Universidad Complutense de Madrid

GUIDE TO JUPYTER NOTEBOOKS

FOR EDUCATIONAL PURPOSES

Authors:
Eduardo Cabrera Granado
Elena Díaz García

Translation by:
Álvaro Díaz Fernández

Contents

Introduction

Jupyter is an open source project that aims to allow the usage of different programming languages in one computational platform. In fact, its name originates from three of the most popular programming languages for scientific computing: Julia, Python and R. It is built upon the Jupyter Notebook web interface, which brings together the possibilities of executing code, including text along with LaTeX equations (via MathJax), video, and everything that can be visualized within a browser. On the other hand, its features can be extended by means of external tools, an example of which is *nbconvert*, which can handle conversion to LaTeX and PDF formats. Also available is the possibility to include web presentations based on Reveal.js, or interactive widgets within the document via *ipywidgets*. Jupyter's development has been supported by companies such as Microsoft or Google, as well as boosted by OpenDreamKit, a Horizon 2020 European Research Infrastructure project.

The popularity of Jupyter for educational purposes has grown exponentially due to its flexibility, ease of access through a browser, and the added value brought by the possibility to isolate the document itself from the computational core. This architecture allows to forget about local installation. The lecturer can use an external server (or even their laptop) to supply students with lectures, slides, or interactive assignments. This feature enables to completely remove the most common obstacle that arises when trying to put forward a new document or program: having the student to install it locally on their personal laptop. This new kind of documents makes it possible to create a rich variety of lectures and supplemental material that can include computing modules. These in turn can be used for self-study, but also for collaborative learning. Eventually, students of science will be faced with some

kind of computer program during their studies, as well as in their future workplace. Thus, including this kind of computational elements endows the student with a highly valued transversal skill. Moreover, it contributes to achieve a deeper level of understanding for those motivated students willing to explore the details of the subject, even if those fine points are out of the syllabus.

Throughout this guide, we will introduce the main characteristics of the Jupyter Notebook interface and its installation. We will also provide useful lectures and assignments for scientific studies. At the same time, we will describe different alternatives to supply students with a server of this kind, as well as specific add-ons for teaching, such as the entire management of a course implemented in Jupyter Notebooks.

Jupyter Notebook

Jupyter Notebook is an open source web interface that enables to include text, video, audio, images, along with the possibility to execute code from different programming languages. This execution is accomplished by way of communication to a computational core (Kernel). By default, Jupyter Notebook only includes the Python kernel. However, being an open source project, it has been possible to increase the number of available kernels [1]. These include kernels for Octave, Julia, R, Haskell, Ruby, C/C++, Fortran, Java, SageMath, Scala, Matlab and Mathematica. Thus, this interface can mark a milestone in the standardization of ways to deliver scientific content, without being circumscribed to a single language. In fact, the name Jupyter is a combination of the most popular, open source programming languages for scientific computing: Ju-lia, Py-thon and R.

This versatility has allowed to spread its use into both educational and research environments. In this last area, it is becoming more and more common to include Jupyter Notebooks with calculations, data and additional figures to scientific papers. For instance, the LIGO team, who experimentally discovered gravitational waves in October 2017, have provided their data and calculations in this format In the educational area, the course *AeroPython* by Lorena Barba from The George Washington University, stands out as a course entirely performed in Jupyter Notebooks.

History

Jupyter Notebook originates from IPython Notebook, with the same basic functionalities, yet including the possibility to execute code in multiple languages. On the

other hand, the web interface, common to both Jupyter Notebook and its predecessor, means the realization of an open source çomputational notebook"based on those included in Mathematica or Maple. The first attempt to achieve such a notebook was made by William Stein for Sage, successfully culminating in the renowned Sage Notebooks. Fernando Pérez and Robert Kern from the University of Berkeley (California, USA) where the first developers of Jupyter Notebook. Their development was based on the programming language Python, because of it being flexible, open source and vastly adopted by the Scientific Community. Initially, the goal was not to create a web interface but an interactive interpreter for Python, named IPython. However, from 2010 the IPython development team could give birth to the Jupyter Notebook interface.

From then on, the development of this çomputational notebooks"has grown at a fast rate as well as their use. For instance, the most important platform for the development of open source projects, GitHub, allows to visualize Jupyter Notebooks directly on their webpage, hosting more than a million of this kind of documents on their servers for their free access.

User Interface Description

Although the installation process will be described in the next section, let us assume Jupyter Notebook is already installed. In order to launch it, we will type `jupyter notebook` in a console. Automatically, a new tab in our browser will show up, displaying a list with the contents of the directory where the console was at[a]. This is known as the Jupyter Dashboard. It is possible to install add-ons, such as Nbgrader, a course management and automatic correction system, which will also appear in this control panel. Another important element is the *Upload* button, at the top-right corner. This button is not particularly interesting if Jupyter Notebook is executed locally, in sharp contrast to the case where it is executed remotely from a different computer and a browser, where it becomes very convenient.

