Computational Physics Lectures: How to write a scientific report/thesis

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

Aug 16, 2019

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 computational part is actually a central element in your thesis. How do we document that?

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.

Some basic ingredients for a successful numerical project/thesis work

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, see slides on clean code later
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 thesis/report: how to write a good one

What should it contain? A typical structure.

- An abstract where you give the main summary of your work
- 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
- Implementation, the software you have developed
- Results and discussion
- Conclusions and perspectives
- Appendix with extra material
- Bibliography

Keep always a good log of what you do.

The report, the abstract

The abstract gives the reader a quick overview of what has been done and the most important results. Here is a typical example

As is well known, angular position and orbital angular momentum (OAM) of photons are a conjugate pair of variables that have been extensively explored for quantum information science and technology. In contrast, the radial degrees of freedom remain relatively unexplored. Here we exploit the radial variables, i.e., radial position

and radial momentum, to demonstrate Einstein-Podolsky-Rosen correlations between down-converted photons. In our experiment, we prepare various annular apertures to define the radial positions and use eigenmode projection to measure the radial momenta. The resulting correlations are found to violate the Heisenberg-like uncertainty principle for independent particles, thus manifesting the entangled feature in the radial structure of two-photon wave functions. Our work suggests that, in parallel with angular position and OAM, the radial position and radial momentum can offer a new platform for a fundamental test of quantum mechanics and for novel application of quantum information.

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 and your implementation

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 implementation part is central, this is where most of you will present what you have developed, how you have tested the code, its structure, validation and verification and much more.

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

Finally, figures and tables should be clear with labels on axes and good captions!

Where do I find scientific articles, books etc and examples of reports

- With a UiO IP number you can access freely all books and scientific journals available at our University library
- For scientific articles, go to for example the journal Physical Review Letters of the American Physical Society

Additional resources

- Use always version control software like Git with providers like GitHub, GitLab, BitBucket and other
- Introduce Unit testting from day one (see the how to write a good code slides).
- Use if you like IDEs like QtCreator and other. They have excellent debugging and analysis options.
- Use LaTex for typesetting and BibTex for managing references.
- Jupyter Notebooks and Jupyter Books are extremely useful. See also list over various Python libraries
- Look up previous Master theses at duo.uio.no

Python installers

We recommend two widely used distrubutions which set up all relevant dependencies for Python, namely

- Anaconda,

which is an open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system **conda**.

- Enthought canopy

is a Python distribution for scientific and analytic computing distribution and analysis environment, available for free and under a commercial license.

Furthermore, Google's Colab is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. Try it out!

Useful Python libraries

Here we list several useful Python libraries we strongly recommend (if you use anaconda many of these are already there)

- NumPy is a highly popular library for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays
- The pandas library provides high-performance, easy-to-use data structures and data analysis tools
- Xarray is a Python package that makes working with labelled multi-dimensional arrays simple, efficient, and fun!
- Scipy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering.
- Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.
- Autograd can automatically differentiate native Python and Numpy code. It can handle a large subset of Python’s features, including loops, ifs, recursion and closures, and it can even take derivatives of derivatives of derivatives
- SymPy is a Python library for symbolic mathematics.
- scikit-learn has simple and efficient tools for machine learning, data mining and data analysis
- TensorFlow is a Python library for fast numerical computing created and released by Google
- Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano
- And many more such as pytorch, Theano etc

Installing R, C++, cython or Julia

You will also find it convenient to utilize **R**. We will mainly use Python during lectures and in various projects and exercises. Those of you already familiar with **R** should feel free to continue using **R**, keeping however an eye on the parallel Python set ups. Similarly, if you are a Python afecionado, feel free to explore **R** as well. Jupyter/Ipython notebook allows you to run **R** codes interactively in your browser. The software library **R** is tuned to statistically analysis and allows for an easy usage of the tools we will discuss in these lectures.

To install **R** with Jupyter notebook follow the link here

Installing R, C++, cython, Numba etc

For the C++ aficionados, Jupyter/IPython notebook allows you also to install C++ and run codes written in this language interactively in the browser. Since we will emphasize writing many of the algorithms yourself, you can thus opt for either Python or C++ (or Fortran or other compiled languages) as programming languages.

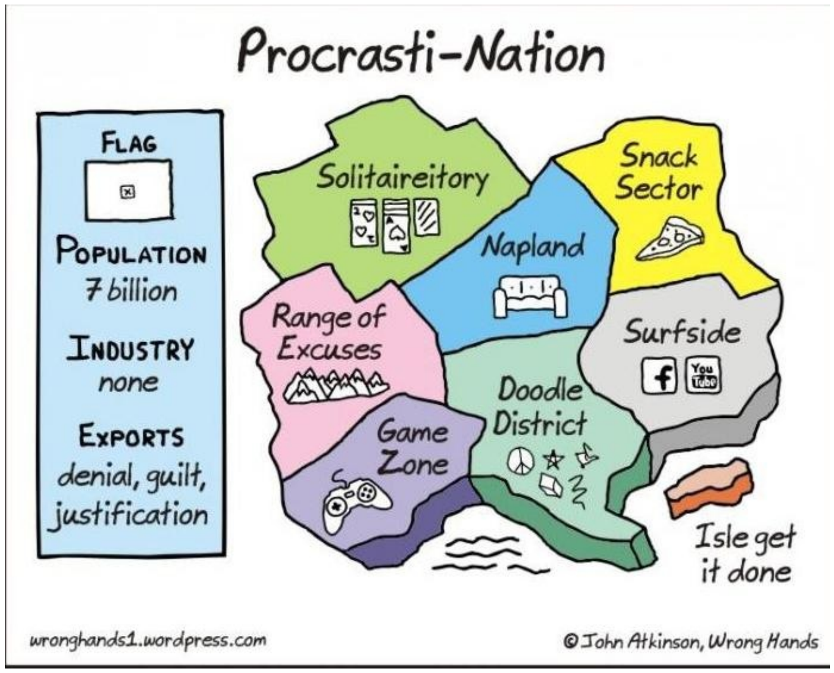
To add more entropy, **cython** can also be used when running your notebooks. It means that Python with the jupyter notebook setup allows you to integrate widely popular softwares and tools for scientific computing. Similarly, the Numba Python package delivers increased performance capabilities with minimal rewrites of your codes. With its versatility, including symbolic operations, Python offers a unique computational environment. Your jupyter notebook can easily be converted into a nicely rendered **PDF** file or a Latex file for further processing. For example, convert to latex as

```
pycod jupyter nbconvert filename.ipynb --to latex
```

And to add more versatility, the Python package SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) and is entirely written in Python.

Finally, if you wish to use the light mark-up language doconce you can convert a standard ascii text file into various HTML formats, ipython notebooks, latex files, pdf files etc with minimal edits. These lectures were generated using **doconce**.

Procrastination... the curse of all? Don't start too late with writing!



Enjoy this video

And research shows that procrastinating enhances creativity!!