Unit2NotesPart1

October 10, 2022

1 PHAS0007 Unit 2 Part 1: More about the Jupyter Notebook

Authors: Dr Becky Chislett, Dr Louise Dash and Dr Ben Waugh

Last updated 2022-10-05

1.1 How the Jupyter Notebook works

You do not need to know much about how the Jupyter Notebook application works for this course, so this section is for information only, to provide a bit of background for those who are interested. If you want a more detailed explanation, you may also want to look at - What is the Jupyter Notebook? in the Jupyter/IPython Notebook Quick Start Guide; - Chapter 3 : Mastering the Jupyter Notebook in the IPython Cookbook; - Architecture in the Jupyter Project Documentation.

1.1.1 Structure of the application

You will have noticed by now that when you start the Jupyter Notebook application on your computer, you actually end up working in a web browser. The web browser itself is not actually running your Python code though: this is handled by a separate process, which functions as a web server although it is (in this case) running on the same computer as the browser. This is known as a *client-server* structure: the web browser is a client, which passes requests to the server, and displays the results.

The server process itself actually uses a separate component called the *kernel* to execute the Python code itself. In fact one Jupyter Notebook server can run several kernels at the same time, for different notebooks, and these can even run different versions of Python or even different programming languages. In this course we will only use the Python 3 kernel.

1.1.2 Notebook files

Jupyter notebooks are stored on the computer as text files, using a syntax called JSON (JavaScript Object Notation), which is also used in many other applications. To specify that a particular file is a Jupyter notebook, we use the extension ".ipynb".

1.1.3 Terminology

The file extension is ".ipynb" because these documents used to be called "IPython notebooks". The project was renamed when it gained support for programming languages other than Python: the name "Jupyter" refers to the languages Julia, Python and R, although kernels are now available for many other languages.

In this course we are using "Jupyter Notebook" (with a capital N) to refer to the *application* that we are using, and "Jupyter notebook" (with a lower-case N) to refer to one of the documents we edit and run, containing Python code and Markdown text.

1.2 The Jupyter Notebook interface

1.2.1 Overview

You have already used the Jupyter Notebook to run and save Python code, but have not had to use many of the available tools. Here we give a brief description of the most important ones for this course, but you can find out more by taking the user interface tour: select *Help* in the menu bar, and then select *User Interface Tour*.

Jupyter notebooks get saved automatically every so often. To save them manually, click on the disk icon in the top left corner, or press CTRL-S. However, they don't always act exactly like other documents: in particular (and this can trip you up) there is only very limited "undo" (CTRL-Z) functionality. In general, CTRL-Z will work to undo what you have done in the current cell, but once you move to a different cell you won't be able to undo any further changes.

The other icons in the row at the top will be more useful when you are working on longer notebooks with multiple cells. The let you (in order): - add a new cell below the current one; - cut the current cell; - copy the current cell; - paste from the clipboard; - move the current cell up; - move the current cell down; - run the current cell (as an alternative to shift+enter); - interrupt the kernel (see below); - restart the kernel; - restart the kernel and rerun the entire notebook.

The dropdown menu lets you change the type of the current cell between *Code* (i.e. Python) and *Markdown* (i.e. text). You will not need the other option, *Raw NBConvert*, which is only needed to customise the way notebooks are converted to HTML documents.

1.2.2 Restarting and interrupting the kernel

The kernel is the process that actually runs your Python code, and holds the values of all the variables that you have defined. *Restarting* the kernel will clear all the variables from memory, and when you re-run cells they will be numbered starting from [1]. It's a good idea to do this every so often, as you can run into problems if you run cells in a different order (which is common when debugging and developing code) or if you define a variable, then delete it, but continue to refer to it somewhere else in your code.

You should restart the kernel and rerun your notebook in order before you upload your notebook to be marked. Even if the contents of your notebook look correct, you will not get full marks if it doesn't work when the marker runs it.

Interrupting the kernel is a less drastic step: it will stop the current cell from executing, which can be useful if it is taking too long, but will not clear variables from memory. You can tell if a cell is still executing because the label next to it will say

```
In [*]
```

instead of changing to a number.

