1 Contents

- Overview 2
- Anaconda 3
- Jupyter Notebooks 4
- Installing Libraries 5
- Working with Files 6
- Editors and IDEs 7
- Exercises 8

2 Overview

In this lecture, you will learn how to

1. get a Python environment up and running with all the necessary tools
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 free Python distribution that contains
1. the core Python language and
2. 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

Installing Anaconda is straightforward: 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.
- Otherwise, you can accept all of the defaults.

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 possibilities, Jupyter is fast turning into a major player in the scientific computing ecosystem.

Here’s an image showing execution of some code (borrowed from here) in a Jupyter notebook:

You can find a nice example of the kinds of things you can do in a Jupyter notebook (such as include maths and text) here.

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 \texttt{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 (click to enlarge)

![Terminal output](image)

The output tells us the notebook is running at \url{http://localhost:8888/}

- \texttt{localhost} is the name of the local machine
- \texttt{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 (click to enlarge)
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.

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. As a result, you can type in Python code and it will appear in the cell. When you’re ready to execute the code in a cell, hit **Shift-Enter** instead of the usual **Enter**.
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
   - Whatever you type appears as is in that cell

1. Command mode
   - The green border is replaced by a grey border
   - Key strokes 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 a little tricky at first but very efficient when you get used to it.

User Interface Tour
At this stage, we recommend you take your time to

• look at the various options in the menus and see what they do
• take the “user interface tour”, which can be accessed through the help menu

Inserting Unicode (e.g., Greek Letters)
Python 3 introduced support for unicode characters, allowing the use of characters such as \( \alpha \) and \( \beta \) in your code.

Unicode characters can be typed quickly in Jupyter using the tab key.

Try creating a new code cell and typing \( \alpha \), then hitting the tab key on your keyboard.

A Test Program
Let’s run a test program.

Here’s an arbitrary program we can use: http://matplotlib.org/1.4.1/examples/pie_and_polar_charts/polar_bar_demo.html.

On that page, you’ll see the following code

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

N = 20
θ = np.linspace(0.0, 2 * np.pi, N, endpoint=False)
radii = 10 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)

ax = plt.subplot(111, polar=True)
bars = ax.bar(θ, radii, width=width, bottom=0.0)

# Use custom colors and opacity
for r, bar in zip(radii, bars):
    bar.set_facecolor(plt.cm.jet(r / 10.))
    bar.set_alpha(0.5)

plt.show()
```
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 into a cell in the notebook.
(In older versions of Jupyter you might need to add the command `%matplotlib inline`
before you generate the figure)

4.3 Working with the Notebook

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

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 (click to enlarge)
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.

**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.

**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 \texttt{pip}.
One library we’ll be using is \texttt{QuantEcon.py}.
You can install \texttt{QuantEcon.py} by starting Jupyter and typing

\texttt{!pip install --upgrade quantecon}

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

\texttt{pip install quantecon}

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

\texttt{pip install --upgrade quantecon}

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

\texttt{!pip install interpolation}

6 Working with Files

How does one run a locally saved Python file?
There are a number of ways to do this but let’s focus on methods using Jupyter notebooks.

6.1 Option 1: Copy and Paste

The steps are:

1. Navigate to your file with your mouse/trackpad using a file browser.
2. Click on your file to open it with a text editor.
3. Copy and paste into a cell and \texttt{Shift-Enter}.

6.2 Method 2: Run

Using the \texttt{run} command is often easier than copy and paste.

- For example, \texttt{%run test.py} will run the file \texttt{test.py}. 

(You might find that the % is unnecessary — use %automagic to toggle the need for %)

Note that Jupyter only looks for test.py in the present working directory (PWD).

If test.py isn’t in that directory, you will get an error.

Let’s look at a successful example, where we run a file test.py with contents:

```python
for i in range(5):
    print('foobar')
```

foobar
foobar
foobar
foobar
foobar

Here’s the notebook (click to enlarge)

Here

- `pwd` asks Jupyter to show the PWD (or `%pwd` — see the comment about automagic above)
  
  - This is where Jupyter is going to look for files to run.
  - Your output will look a bit different depending on your OS.
• `ls` asks Jupyter to list files in the PWD (or `%ls`)
  
  – Note that `test.py` is there (on our computer, because we saved it there earlier).

• `cat test.py` asks Jupyter to print the contents of `test.py` (or `!type test.py` on Windows)

• `run test.py` runs the file and prints any output

### 6.3 But File X isn’t in my PWD!

If you’re trying to run a file not in the present working directory, you’ll get an error.

To fix this error you need to either

1. Shift the file into the PWD, or
2. Change the PWD to where the file lives

One way to achieve the first option is to use the **Upload** button

• The button is on the top level dashboard, where Jupyter first opened to
• Look where the pointer is in this picture

The second option can be achieved using the **cd** command

• On Windows it might look like this `cd C:/Python27/Scripts/dir`
• On Linux / OSX it might look like this `cd /home/user/scripts/dir`

Note: You can type the first letter or two of each directory name and then use the tab key to expand.
6.4 Loading Files

It’s often convenient to be able to see your code before you run it. In the following example, we execute `load white_noise_plot.py` where `white_noise_plot.py` is in the PWD. (Use `%load` if automagic is off)

Now the code from the file appears in a cell ready to execute.

6.5 Saving Files

To save the contents of a cell as file `foo.py`

- put `%file foo.py` as the first line of the cell
- Shift+Enter

Here `%file` is an example of a cell magic.

7 Editors and IDEs

The preceding discussion covers most of what you need to know to interact with this website.
However, as you start to write longer programs, you might want to experiment with your workflow. There are many different options and we mention them only in passing.

7.1 JupyterLab

_JupyterLab_ is an integrated development environment centered around Jupyter notebooks. It is available through Anaconda and will soon be made the default environment for Jupyter notebooks. Reading the docs or searching for a recent YouTube video will give you more information.

7.2 Text Editors

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.

Among the most popular are _Sublime Text_ and _Atom_. For a top quality open source text editor with a steeper learning curve, try _Emacs_. 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_.

7.3 Text Editors Plus IPython Shell

A text editor is for writing programs. To run them you can continue to use Jupyter as described above. Another option is to use the excellent _IPython shell_. To use an IPython shell, open up a terminal and type _ipython_. You should see something like this
The IPython shell has many of the features of the notebook: tab completion, color syntax, etc.

It also has command history through the arrow key.

The up arrow key to brings previously typed commands to the prompt.

This saves a lot of typing...

Here’s one set up, on a Linux box, with

- a file being edited in **Vim**
- an IPython shell next to it, to run the file
7.4 IDEs

IDEs are Integrated Development Environments, which allow you to edit, execute and interact with code from an integrated environment.

One of the most popular in recent times is VS Code, which is now available via Anaconda. We hear good things about VS Code — please tell us about your experiences on the forum.

8 Exercises

8.1 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.
8.2 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 an exercise, 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)

Even better,

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.

References