Setting up Your Python Environment

Thomas J. Sargent and John Stachurski

March 13, 2020

1 Contents

• Overview 2
• Anaconda 3
• Jupyter Notebooks 4
• Installing Libraries 5
• Working with Python Files 6
• Exercises 7

2 Overview

In this lecture, you will learn how to

1. get a Python environment up and running
2. execute simple Python commands
3. run a sample program
4. install the code libraries that underpin these lectures

3 Anaconda

The core Python package is easy to install but not what you should choose for these lectures. These lectures require the entire scientific programming ecosystem, which

• the core installation doesn’t provide
• is painful to install one piece at a time.

Hence the best approach for our purposes is to install a Python distribution that contains

1. the core Python language and
2. compatible versions of the most popular scientific libraries.
The best such distribution is Anaconda.

Anaconda is

• very popular
• cross-platform
• comprehensive
• completely unrelated to the Nicki Minaj song of the same name

Anaconda also comes with a great package management system to organize your code libraries.

All of what follows assumes that you adopt this recommendation!

3.1 Installing Anaconda

To install Anaconda, download the binary and follow the instructions.

Important points:

• Install the latest version!
• If you are asked during the installation process whether you’d like to make Anaconda your default Python installation, say yes.

3.2 Updating Anaconda

Anaconda supplies a tool called conda to manage and upgrade your Anaconda packages.

One conda command you should execute regularly is the one that updates the whole Anaconda distribution.

As a practice run, please execute the following

1. Open up a terminal
2. Type `conda update anaconda`

For more information on conda, type conda help in a terminal.

4 Jupyter Notebooks

Jupyter notebooks are one of the many possible ways to interact with Python and the scientific libraries.

They use a browser-based interface to Python with

• The ability to write and execute Python commands.
• Formatted output in the browser, including tables, figures, animation, etc.
• The option to mix in formatted text and mathematical expressions.

Because of these features, Jupyter is now a major player in the scientific computing ecosystem.

Here’s an image showing execution of some code (borrowed from here) in a Jupyter notebook
While Jupyter isn’t the only way to code in Python, it’s great for when you wish to

- start coding in Python
- test new ideas or interact with small pieces of code
- share or collaborate scientific ideas with students or colleagues

These lectures are designed for executing in Jupyter notebooks.

### 4.1 Starting the Jupyter Notebook

Once you have installed Anaconda, you can start the Jupyter notebook.

Either

- search for Jupyter in your applications menu, or
- open up a terminal and type `jupyter notebook`
  - Windows users should substitute “Anaconda command prompt” for “terminal” in the previous line.
If you use the second option, you will see something like this

The output tells us the notebook is running at http://localhost:8888/

- **localhost** is the name of the local machine
- **8888** refers to port number 8888 on your computer

Thus, the Jupyter kernel is listening for Python commands on port 8888 of our local machine.

Hopefully, your default browser has also opened up with a web page that looks something like this
What you see here is called the Jupyter dashboard.

If you look at the URL at the top, it should be localhost:8888 or similar, matching the message above.

Assuming all this has worked OK, you can now click on New at the top right and select Python 3 or similar.

Here’s what shows up on our machine:
The notebook displays an active cell, into which you can type Python commands.

4.2 Notebook Basics

Let’s start with how to edit code and run simple programs.

4.2.1 Running Cells

Notice that, in the previous figure, the cell is surrounded by a green border.

This means that the cell is in edit mode.

In this mode, whatever you type will appear in the cell with the flashing cursor.

When you’re ready to execute the code in a cell, hit Shift-Enter instead of the usual Enter.
4.2.2 Modal Editing

The next thing to understand about the Jupyter notebook is that it uses a modal editing system.