[a]The directory where we are at can be checked by typing in a console `pwd`, short for *print working directory*.

We will click *New* in order to generate a new Jupyter Notebook. Once we have done it, we can choose the computational kernel among those we have installed. By default, we will only have Python. An example of a Jupyter Notebook can be seen in Figure 1.

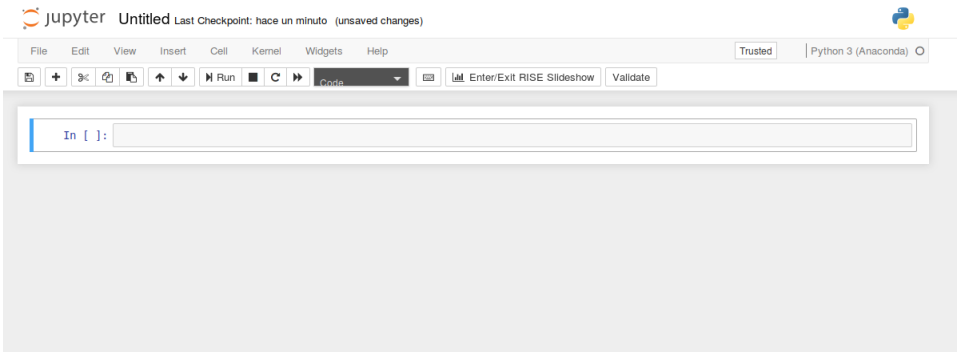

Figura 1. *Ejemplo de un documento Jupyter Notebook.*

There are different menus at our disposal at the top part of the document. These allow to edit, insert and delete parts of the document. Additionally, they let us download our document in different formats (e.g. in PDF). Also important is a menu which enables to choose and control the computational kernel (it even allows to stop the kernel if needed). Finally, there is a *Help* menu with a great deal of information concerning Jupyter Notebook's features.

The document itself is divided into separate cells, the behavior of which can be chosen from different options: different size headings[b], plain or enhanced text by means of the Markdown language, or code. Apart from this options, specific libraries allow to import images or videos, include webpage links or even embed a whole webpage within the document.

In order to insert enhanced text, the Jupyter team decided to go for the Markdown syntax, a powerful tool to include lists, bold and italics text, tables or images. There are now myriads of tutorials about this syntax, so we will focus exclusively

[b]This feature is actually deprecated. Now this is done directly within a Markdown cell, preceding the heading text by the character #. The heading size will be controlled by the number of # (six max.) before the text. The larger the number of #, the smaller the size will be.

in the most used formatting options:

- Bold and italics text: in order to include bold text, the word must be placed within two asterisks. For instance, `**word**` will output **word**. For italics, the syntax is very much the same as with bold text, but only an asterisk is needed instead of two. For example, `*word*` will output *word*. Alternatively, in both two cases the asterisks can be substituted by underscores.

- Lists: these are very straightforward. For unordered lists, asterisks play the role of bullets, whereas for ordered lists we will write a number followed by a dot. Markdown takes care of assigning correctly a number to each item.

- Including images: the syntax to include images is `![alternative name](image location)`. If the image cannot be loaded, the alternative name will appear instead. The image location can be either that of a local image or a link to a webpage.

- Including HTML code: Markdown is a subcategory of HTML. Thus, if further control of format is needed, it can be directly included in HTML.

- Links: text cells can include links, both to other parts of the document and webpages, or even to local files. The syntax is `[text](link)`.

- Mathematical formulae: thanks to MathJax, LaTeX code can be included to show all types of mathematical formulae and expressions. Equations in line with the text are written in between dollar signs, as in (`$...$`), whereas equations within two paragraphs use two dollar sings instead, that is, (`$$...$$`).

All these features will let us use text cells a in a versatile way to document code from other cells. For instance, it can be used to explain some theoretical concept in detail, which can be followed by an exercise related to it.

When we create a new cell, it automatically corresponds to a code cell. The programming language will be determined by the kernel used for that document.

As of now, it is not possible to mix two programming languages within the same document[c].

Installing Jupyter Notebook

In order to install Jupyter Notebook, it is necessary to have Python installed beforehand. Even though different programming languages can be used in Jupyter, Python is Jupyter's backbone. Due to its ever-increasing popularity in the scientific environment, scientific computation suites have arisen, offering a vast quantity of Python modules. The most well-known is Anaconda, developed by Continuum Analytics. Apart from the Python modules included in Anaconda, this distribution also includes a package manager, which has eased the installation and management processes of scientific Python libraries in an unprecedented manner.

