Universidad Complutense de Madrid

MANUAL DE USO DE JUPYTER NOTEBOOK

PARA APLICACIONES DOCENTES

Autores:
Eduardo Cabrera Granado
Elena Díaz García

Índice general

Introducción

El proyecto de código abierto Jupyter pretende generar una plataforma computacional que permita utilizar diferentes lenguajes (su nombre proviene de unir los 3 lenguajes quizás más populares en cálculo científico: Julia, Python y R). El pilar en el que se sustenta es la interfaz web Jupyter Notebook, que aglutina la ejecución de código, inclusión de texto junto a ecuaciones en LaTeX (vía MathJax), video, y todo lo que se pueda visualizar con un navegador. Su funcionalidad, por otro lado, puede ser extendida gracias a herramientas como *nbconvert* , encargada de gestionar la conversión a formatos como LaTeX, PDF o presentaciones web basadas en Reveal.js, o *ipywidgets* que proporciona widgets interactivos dentro del documento. Su desarrollo ha sido patrocinado por compañías como Micro- soft o Google así como impulsado por el proyecto de infraestructura europea Horizonte 2020 denominado OpenDreamKit. Su uso en entornos docentes ha crecido en popularidad debido a su gran flexibilidad, fácil acceso a través del navegador y las posibilidades añadidas de separar el propio documento del núcleo de cálculo en el lenguaje escogido. Esta arquitectura permite abandonar la instalación local. El profesor puede hacer uso de un servidor externo (o su propio ordenador) para proporcionar a los estudiantes lecciones, presentaciones, o tareas interactivas. Esta característica permite eliminar una de las grandes barreras de acceso que se presentan al incluir un nuevo tipo de documento o programa, como es su instalación en el equipo personal del estudiante. Utilizando este tipo de documentos se pueden generar lecciones o material de apoyo al estudio individual ricos en contenido, así como material de trabajo colaborativo que incluya elementos de cálculo numérico. Dado que resulta inevitable que un estudiante del área de ciencias necesite trabajar con algún tipo de programa de cálculo a

lo largo de su carrera universitaria así como en su futuro laboral, la inclusión de este tipo de elementos permite al alumno comenzar a familiarizarse con ellos, adquiriendo por tanto una competencia transversal de gran valor para su formación. Además, proporciona otro nivel más de profundización en los contenidos de la asignatura, que podrá ser explorado por aquellos estudiantes con una mayor motivación, aunque no se requiera para el desarrollo normal del curso.

A lo largo del presente manual veremos las características generales de la interfaz Jupyter Notebook, su instalación y uso para proporcionar lecciones y ejercicios útiles en estudios científicos. A su vez, se describirán varias opciones para proporcionar un servidor de este tipo de documentos accesible por los estudiantes a través del ordenador, así como añadidos específicos para docencia como es la gestión de un curso íntegramente realizado en Jupyter Notebooks.

Jupyter Notebook

Jupyter Notebook es una interfaz web de código abierto que permite la inclusión de texto, vídeo, audio, imágenes así como la ejecución de código a través del navegador en múltiples lenguajes. Esta ejecución se realiza mediante la comunicación con un núcleo (Kernel) de cálculo. Aunque en principio, el equipo de desarrollo de Jupyter Notebook incluye por defecto únicamente el núcleo de cálculo Python, el carácter abierto del proyecto ha permitido aumentar el número de núcleos disponibles [1], incluyendo, por ejemplo Octave, Julia, R, Haskell, Ruby, C/C++, Fortran, Java, SageMath, Scala, o también Matlab y Mathematica. Esta interfaz, agnóstica del lenguaje (de ahí su nombre al unir 3 de los lenguajes de programación de código abierto más utilizados en el ámbito científico: Ju-lia, Py-thon y R), puede suponer por tanto una estandarización para mostrar el contenido de cursos científicos, sin encontrarse limitado a la adopción de un único lenguaje.

Esta versatilidad ha permitido una adopción creciente tanto en el ámbito docente como investigador. En este último campo, la inclusión de Jupyter Notebooks con cálculos, datos y figuras adicionales a artículos aparecen cada vez más en las revistas científicas. Por ejemplo, los datos y cálculos facilitados en este formato por el equipo de LIGO, dedicado al descubrimiento de las ondas gravitacionales. En lo referente a su uso docente cabe destacar el curso *AeroPython* de la Universidad George Washington e impartido por la profesora Lorena Barba enteramente utilizando Jupyter Notebooks.