This means that the effect of typing at the keyboard depends on which mode you are in. The two modes are

1. Edit mode
   - Indicated by a green border around one cell, plus a blinking cursor
   - Whatever you type appears as is in that cell

1. Command mode
   - The green border is replaced by a grey (or grey and blue) border
   - Keystrokes are interpreted as commands — for example, typing b adds a new cell below the current one

To switch to
   - command mode from edit mode, hit the Esc key or Ctrl-M
   - edit mode from command mode, hit Enter or click in a cell

The modal behavior of the Jupyter notebook is very efficient when you get used to it.
4.2.3 Inserting Unicode (e.g., Greek Letters)

Python supports unicode, allowing the use of characters such as \(\alpha\) and \(\beta\) as names in your code.

In a code cell, try typing \texttt{\alpha} and then hitting the tab key on your keyboard.

4.2.4 A Test Program

Let’s run a test program.

Here’s an arbitrary program we can use: \url{http://matplotlib.org/3.1.1/gallery/pie_and_polar_charts/polar_bar.html}.

On that page, you’ll see the following code:

```
In [1]:
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Fixing random state for reproducibility
np.random.seed(19680801)

# Compute pie slices
N = 20
\theta = np.linspace(0.0, 2 * np.pi, N, endpoint=False)
radii = 10 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)
colors = plt.cm.viridis(radii / 10.)

ax = plt.subplot(111, projection='polar')
ax.bar(\theta, radii, width=width, bottom=0.0, color=colors, alpha=0.5)
plshow()
```

![Polar chart](image)
Don’t worry about the details for now — let’s just run it and see what happens.

The easiest way to run this code is to copy and paste it into a cell in the notebook.

Hopefully you will get a similar plot.

4.3 Working with the Notebook

Here are a few more tips on working with Jupyter notebooks.

4.3.1 Tab Completion

In the previous program, we executed the line `import numpy as np`

- NumPy is a numerical library we’ll work with in depth.

After this import command, functions in NumPy can be accessed with `np.function_name` type syntax.

- For example, try `np.random.randn(3)`.

We can explore these attributes of `np` using the Tab key.

For example, here we type `np.ran` and hit Tab
Jupyter offers up the two possible completions, `random` and `rank`.

In this way, the Tab key helps remind you of what’s available and also saves you typing.

### 4.3.2 On-Line Help

To get help on `np.rank`, say, we can execute `np.rank?`.

Documentation appears in a split window of the browser, like so

Clicking on the top right of the lower split closes the on-line help.

### 4.3.3 Other Content

In addition to executing code, the Jupyter notebook allows you to embed text, equations, figures and even videos in the page.

For example, here we enter a mixture of plain text and LaTeX instead of code
Next we **Esc** to enter command mode and then type **m** to indicate that we are writing **Markdown**, a mark-up language similar to (but simpler than) **LaTeX**.

(You can also use your mouse to select **Markdown** from the **Code** drop-down box just below the list of menu items)

Now we **Shift+Enter** to produce this
4.4 Sharing Notebooks

Notebook files are just text files structured in JSON and typically ending with `.ipynb`.

You can share them in the usual way that you share files — or by using web services such as nbviewer.

The notebooks you see on that site are static html representations.

To run one, download it as an ipynb file by clicking on the download icon at the top right.

Save it somewhere, navigate to it from the Jupyter dashboard and then run as discussed above.

4.5 QuantEcon Notes

QuantEcon has its own site for sharing Jupyter notebooks related to economics – QuantEcon Notes.