By default, Anaconda installs Jupyter Notebook. Thus, independent of the operating system, it is most recommended to install this scientific Python distribution. For that matter, Anaconda can be downloaded free of charge at their download webpage [2]. Once installed, following the steps described in that webpage, we will have Jupyter Notebook ready to use. In order to use it, it is as simple as typing `jupyter notebook` in a console.

Online platforms for Jupyter without installation

Even though installing Jupyter Notebook is almost straightforward nowadays, one of the essential features for its educational use is the possibility to disjoin the interface from the computational kernel. This way, a student can have access to the notebooks via a browser, whilst the execution of code is performed elsewhere, either the teacher's computer or a server in the cloud. We will now consider some of these alternatives, focusing on those that also allow to execute code instead of those offering a static view of the document. Examples of the latter are GitHub [3] or

[c]This can be achieved by installing SoS, a Jupyter Notebook kernel that allows to include different kernels in one notebook: `https://vatlab.github.io/sos-docs/`

NbViewer [4].

1. **CoCalc**: this online computational environment [5] provides different unique tools for their educational use: it allows to collaboratively edit Jupyter Notebook documents, increasing the possibilities for teamwork; it endows the teacher with a course and tasks' manager, which enables them to upload educational content, as well as to mark and provide feedback very easily. Moreover, it is free of charge and students can still use their account once the course has ended, allowing them to take advantage of simulations and explanations in future subjects. There is also the option for paid access, providing more stable servers and a larger number of computational resources in comparison to the free accounts.

2. **Gryd**: this server [6] supplies free accounts for students as well as a course manager, although this option is not free of charge.

3. **Google Colab**: this Jupyter Notebook server [7] includes the option to work collaboratively within the same document, very much like in Google Documents. This functionality makes it specially appealing as an excellent server for tasks requiring teamwork.

4. **MyBinder**: this server [8] allows to create a temporal server to execute all Jupyter Notebooks allocated within a GitHub repository. It is free of charge.

5. **JupyterHub**: this tool falls out of the category of web-based servers for executing Jupyter Notebooks. Instead, it is a Jupyter tool focused on providing a multiuser server that can be used either in class or by a research team, to name a few examples. This tool will be discussed in detail later in this handbook.

Export to other formats

Jupyter Notebook documents are easily accessed via a browser. However, it is sometimes useful or even necessary to have its contents in a different format, as a

Python file for instance. In order to ease this conversion, Jupyter includes a tool named **nbconvert**. It is possible to convert a notebook to different static formats, that is, formats where the cells cannot be executed. These include: HTML, LaTeX, Markdown, reStructuredText, executable Python scripts. It is also possible to convert to a presentation format, which requires the previous installation of *Pandoc* via a package manager or their webpage [9].

This tool can be accessed via a console or even within the document itself. The latter can be done through the *File* menu at the top-left of the document as `File ->Download as...`. However, in order to exploit all capabilities included in **nbconvert**, we indicate the main commands to use in a console. The most basic command would be

```
$ jupyter nbconvert --to FORMAT notebook.ipynb
```

where `FORMAT` is the desired format. For instance, if we want to convert our document to a LaTeX file, we would write

```
$ jupyter nbconvert --to latex notebook.ipynb
```

Once the .tex file is generated, we can convert it to PDF by means of `pdflatex` or any LaTeX editor. However, it is possible to do it directly with **nbconvert** as follows,

```
$ jupyter nbconvert --to latex notebook.ipynb --post PDF
```

It is also possible to modify our generated .tex file to a book format by adding `--template book`. It is important to notice that the presentation format will not generate a PowerPoint presentation or anything alike. Instead, it will generate a Reveal.js presentation in HTML, which in order to be visualized requires a browser. This option is achieved by adding `--post serve` to the previous command. This kind of presentations, although more complicated to display due to the need of a

browser to execute HTML, include a lot more options than ordinary presentations. Moreover, they become shareable much more easily through different platforms and operating systems.

More information can be found at [10, 11].

Online resources

The amount of online resources devoted to Jupyter Notebook keeps increasing at an exponential rate. Thus, the following is not an exhaustive list of resources. Rather, it is a very short selection of these, whose validity may change during time. However, due to the relevance of these sites, it is likely that, even though new links will appear, the ones considered below will still stand out in the future.

1. Jupyter Notebook Documentation [12].
2. Reddit forum devoted to Jupyter Notebook [13].
3. An interesting collection of Jupyter Notebooks [14].
4. CoCalc's development webpage [15].

Since the main programming language of the notebooks included at the Appendix is Python, in the following chapter we will discuss some of its characteristics. Specifically, we will compare to other computational programs such as Matlab, widely used in scientific environments.

