Computational Physics

Lecture handouts

David Buscher <db106@cam.ac.uk>

January 2023

Some questions we will try to answer

- Many *mathematical problems* arise in the course of doing physics (e.g. solving a differential equation) where no analytic solution is available - how can we solve these problems in an efficient way on a computer ?

- What computations can we do with finite resources including time and space (memory)
- How *hard* — or *complex* — are the computational tasks we wish to perform? Can we quantify this?

- How do we represent and operate on the kind of data we encounter in physics (trajectories $\mathbf{x}(t)$, fields $\mathbf{E}(\mathbf{r}, t)$, etc.)?

- How do I start Python?

In this course you will learn

1. About the Python scientific stack (based on NumPy)
2. Its use in implementing some common algorithms in computational physics
3. Basic ideas of computational complexity used in the analysis of algorithms

Prerequisites

- Assume a knowledge of the Python language, including variables, control flow (if/else, loops, etc), and writing and using functions
- Refer to last year's IB course (which had an excellent handout)...
- ...and of course the internet
- For an absolutely bare bones intro to Python try the first half of this tutorial

Resources

- Course handouts (on TiS)
- Slides (linked from TiS)
- IB handouts (on TiS)
- Course website (linked from TiS)

Housekeeping

- Eight lectures. Mondays and Wednesdays at 10.00 in the Pippard
- Weeks 4-7: "practicals" - **computing exercises** with demonstrator assistance Monday, Wednesday and Friday afternoons
- Exercises to be completed and handed in by the end of Lent Term
- Exercises count for 0.2 units of further work, or roughly 2% of your final mark for the year

Schedule

- First lecture: Monday 22nd January
- Last lecture: Wednesday 14th February
- First practical session: Monday 12th February
- Last practical session: Friday 8th March
- Exercises hand-in: **16:00 on Friday 15th March** (last day of Full Term)

Computing Project

- You may choose to offer a Computational Physics project for one unit of further work
- Choose a problem from the project list. Analyse the problem, write and test Python code to investigate it, then write up your work in a report
- Project list is published by 17th February
- Deadline for submission of the project report is **16:00 on Monday 29th April** (first Monday of Full Easter term)

Getting going

Finding your way

- Everyone finds their own workflow for coding (language, editor, etc.)
- This is a roundup of some popular tools in the Python ecosystem

---

Your coding environment

- You will need to install the Python language (or run online)
- I recommend the Anaconda distribution
- Comes with all parts of the toolkit we'll need such as Jupyter notebooks and the major libraries NumPy and SciPy

---

- Try running `python` at the command line
- You should get something like

```
Python 3.9.12 (main, Apr  5 2022, 01:53:17)
[Clang 12.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Confirm you are using Python 3 (the command `python3` will also work and guarantee this if you happen to have Python 2 as the default)

- Prompt `>>>` indicates that you have started the Python interactive shell or REPL and are good to go:

```
print("Hello world!")
1 + 2
```

```
Hello world!
```

```
3
```

- To leave and return to command line, run `quit()` or `exit()`

IPython

- If you the above with `python` nice colour scheme is absent
- This is called syntax highlighting and provides a visual guide to the syntax of the language
- IPython is an interactive shell that provides syntax highlighting and much more
- If you have installed IPython (it comes with Anaconda) you can start it from the command line with `ipython`

Helpful features of IPython:

- Tab completion: hit `tab` to autocomplete. Particularly useful for viewing all properties or methods of an object:
- Typing `?obj` or `obj?` prints detailed information about the object `obj` (`??` provides additional detail)

- *Magic commands* prefixed by `%` provide additional functionality
- `%timeit` finds the executation time of a single line statement, which is useful when profiling the performance of code:

```python
%timeit L = [n ** 2 for n in range(1000)]
```

```
30.1 µs ± 582 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

- `%timeit` automatically runs several times to give statistics on execution time. For multiple lines you can use the `%%timeit` magic.
- Much more in the IPython documentation

Running a Python program

- Python code in a file with a `.py` extension can be run from the command line with

```
python hello_world.py
```

or

```
python -m hello_world
```

- In the latter case `-m` option tells interpreter to look for a *module* called `hello_world`

- From the IPython shell you can instead use

```
run hello_world.py
```

or just

```
run hello_world
```

Importing code

- A Python module is a file containing definition and statements
- Breaking long code into modules is good practice for writing clear and reusable software
- Users may not want to see the details of a function in order to be able to us it

---

- If I make the file `hello_world.py` containing the function:

```python
def hello():
    print("Hello world!")
```

- I can run this function by first importing the module:

```python
import hello_world
hello_world.hello()
```

```
Hello world!
```

- The function `hello` is accessed from the `hello_world` *namespace*
- This is to avoid confusion if more than one imported module has a function of the same name

---

- If you are confident this is not an issue and want more concise code you can do this:

```python
from hello_world import hello
hello()
```

```
Hello world!
```

- or even import everything with the wildcard `*`:

```python
from hello_world import *
hello()
```

```
Hello world!
```

- The issue with the latter is that it may introduce a whole bunch of names that can interfere with things you already defined

---

Packages

- A collection of modules in a folder is called a *package*
- You can import a package in the same way and access all the modules using the same `.` notation i.e. `package.module1`, `package.module2`, etc..
- Since explicit namespaces are preferred to avoid ambiguity use shorthands for the package or module you are importing:

```python
import numpy as np
np.arange(10)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

- (You can call it what you like, of course!)

Installing libraries

- 99% of the code you run will have been written by somebody else in the form of a library
- Package installation is handled by the command line utilities `pip` or `conda`, the latter being the package manager for the Anaconda distribution
- If you have NumPy and SciPy installed you won't need to worry about this too much

Editors

- Modern editors come with a huge number of tools that make writing code much easier
- Syntax highlighting, code completion, parameter information and documentation popups as you type
- These go under the general heading IntelliSense
- The latest hotness is GitHub Copilot: AI code suggestions
- (imo) these are all part of a continuum of productivity enhancements that enable people to write better code faster. Try them out!
- I use Visual Studio Code

Notebooks

- Software developers write `.py` files, modules and packages
- Scientists and others doing more exploratory work tend to favour a Notebook format that mixes code, text, and plots
- Dominant option is Jupyter notebook, which comes with the Anaconda distribution
- Start from command line with `jupyter notebook` (or from the Anaconda Navigator application)
- Opens a notebook as a web page in your browser, where it can be edited and saved. The default extension is `.ipynb`

- Jupyter notebooks can run code in different languages, but the default process is IPython
- Text cells can be formatted using Markdown and also support LaTeX equations, which is pretty handy for us
- Google has their own cloud version of the Jupyter notebook called Colab
- Try it out for free, though you have to pay for significant compute
- The “next generation” of the Jupyter notebook is called JupyterLab and can be started with `jupyter lab`
- Notebook files can be opened in either Jupyter Lab or Jupyter Notebook