1.3 Getting help

You'll notice the *Help* drop-down menu at the top of the screen: take a few minutes to explore here, especially the *User Interface Tour*.

One of the particularly helpful features of Jupyter is the way you can get help for a function. For example, place your cursor somewhere on the `abs()` function in the code cell below, and press SHIFT+TAB - you will see a documentation window pop up which gives you information about how to use that function and what it will output.

[1]:
```
abs(-2.5j + 3) # Press SHIFT+TAB with your cursor somewhere on the abs function␣
↪to get help.
```

[1]:
```
3.905124837953327
```

Another way to get help is to precede the function name with a question mark (?), like in the cell below:

[2]:
```
# Run this cell to open a documentation pane.
?print
```

1.4 Formatting text and maths with Markdown

1.4.1 Text

Markdown is a lightweight *markup language*: it provides a way of including additional information in a text file so that the text can be formatted nicely (with headings, subheadings, lists, embedded code samples etc.) while remaining fairly readable even as a plain text file, and being easy to edit.

Markdown provides quick ways to specify some basic types of formatting: - for a bulleted list, start each line with a - character or an asterisk *; - for* italic text *enclose the text with underscores, or with single asterisks* like this*; - for **bold text** enclose the text in double asterisks; - for section headings start a line with one or more # characters.

Double click on this cell to see how to do this.

Other features are explained in Markdown Cells in the Jupyter Notebook documentation. For even more possibilities, you can just include HTML in your Markdown cells, at the cost of making the text harder to edit.

1.4.2 Maths

It's useful to know how to write maths as well. Jupyter notebooks use LaTeX markup (pronounced Lay-Tech, not like rubber) which is either a particularly beautiful and elegant markup language for typesetting both maths and everything else (for all right-thinking people) or a form of torture that should be outlawed under the Geneva convention (for the not-yet-enlightened). You can find more information on LaTeX here: http://en.wikipedia.org/wiki/LaTeX

For our purposes, we won't need to do anything particularly complicated and you only need to know the very basics, which are fairly intuitive, so there will be no torture involved.

Double click on this cell, or on any other cell with maths in it, and you will be able to see exactly how it was generated. Here is a brief summary of the basics:

- to switch to math formatting within a paragraph, like this $x = 2y^3$, enclose the maths in dollar signs.
- to format an equation on a separate line, enclose it in double dollar signs, like this:

$$x = 2y^3$$

- Most greek letters are prefixed with a backslash followed by the name of the letter, eg \alpha, \beta, \gamma, \delta becomes $\alpha, \beta, \gamma, \delta$ when enclosed in dollar signs.
- Subscripts use an underline character, superscripts use ^. To format more than one character as a subscript or subscript, the characters must be grouped within curly brackets `{}`. For example $x_0 = y^2, x_1 = \alpha^{12}$
- Common functions like sin, cos, exp, are also prefixed by a backslash. For example

$$\exp(5\alpha) = \sin(2\pi x)$$

For most purposes in this course, you will frequently be able to copy and paste an equation from the session script (itself an Jupyter notebook) so don't worry about being expected to know more than this. In case you find it useful, there are also websites like http://www.codecogs.com/latex/eqneditor.php which let you pick maths symbols from a palette and convert them to LaTeX code.

1.5 Using Jupyter notebooks

A Jupyter notebook can be a full scientific report, with the bonus of executable code contained within it. When you are submitting your work, you should be aiming to make your notebook as clear and well structured as any other document.

1.5.1 When to use text cells and when to use comments

Sometimes it's not obvious when it's best to use text cells in your own notebooks and when to use code comments (starting with #). A good general rule of thumb is: - if you're describing something about the physics or maths of the problem you're solving, use a text cell; - if you're describing something about the Python code you're writing, use a comment in the code cell.

1.5.2 Using Jupyter notebooks elsewhere in your courses

Jupyter notebooks are a relatively new technology, and are being used more and more as people realise how useful they are. We hope that you too will find them useful beyond PHAS0007, whether just for making notes, or doing quick calculations and solving problems numerically.