Python

Among the different alternatives for open source software devoted to computational programming, Python has become one of the most used options. Thus, learning this programming language can be very beneficial for students in their future careers. Moreover, there is a vast amount of scientific modules that are easily imported, matching the capabilities of Python to those of commercial software.

The most relevant libraries in the scientific environment are:

- **SciPy**: it groups together many relevant functions for numerical calculations.
- **NumPy**: it provides specific functions for vector and matrix calculations.
- **SymPy**: it encompasses all necessary functions for symbolic calculations.
- **Matplotlib**: it contains tools for 2D and 3D plots (similar to those in MATLAB).
- **Mayavi**: specific library for 3D plots.
- **Pandas**: specific library for data management and analysis.

Apart from these, there are numerous libraries designes for specific needs, such as image processing (openCV), machine learning (scikit-learn) and many more.

Most common commands

We will now enumerate the most common commands in Python to perform scientific operations in Jupyter Notebook using Python[d] as the computational kernel.

[d]Notice that we will consider Python 3 as the computational kernel. There are not many differences with Python 2, except for some syntax subtleties, an example of which corresponds to the

- Library management

  - **Import all functions within a library**: taking the NumPy library as an example, we would have to write in our document

    ```
    from numpy import *
    ```

    There are functions appearing in two different libraries which act differently. Hence, it is sometimes useful to import a library with a distinctive prefix. Moreover, specially when writing Python scripts, it is good practice to include this prefix for a future reading of the script and to fix possibly appearing bugs. It is common to choose the prefix np for the NumPy library. This is accomplished by

    ```
    import numpy as np
    ```

  - **Import a specific function**: sometimes we just want to import a function from a library, instead of the whole library. As an example, let us consider importing the cosine function from the NumPy library. In this case, we would write

    ```
    from numpy import cos
    ```

  - **Advice for educational applications**: in general, for scientific subjects, the easiest thing to do would be to use the PyLab environment. Once executed, this environment automatically imports the most used functions and allows to write code in a very similar way to MATLAB. Moreover, in order for the plots to appear in the notebook instead of in a different tab, it is necessary to include the `inline` command. Both aspects are taken care of by writing at the beginning of the file

    ```
    %pylab inline
    ```
    [e]

`print` command.

[e] Another alternative would be to write the following three lines at the beginning of any document:

```
from numpy import *
from matplotlib.pyplot import *
%matplotlib inline
```

- Commands for numerical calculations:
  - **Help**: we can retrieve more information from a specific function by typing

    `help(function)` or `function?`
  - **Printing on screen**: in order to display text or the value of a variable, `x`, when executing a cell, we would write

    ```
    print('Variable x has the value')
    print(x)
    ```
  - **Inserting an image**: this is done by importing the function `Image`. Its syntax is[f]

    ```
    from IPython.display import Image
    Image(filename="TestFigure")
    ```

    As an alternative, an image can also be included directly in a Markdown cell by using Markdown's syntax, or even with HTML for further customization options, such as changing the size or position of the image in a cell. In order to include an image in a Markdown cell we would type the following:

    ```
    ![Alternative Text](files/Test Figure)
    ```

    `Alternative Text` in the previous command refers to a text that would be shown in case the image could not be loaded. The text in parenthesis corresponds to the directory of the image, which can also be a webpage. If we want to include an image that is in the same directory as the notebook is, we have to include the directory after `files/`.
  - **Embed a webpage**: Jupyter Notebook allows to embed a webpage within the notebook, indicating the width and height of the frame containing it. In order to do so, we must include the HTML function. An example would be:

    ```
    from IPython.display import HTML
    ```

[f] In this case, the image is considered to be in the same directory as the notebook is, but it is also possible to insert an image from the Internet.

```
HTML('<iframe
src=http://www.wikipedia.org width=800px height=400px >')
```

– **Embed a YouTube video**: embeding videos allows to broaden the spectrum of educational possibilities. Among the different alternatives, YouTube hosts an enormous amount of scientific videos. In order to display one of them, we would need to use the video ID assigned by YouTube to each that specific video:

```
from IPython.display import Youtube
Youtube("videoID")
```

– **Loop**: we will consider the most common loop, the `for` loop. For instance, if we want to print the numbers from 1 to 10 on the screen, we would write

```
for j in range(1,10):
   print(j)
```

– **Conditionals**: we will consider the three most general cases. The simplest conditional is usually written with the `if` command, as is shown in the following example,

```
if j>0:
   print('The variable', j, 'is positive')
```

For a conditional including two cases, we would use `if ... else ...` as follows

```
if j>0:
   print('Variable', j, 'is positive')
else:
   print('Variable', j, 'is negative or zero')
```

Finally, the `while` command, which is used in conditionals within loops, will be used in the following way

```
j=-5
while j<0:
```

```
    print('Variable', j, 'is still negative')
    j++
```