Orígenes

Jupyter Notebook nace como una evolución de la interfaz Ipython Notebook, básicamente con las mismas funcionalidades pero añadiendo la posibilidad de ejecutar código en múltiples lenguajes. Por otro lado, la interfaz web, común tanto a Jupyter Notebook como a Ipython Notebook, supone el desarrollo en software libre de un "cuaderno computacional" al mismo estilo que los cuadernos empleados en el programa Mathematica o en Maple y cuyo primer intento fue realizado por William Stein para el programa de cálculo Sage (Sage Notebooks). Para este desarrollo, llevado a cabo por Fernando Pérez y Robert Kern, en la Universidad de Berkeley (California, EEUU), se basaron en principio en el lenguaje de programación Python, dada su gran versatilidad, amplia adopción por la comunidad científica y su carácter libre. Aunque en un principio el desarrollo no se centró en una interfaz web, sino en proporcionar un intérprete de comandos interactivo para el lenguaje Python (denominado IPython), a partir de 2010 el trabajo del equipo de desarrollo de IPython permite la aparición de la interfaz Jupyter Notebook.

Desde entonces, el desarrollo de estos "cuadernos computacionales", así como su uso, no ha parado de crecer. Por ejemplo, la plataforma de desarrollo de proyectos libres más importante actualmente, GitHub, permite la visualización de Jupyter Notebooks directamente en su página web, alojando más de un millón de este tipo de documentos, de libre acceso, en sus servidores.

Descripción de la interfaz

Al ejecutar Jupyter Notebook en el ordenador, aparecerá, en nuestro navegador una primera página denominada Jupyter Dashboard. En ella podremos ver la lista de archivos disponibles en la carpeta en donde hayamos ejecutado nuestro programa. Además, en caso de tener instalados añadidos (extensiones) a nuestro Jupyter Notebook, como por ejemplo el gestor de cursos y corrección automática Nbgrader, también nos aparecerá en esta página de acceso. Otro elemento importante es el botón, en la esquina superior derecha, para subir archivos al servidor. En caso de

ejecutarse Jupyter Notebook localmente, este botón no presenta una gran funcionalidad, pero sí para el caso en que accedamos a estos documentos desde otro ordenador y a través del navegador.

Para generar un primer Jupyter Notebook, pincharemos en *New* y elegiremos el núcleo de cálculo entre aquellos que tengamos instalados. Por defecto únicamente tendremos Python. Un ejemplo de un documento tipo Jupyter Notebook puede visualizarse en la Figura 1.

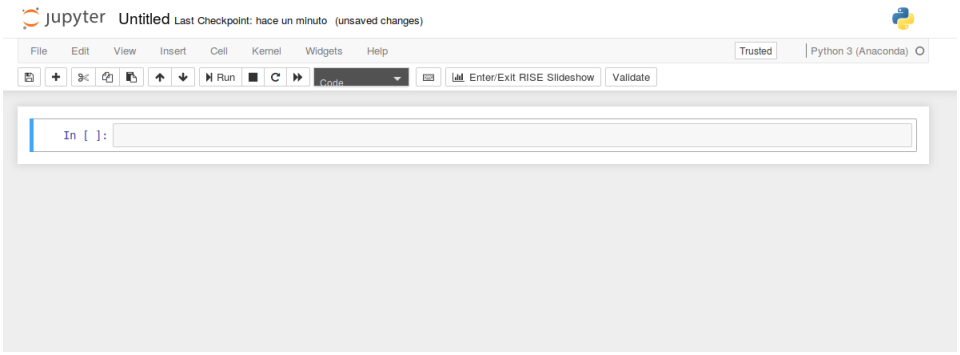

Figura 1. *Ejemplo de un documento Jupyter Notebook.*

En la parte superior del documento disponemos de los distintos menús para controlar tanto la edición, como para descargar el documento en varios formatos (por ejemplo, PDF) así como insertar o borrar partes del documento. También incluye la elección y el control del núcleo de cálculo (pudiendo pararlo si así se desea) y un botón de ayuda que cubre gran parte de las características de Jupyter Notebook.

El documento en sí se divide en distintas celdas, el comportamiento de las cuales puede seleccionarse entre distintas opciones: título (de diversos tamaños), texto (plano o enriquecido utilizando el lenguaje Markdown) o código. Aparte de estas opciones, las librerías específicas permiten importar imágenes o vídeos, incluir enlaces a recursos web o incluso incrustar una página web completa dentro del documento.

Para insertar texto con formato, la opción elegida por Jupyter Notebook es utilizar el lenguaje Markdown. De este modo, por ejemplo, se pueden incluir listas, texto en negrita o cursiva, tablas o imágenes. La cantidad de tutoriales en la red sobre este lenguaje es inmenso por lo que nos centraremos en indicar cómo se utiliza

para las opciones de formato más utilizadas:

- Texto en negrita/cursiva: El texto en negrita se indica entre dos pares de asteriscos. De este modo `**palabra**` aparecerá como **palabra**. Por otro lado, el texto en cursiva se indica entre dos asteriscos simples, es decir `*palabra*` aparecerá como *palabra*.

- Listas: Las listas en Markdown se realizan indicando un asterisco o un número seguido de un punto si se desean listas numeradas. Markdown organiza automáticamente los items asignándoles el número correcto.