Notebooks submitted to QuantEcon Notes can be shared with a link, and are open to comments and votes by the community.
5 Installing Libraries

Most of the libraries we need come in Anaconda. Other libraries can be installed with pip. One library we’ll be using is QuantEcon.py. You can install QuantEcon.py by starting Jupyter and typing

```
!pip install --upgrade quantecon
```

into a cell. Alternatively, you can type the following into a terminal

```
pip install quantecon
```

More instructions can be found on the library page. To upgrade to the latest version, which you should do regularly, use

```
pip install --upgrade quantecon
```

Another library we will be using is interpolation.py. This can be installed by typing in Jupyter

```
!pip install interpolation
```

6 Working with Python Files

So far we’ve focused on executing Python code entered into a Jupyter notebook cell. Traditionally most Python code has been run in a different way. Code is first saved in a text file on a local machine. By convention, these text files have a .py extension. We can create an example of such a file as follows:

```
In [2]: %%file foo.py

    print("foobar")

    Overwriting foo.py
```

This writes the line `print("foobar")` into a file called `foo.py` in the local directory. Here `%%file` is an example of a cell magic.
6.1 Editing and Execution

If you come across code saved in a *.py file, you’ll need to consider the following questions:

1. how should you execute it?
2. How should you modify or edit it?

6.1.1 Option 1: JupyterLab

JupyterLab is an integrated development environment built on top of Jupyter notebooks. With JupyterLab you can edit and run *.py files as well as Jupyter notebooks. To start JupyterLab, search for it in the applications menu or type jupyter-lab in a terminal. Now you should be able to open, edit and run the file foo.py created above by opening it in JupyterLab. Read the docs or search for a recent YouTube video to find more information.

6.1.2 Option 2: Using a Text Editor

One can also edit files using a text editor and then run them from within Jupyter notebooks. A text editor is an application that is specifically designed to work with text files — such as Python programs. Nothing beats the power and efficiency of a good text editor for working with program text. A good text editor will provide

- efficient text editing commands (e.g., copy, paste, search and replace)
- syntax highlighting, etc.

Right now, an extremely popular text editor for coding is VS Code. VS Code is easy to use out of the box and has many high quality extensions. Alternatively, if you want an outstanding free text editor and don’t mind a seemingly vertical learning curve plus long days of pain and suffering while all your neural pathways are rewired, try Vim.

7 Exercises

7.1 Exercise 1

If Jupyter is still running, quit by using Ctrl-C at the terminal where you started it. Now launch again, but this time using jupyter notebook --no-browser. This should start the kernel without launching the browser. Note also the startup message: It should give you a URL such as http://localhost:8888 where the notebook is running. Now
1. Start your browser — or open a new tab if it’s already running.

2. Enter the URL from above (e.g. http://localhost:8888) in the address bar at the top.

You should now be able to run a standard Jupyter notebook session.
This is an alternative way to start the notebook that can also be handy.

### 7.2 Exercise 2

This exercise will familiarize you with git and GitHub.

*Git* is a version control system — a piece of software used to manage digital projects such as code libraries.

In many cases, the associated collections of files — called repositories — are stored on GitHub.

GitHub is a wonderland of collaborative coding projects.
For example, it hosts many of the scientific libraries we’ll be using later on, such as this one.

Git is the underlying software used to manage these projects.

Git is an extremely powerful tool for distributed collaboration — for example, we use it to share and synchronize all the source files for these lectures.

There are two main flavors of Git

1. the plain vanilla command line Git version
2. the various point-and-click GUI versions
   - See, for example, the GitHub version

As the 1st task, try

1. Installing Git.
2. Getting a copy of QuantEcon.py using Git.

For example, if you’ve installed the command line version, open up a terminal and enter.

```
git clone https://github.com/QuantEcon/QuantEcon.py
```

(This is just *git clone* in front of the URL for the repository)

As the 2nd task,

1. Sign up to GitHub.
2. Look into ‘forking’ GitHub repositories (forking means making your own copy of a GitHub repository, stored on GitHub).
3. Fork QuantEcon.py.
4. Clone your fork to some local directory, make edits, commit them, and push them back up to your forked GitHub repo.

5. If you made a valuable improvement, send us a pull request!

For reading on these and other topics, try

- The official Git documentation.
- Reading through the docs on GitHub.
- Pro Git Book by Scott Chacon and Ben Straub.
- One of the thousands of Git tutorials on the Net.