– **Defining functions**: specific functions can be defined by means of the `def` command. For instance, if we want to define the step function we would do

```
def fun(x):
    if(x>0):
      return 1
    else:
      return 0
```

– **Vectors and matrices**: there are different alternatives to create vectors and matrices. Here we will only consider a few examples. In order to create a vector of numbers with a fixed number of elements, say, a 100, with values within an initial and a final value, say, 0 and a 10, the `linspace` function is very useful, and in our example would be used as follows,

```
x=linspace(0,10,100)
```

If instead of the number of elements, we wish to specify the step between different elements of the vector, the function to use is `arange`. For instance, if we want to generate a vector from 0 to 10 with a step of 1, we would write:

```
arange(0,11,1)
```

Notice that the second element of `arange` has to be the final value of our desired range plus the step.

Other useful functions to generate `m` $\times$ `n` matrices are the following:

* If all elements are equal to zero:

  ```
  zeros((m,n))
  ```

* If all elements are equal to one:

  ```
  ones((m,n))
  ```

* If all elements do not have an initially predetermined value:

```
empty((m,n))
```

– **Plots**: a basic plot would be that of the cosine function. This can be achieved by using the following sequence of commands:

```
x=linspace(0,10,100)
y=cos(x)
plot(x,y)
xlabel('X')
ylabel('Y')
title('PlotTest')
```

The last three lines define the X-axis, Y-axis and plot titles, respectively.

As it can be seen in the previous examples, in Jupyter Notebook it is not necessary to separate each line of code by any punctuation mark. However, it is very important to respect the indentation style. This characteristic eases legibility and cleanness of the code, which can be helpful for the students.

Finally, each cell can be executed using the combination of keys Shift + Enter or clicking the play button that appears in the notebook.

Converting code from a different language to Python

It is possible to execute code written in different programming languages using a specific kernel to those languages. However, the ever-increasing applicability of Python to the scientific environment, as well as the vast collection of libraries that are created continuously, make this language an excellent choice of open source software for general purposes. For this reason, we will now analyze how to convert code from a different language to Python, focusing on the most used languages: MATLAB/Octave, Mathematica and C/C++.

- **MATLAB/Octave.** MATLAB is one of the most used programming softwares due to its power and flexibility, as well as to it being easy to learn.

In Table 1 we provide some of the most common commands in MATLAB and their equivalent in Python, using NumPy or SciPy. More information can be found at [16].

Operations	**MATLAB**	**NumPy/SciPy**
First element of the array `a`	`a(1)`	`a[0]`
Last element of the array `a`	`a(end)`	`a[-1]`
Elementwise multiplication of two arrays `a` and `b`	`a.*b`	`a*b`
Size of a matrix `A`	`size(A)`	`shape(A)`
Find indices where `a > 2`	`find(a>2)`	`nonzero(a>2)`
Maximum of a matrix `A`	`max(max(A))`	`A.max()`
Generate a vector with elements from 0 to 5	`0:5`	`arange(6)`
Generate a 3×3 zero matrix	`zeros(3,3)`	`zeros((3,3))`
`a` to the power 4	`a^4`	`a**4`
Fourier transform of `a`	`fft(a)`	`fft.fft(a)`

Tabla 1. *Equivalence of MATLAB y NumPy/SciPy commands.*

- **Mathematica.** Mathematica is primarily a symbolic calculation software, although its newest versions have increased the number of numerical calculation capabilities. Probably, the most similar module in Python would be SymPy, which enables to perform this kind of symbolic operations very easily. Even though its power and library is not at the same level of Mathematica, it is more than enough for educational purposes, as well as for most scientific needs. On the other hand, its constant development guarantees continuous improvements in the future. Another more powerful alternative is SageMath, which allows to use various open software alternatives, such as Maxima, SymPy, NumPy, etc. using a unified syntax. However, SageMath is out of the Python ecosystem so it will not be considered here.

  In Table 2 we provide a list with some equivalences between Mathematica and SymPy. More Information can be found at [17].

