some kind of version control software when working on the projects. We recommend strongly github. All lectures and additional material is available at the github address of the course

Programming languages

We recommend that you use either C++ or Fortran2008 for your projects. You can use Python as programming language, but normally the efficiency of Python for the problems addressed in this course is lower than for codes written in Fortran or C++. We recommend however that use Python as a scripting language for running codes and making plots, as well as using the ipython notebooks provided by us.

Some basic ingredients for a successful numerical project

When building up a numerical project there are several elements you should think of, amongst these we take the liberty of mentioning the following:

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)
6. How to refactor code in terms of classes (instead of functions only), in our case you think of a system and a solver class
7. How to conduct and automate large-scale numerical experiments
8. How to write scientific reports in various formats (LaTeX, HTML)

More basic ingredients

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.

The report: how to write a good scienfitic/technical report

What should it contain? A typical structure.

- An introduction where you explain the aims and rationale for the physics case and what you have done. At the end of the introduction you should give a brief summary of the structure of the report
- Theoretical models and technicalities. This is the methods section.
- Results and discussion
- Conclusions and perspectives
- Appendix with extra material
- Bibliography

Keep always a good log of what you do.

The report, the introduction

What should I focus on? Introduction. You don't need to answer all questions in a chronological order. When you write the introduction you could focus on the following aspects

- Motivate the reader, the first part of the introduction gives always a motivation and tries to give the overarching ideas
- What I have done
- The structure of the report, how it is organized etc

The report, discussion of methods

What should I focus on? Methods sections.

- Describe the methods and algorithms
- You need to explain how you implemented the methods and also say something about the structure of your algorithm and present some parts of your code
- You should plug in some calculations to demonstrate your code, such as selected runs used to validate and verify your results. The latter is extremely important!! A reader needs to understand that your code reproduces selected benchmarks and reproduces previous results, either numerical and/or well-known closed form expressions.

The report, results part

What should I focus on? Results.

- Present your results
- Give a critical discussion of your work and place it in the correct context.
- Relate your work to other calculations/studies
- An eventual reader should be able to reproduce your calculations if she/he wants to do so. All input variables should be properly explained.
- Make sure that figures and tables should contain enough information in their captions, axis labels etc so that an eventual reader can gain a first impression of your work by studying figures and tables only.

The report, conclusions and perspectives

What should I focus on? Conclusions.

- State your main findings and interpretations
- Try as far as possible to present perspectives for future work
- Try to discuss the pros and cons of the methods and possible improvements

The report, appendices

What should I focus on? additional material.

- Additional calculations used to validate the codes
- Selected calculations, these can be listed with few comments
- Listing of the code if you feel this is necessary

You can consider moving parts of the material from the methods section to the appendix. You can also place additional material on your webpage.

The report, references

What should I focus on? References.

- Give always references to material you base your work on, either scientific articles/reports or books.
- Refer to articles as: name(s) of author(s), journal, volume (boldfaced), page and year in parenthesis.
- Refer to books as: name(s) of author(s), title of book, publisher, place and year, eventual page numbers

How can I use Python and matplotlib to make figures for my report

Writing scripts in for example Python to produce high-quality figures allows you in a fast and efficient way to produce scientific results that can be included in a report. Furthermore, many operations can easily be automated, avoding thereby tedious repetitions of commands, as well as possible errors. Here we present a simple Python program which solves parts of project 2 for one quantum mechanical particle in a harmonic oscillator potential. The code plots the radial distribution of the three lowest-lying states, in addition to displaying the lowest three eigenvalues. It is easy to modify the trapping potential and run numerical experiments and test different boundary conditions. The plot is obtained using matplotlib, a Python plotting library which produces publication quality figures in a variety of formats and interactive environments across platforms.

The Python code

The code sets up the Hamiltonian matrix by defining the the minimun and maximum values of r with a maximum value of integration points. These are set in the initialization function. It plots the eigenfunctions of the three lowest eigenstates.

```
#Program which solves the one-particle Schrodinger equation
#for a potential specified in function
#potential(). This example is for the harmonic oscillator in 3d

from  matplotlib import pyplot as plt
import numpy as np
#Function for initialization of parameters
def initialize():
    RMin = 0.0
    RMax = 10.0
    lOrbital = 0
    Dim = 400
    return RMin, RMax, lOrbital, Dim
# Here we set up the harmonic oscillator potential
def potential(r):
    return r*r

#Get the boundary, orbital momentum and number of integration points
RMin, RMax, lOrbital, Dim = initialize()

#Initialize constants
Step    = RMax/(Dim+1)
DiagConst = 2.0 / (Step*Step)
NondiagConst =  -1.0 / (Step*Step)
OrbitalFactor = lOrbital * (lOrbital + 1.0)

#Calculate array of potential values
v = np.zeros(Dim)
r = np.linspace(RMin,RMax,Dim)
for i in xrange(Dim):
    r[i] = RMin + (i+1) * Step;
    v[i] = potential(r[i]) + OrbitalFactor/(r[i]*r[i]);
```

```
#Setting up a tridiagonal matrix and finding eigenvectors and eigenvalues
Hamiltonian = np.zeros((Dim,Dim))
Hamiltonian[0,0] = DiagConst + v[0];
Hamiltonian[0,1] = NondiagConst;
for i in xrange(1,Dim-1):
    Hamiltonian[i,i-1]  = NondiagConst;
    Hamiltonian[i,i]    = DiagConst + v[i];
    Hamiltonian[i,i+1]  = NondiagConst;
Hamiltonian[Dim-1,Dim-2] = NondiagConst;
Hamiltonian[Dim-1,Dim-1] = DiagConst + v[Dim-1];
# diagonalize and obtain eigenvalues, not necessarily sorted
EigValues, EigVectors = np.linalg.eig(Hamiltonian)
# sort eigenvectors and eigenvalues
permute = EigValues.argsort()
EigValues = EigValues[permute]
EigVectors = EigVectors[:,permute]
# now plot the results for the three lowest lying eigenstates
for i in xrange(3):
    print EigValues[i]
FirstEigvector = EigVectors[:,0]
SecondEigvector = EigVectors[:,1]
ThirdEigvector = EigVectors[:,2]
plt.plot(r, FirstEigvector**2 ,'b-',r, SecondEigvector**2 ,'g-',r, ThirdEigvector**2 ,'r-')
plt.axis([0,4.6,0.0, 0.025])
plt.xlabel(r'$r$')
plt.ylabel(r'Radial probability $r^2|R(r)|^2$')
plt.title(r'Radial probability distributions for three lowest-lying states')
plt.savefig('eigenvector.pdf')
plt.show()
```