Operations	Mathematica	SymPy
Definition of a symbol	Not needed	`x=Symbol('x')`
Fraction decomposition	`Apart[expr]`	`apart(expr,x)`
Fraction combination	`Together[expr]`	`together(expr,x)`
Evaluate an expression	`N[]`	`N()/evalf()`
Limit of a function	`Limit[expr,x->x0]`	`limit(expr,x,x0)`
Differentiation	`D[expr,var]`	`diff(expr,var)`
Power Series expansion	`Series[f,{x,x0,n}]`	`f.series(x,x0,n)`
Indefinite integral	`Integrate[f,x]`	`integrate(f,x)`
Definite integral	`Integrate[f,{x,a,b}]`	`integrate(f,(x,a,b))`
Algebraic equation solver	`Solve[expr,x]`	`solve(expr,x)`

Tabla 2. *Command equivalences between Mathematica and SymPy.*

- **C/C++.**

  C/C++, together with Fortran, is well-known for being extremely fast, especially in executing loops. Since Python is an interpreted language, execution is slower than in a compiled language such as C. In turn, legibility and easiness for programming is obtained, which saves time both for developing code and for fixing bugs. More importantly, from an educational viewpoint, it is much simpler to learn that with C/C++ or Fortran, which limits the use of the latter for generating simple codes to support the explanations of other concepts. However, the fact that C is faster than Python has led to the development of some alternatives to use C in Python. Below we describe two of them:

  - **Weave**: it is part of the SciPy module and allows to write C or C++ code directly within a Python program. In order to use it, a C/C++ compiler must be installed beforehand, apart from Python. Weave can be used in two ways: with the `inline` function or with the `blitz` function. In the first case, C/C++ code is written directly as a chain of characters (within quotation marks) inside the function `inline()`. The first time the

program is executed, Weave takes care of compiling that code (calling the previously installed compiler) and it will generate a library which in successive executions will not need to be compiled anymore, substantially reducing executing time. On the other hand, `blitz()` transforms expressions from NumPy to C/C++ code, performing that exact same function, but in a much faster way. In this case, the main advantage is that the user needs not know how to program in C/C++, but only how to use the expressions from the NumPy module.

– **Cython**: during the last years, Cython has gained popularity and it is a very flexible solution to save time executing Python programs. In contrast to Weave, Cython is a programming language itself, based on Python and encompassing it. It can execute Python code or modify it including definition of variables with type declaration, which allows to fasten execution. Using Cython requires a much more advance knowledge than that described in this Guide, so it will not be considered in detail here. More information can be found at [18].

Course management with Jupyter Notebook

Jupyter Notebook allows to generate interactive documents for students to help them explore and delve into the different aspects considered in a scientific subject. In this sense, they represent a very useful tool for selfstudy, as well as for the teacher to extend the class explanations to other levels of difficulty. However, managing a course completely or partly based in this kind of documents forces to have a server able to offer access to these interactive documents. Moreover, if exercises in this format are included, we would need also a tool to mark and distribute those marks to all students.

In this chapter, we will consider two possibilities for course management using Jupyter Notebook. The first one is an external server, with the great advantage of possessing already a platform and needs no extra work to be launched. The second one is a local solution instead, in which the teacher configures this Jupyter Notebook server. The advantage in this last case is a full control by the teacher, at the expense of more complexity on its configuration.

CoCalc

Cocalc [5], previously known as SageMathCloud, provides a complete platform in the cloud for scientific computing. Among its characteristics, apart from being able to execute Jupyter Notebooks, it includes a complete terminal (Linux), a text editor, a LaTeX editor with instantaneous preview, a task manager and a complete course

manager. The installed software, all of it open source, is extremely vast, covering a large number of programming languages, which can be easily extended by installing other Linux programs in the user space.

The personal space in CoCalc is organized in different projects, each of which has its own files, which can be generated either in the own platform or can be uploaded from the users' computer. On the other hand, it also supplies tutorials about the platform uses, but also about different programming languages.

Regarding the course management capability, the teacher can add students by their e-mail address, assign tasks copying automatically all participants of a course to a specific file, keep track of marking of those tasks and provide a marked copy to all students by clicking a single button. On the other hand, it is free of charge and it offers 3GB of space for free accounts, although there are course packs for different number of students at a given cost, starting from 199$ for 25 students[g]. These packs offer private access to computational resources, allowing for a better user experience.

Another interesting feature from the teacher's viewpoint is that it allows simultaneous collaborative editing of Jupyter Notebooks, in the same way as in Google Documents. This is accomplished thanks to a special version of Jupyter Notebook owned by CoCalc. Hence, this capability not included in Jupyter Notebook by default eases teamwork enormously. Furthermore, it is accompanied by an integrated chat within the platform.

All these options turn CoCalc into a complete and simple solution for course management with Jupyter Notebooks. Moreover, its open source nature has led to the appearance of a local version of the platform, which can be installed in any computer. This local version can be downloaded from its GitHub development webpage [19] as a Docker image and, if the computer is sufficiently powerful, use owned resources for delivering a course based on Jupyter Notebooks.

[g]Offered by CoCalc by April 2018.

JupyterHub

JupyterHub is a tool offered by the Jupyter project, consisting on a multiuser server, each of which can access to their own Dashboard and manage their own Jupyter Notebooks. More specifically, it presents an identification page when accessed via a browser, it manages user IDs and, once access has been allowed, it displays a notebooks' server specific for that user. There they can upload from their laptop any kind of file, download, create and work with Jupyter Notebooks. It also allows to open a console and, if add-ons are installed, activate or disable their functionalities. In this sense, it is worth mentioning that the add-on Nbgrader allows to create, manage and mark exercises (or if a student accesses, allow them to work on pending tasks or send them to their teacher).

For this reasons, JupyterHub has great potential for educational purposes, since it endows each student with a server and the teacher has full control of the installed software. Moreover, it is possible to keep track of all students' accesses.

Installing JupyterHub is not difficult, but a few considerations are in order:

- External access: if JupyterHub will be used to supply students with the course contents, it is necessary to be able to access it externally to ease work at home. This access makes it necessary to protect the students passwords, which can be done by setting JupyterHub to use secure protocol, https.

- Docker: each student or user will access their own server where they have a complete console, being executed in the computer chosen by the teacher. This console can lead to important security risks if there are no limits set in place. Not only that, commands can also executed from a Jupyter Notebook. This is why it is important to isolate JupyterHub from the rest of the system, especially if the server is being used for other tasks as well.

  Fortunately, taking care of this issue is quite straightforward thanks to Docker, which provides containers that are able to isolate individual applications from the rest of the system, with a low consumption of resources. The Jupyter team has provided for its download and direct execution one of these docker

containers (named *image*) para JupyterHub [20], as well as a tutorial to guide during the configuration process.

- Scaling: simultaneous use of different Jupyter Notebooks by many students can represent an important workload for the server. It is necessary to make a previous estimation of that workload, taking into consideration the number of students and the calculations that will have to be done in the proposed tasks. As a reference, authors of this Guide have developed without issues a course delivered to 25 students (without requiring vast amounts of data) using as a server an 8 core computer with 16 GB RAM where JupyterHub was installed. For more demanding cases, more powerful computers would be needed, or alternatively clusters could be used as well.

Thus, JupyterHub provides a basis to offer students a unified environment to access Jupyter Notebook from their own browser, without the need of local installation. However, in order to manage a course, a tool is needed to generate, mark and provide feedback to students. This tool is the Nbgrader extension and is described below.

Nbgrader

Nbgrader is a Jupyter Notebook extension that allows to create, mark and exchange tasks between the teacher and the students. It allows to include tasks based on code (using Python), as well as exercises that can be answered with regular text. For that matter, it includes two tools: on the one hand, an additional toolbar to each cell in Jupyter Notebook to choose if that cell corresponds to the instructions of the assignment, if it will include the student's answer, or if it will be a marking cell with the possibility to include code to mark automatically those exercises. On the other hand, it generates a new tab denominated *Formgrader* in the Jupyter Dashboard. For the teacher, it allows to assign tasks, validate those tests that have undergone automatic marking, retrieve exercises sent by the students, mark them and supply the marked versions back to the students. For these tasks, apart from the web interface

As can be drawn from all the aforementioned features, Nbgrader covers the whole process of course management. For its installation, it is advisable to have Anaconda, since the extension will be installed and activated in a single step. Manual installations require subsequent activation (see the Nbgrader documentation at [21]. In case of choosing Anaconda, its package manager will take care of all this work by typing the command `conda install -c conda-forge nbgrader`.

In order to start using the extension in a course, some previous instructions for its configuration may be needed. However, it is also possible to start automatically with the command `nbgrader quistart course_id` where *course_id* is the name we want to give to our course.

Educational Proposals

General remarks

The European Space for Higher Education (ESHE) Committees have elaborated protocols where great emphasis is put on innovation towards improving classroom teaching (theory and practice) with assignments and attendance to seminars. In particular, interactive seminars allow students to acquire a theoretical knowledge along with a practical one. In the Sciences, it is also crucial to provide students with programming resources in order to endow them with the numerical tools needed for high-complexity scientific problems. This way, time is spent more in the conceptual ideas, rather than in tedious calculations. In this regard, we propose Jupyter Notebook for designing interactive seminars, setting them as a new teaching tool that can be access in class and through the Virtual Campus. It is worth mentioning that this platform for symbolic and numerical calculations allows for a natural coexistence between explanatory text, command lines and plotting. This in turn helps lets students to use this environment without needing to understand all of its technicalities. Different levels of complexity can be considered depending on the teaching needs and the student´s motivation.

- The first level would consist on using explanations supported by plots obtained from simulations.

- The next level would allow students to play with the simulations as if these where black boxes, that is, by tuning different parameters and obtaining results without getting into the computational methods.

- A more advanced level would allow students to modify code to increase the given simulations, letting them acquire new computational abilities.

Jupyter Notebook as an educational tool is particularly relevant to those studies with a scientific or technical component, where numerical, symbolic and statistical calculations are routine. Nonetheless, it can also be relevant to other university studies. In fact, it constitutes a way of getting to know open source software which, apart from the lower costs, we believe it is more in line with the university´s philosophy.

We would like to note that Jupyter Notebook is not circumscribed to teaching purposes. Their modules and functions can also be applied in a more advanced research environment, substituting partially or completely the use of other commercial software for symbolic and numerical calculations. Thus, learning how to use Jupyter Notebook can also be beneficial in the future career of students. At the Python Wiki [22] a great deal of information can be found related to scientific and educational projects that use Python or Jupyter to perform and share calculations. Among those, it is worth mentioning the *Ganga* system, developed at CERN to control job definition and management at the LHC. Another one that deserves mentioning is the computational biology course from the Michigan State University. For these reasons, learning how to use both this tool and Python allows students and teachers to interact with other scientific and educational projects from all over the world.

Benefits for students

We now detail some aspects related to learning that would be boosted by interactive seminars based on Jupyter Notebook and their beneficial outcomes:

- Self-learning is encouraged to different levels of complexity depending on the student´s needs and interests.

- Thanks to possibility to embed of images, HQ video and links to websites, learning is much more appealing.

- Active acquisition of numerical calculation abilities is greatly enhanced.
- Learning LaTeX is fostered no end, a very beneficial outcome due to the massive use of this format for elaborating scientific documents or even for printed material.
- Learning new open source programming languages is a natural outcome. This in turn allows to learn software that is free of charge, flexible and easily shareable and exportable.
- The Virtual Campus, a platform that more and more students are getting used to, is enriched with new possibilities.
- Marking and automatic marking can be done remotely in a more flexible manner, tailored to the particular needs of each student. This frees classroom time for the teacher to delve more in depth into the explanations.
- Usage by a large number of students is not a problem since this educational projects are available without needing to install commercial software. Moreover, they can be accessed through different devices, such as smartphones or tablets.

Examples for educational proposals

Recently we have collected a dossier of Jupyter Notebook-based educational proposals, that have been developed within several innovative educational projects at the Complutense University of Madrid since 2012, visit the website [23]. These proposals focus on some of the most relevant concepts of lectures of *Physical Optics* and *Biomedical Optics* of the Bachelor's Degree in Optics and Optometry of the Faculty of Optics and Optometry, and also of the lecture *Solid State Physics* of the Bachelor's Degree in Physics and the Bachelor's Degree in Materials Engineering, all of them offered at the Complutense University of Madrid. Although these notebooks do not cover all the scientific concepts of every presented topic, they present a variety of the educational capabilities of the project Jupyter. Needless to say that our proposals

are far from a closed lecture tool, but indeed as any other teaching resource must be updated and improved continuously.

Conclusions

After our evaluation of interactive documents for educational purposes based on Jupyter Notebooks, we have shown this platform offers a rich environment to facilitate the learning process by naturally integrating numerical simulations written in Python language. Thus, the student acquires new competences on numerical calculation while he gets a deeper understanding of the theoretical concepts. The remote access and the absence of license costs clearly supports an extensive use of this educational tool for the comprehension of scientific and/or mathematical concepts in Science or Technical Bachelors.

Jupyter Notebooks can be used as improved theoretical explanations of concepts presented in lectures or exercises to be evaluated by the professor or by the student itself. They can even be established as a platform to develop dynamical lecture notes (*Wiki*) to be continously improved by successive students.

Bibliografía

[1] https://github.com/jupyter/jupyter/wiki/Jupyter-kernels

[2] https://www.anaconda.com/download/

[3] https://github.com

[4] https://jupyter.nbviewer.org

[5] https://cocalc.com

[6] https://gryd.us

[7] https://research.google.colab.com

[8] https://mybinder.org

[9] http://johnmacfarlane.net/pandoc/installing.html

[10] http://ipython.org/ipython-doc/stable/interactive/nbconvert.html#nbconvert

[11] http://www.slideviper.oquanta.info/tutorial/slideshow_tutorial_slides.html?transition=no

[12] https://jupyter-notebook.readthedocs.io/en/stable/notebook.html

[13] https://www.reddit.com/r/JupyterNotebooks/

[14] https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks

[15] https://github.com/sagemathinc/cocalc

[16] http://wiki.scipy.org/NumPy_for_Matlab_Users

[17] https://github.com/sympy/sympy/wiki/SymPy-vs.-Mathematica

[18] http://cython.org

[19] https://github.com/sagemathinc/cocalc-docker

[20] https://hub.docker.com/r/jupyterhub/jupyterhub/

[21] http://nbgrader.readthedocs.io/en/latest/user_guide/installation.html

[22] https://wiki.python.org/moin/

[23] https://github.com/ecabreragranado/JupyterUCM