- Inclusión de imágenes: La sintaxis para incluir imágenes en Markdown es `![nombre alternativo](dirección de la imagen)` en donde el nombre alternativo aparecerá en caso de que no se pueda cargar la imágen y la dirección puede referirse a una imagen local o un enlace en Internet.

- Inclusión de código HTML: El lenguaje Markdown es un subconjunto del lenguaje HTML y en donde se necesite un mayor control del formato, se puede incluir directamente en HTML.

- Enlaces: Las celdas de texto pueden contener enlaces, tanto a otras partes del documento, como a páginas en internet u otros archivos locales. Su sintaxis es `[texto](dirección del enlace)`.

- Fórmulas matemáticas: Gracias al uso de MathJax, se puede incluir código en LaTeX para mostrar todo tipo de fórmulas y expresiones matemáticas. Las fórmulas dentro de una línea de texto se escriben entre símbolos de dolar (`$...$`) mientras que las expresiones separadas del texto utilizan símbolos de dolar dobles (`$$...$$`).

Estas posibilidades permiten utilizar las celdas de texto de manera versátil documentando el código de otras celdas o, por ejemplo, explicando detalladamente algún concepto teórico, del que después se puede presentar un ejercicio.

La creación de una nueva celda provoca la aparición automática de una celda de código. El lenguaje que se utiliza viene determinado por la elección del núcleo

para todo el documento. Actualmente, mezclar dos lenguajes de programación en el mismo documento no está soportado.

Instalación

Uno de los requisitos que el sistema donde se instale Jupyter Notebook ha de poseer es disponer de Python. Aunque los documentos permiten la ejecución en otros lenguajes de programación, algunas de las funciones de Jupyter Notebook están implementadas en este lenguaje. Dada su enorme popularidad en el ámbito científico, *suites* de cálculo científico ofreciendo gran cantidad de módulos de Python han aparecido en los últimos años. Dentro de ellas, quizás la más conocida es Anaconda, de la empresa Continuum Analytics. Este pack de módulos de Python dispone a su vez de un gestor de paquetes y ha facilitado enormemente la instalación y gestión de librerías de Python relacionadas con aplicaciones científicas.

Uno de los módulos por defecto que Anaconda instala es Jupyter Notebook por lo que la instalación recomendada, independientemente del sistema operativo, es instalar dicha distribución de Python científico. Para ello, la página de descarga de Anaconda [2] permite obtener gratuitamente este producto. Siguiendo los pasos de instalación, podemos tener disponible Jupyter Notebook, el cual podemos ejecutar escribiendo `jupyter notebook` en una consola.

Plataformas online para su uso sin instalación

Aunque la instalación de Jupyter Notebook se ha simplificado enormemente en los últimos años, una de las características esenciales en su uso docente es la separación de esta interfaz del núcleo de cálculo, lo cual hace que el estudiante pueda consultar, mediante un navegador, los documentos interactivos en este formato que se ejecutan en otro ordenador, bien sea el del profesor o un servidor facilitado por un servicio en la nube. A continuación se enumeran algunas de estas alternativas. Nos centraremos en aquellas que no se restrinjan a mostrar una versión estática del documento, sino que permitan la ejecución de código en ellos. Ejemplos del primer caso, podrían ser

GitHub [3] o NbViewer [4].

1. **CoCalc**: Este entorno de computación online [5] proporciona varias herramientas únicas destinadas a un uso docente: permite la edición colaborativa de documentos Jupyter Notebook, lo que multiplica las posibilidades para realizar trabajos en grupo y dispone de un gestor de cursos y tareas, lo que simplifica la tarea del profesor para proporcionar tanto material de apoyo, como para calificar tareas y distribuir una retroalimentación a los estudiantes. Además, es gratuito y por tanto los estudiantes siguen manteniendo su cuenta al finalizar el curso, lo que permite el acceso a las simulaciones y explicaciones que pueden resultar de utilidad en otras asignaturas de cursos posteriores. También dispone de paquetes de cursos de pago en donde se ofrece acceso a servidores más estables que en las cuentas gratuitas y más recursos computacionales.

2. **Gryd**: Este servicio [6] ofrece cuentas gratuitas para estudiantes así como un paquete académico para la gestión de cursos utilizando nbgrader, aunque su precio no se encuentra publicado actualmente.

3. **Google Colab**: Aunque actualmente su acceso se encuentra restringido [7] Google Colab proporciona un servidor de Jupyter Notebooks donde se incluye la opción de editar colaborativamente un mismo documento, al estilo de Google Documents. Esta funcionalidad lo hace especialmente atractivo como servicio destinado a trabajos en grupo durante el desarrollo de un curso universitario.

4. **MyBinder**: Este servicio [8] permite la creación de un servidor temporal para ejecutar los documentos Jupyter Notebooks alojados en un repositorio en GitHub. Es completamente gratuito.

5. **JupyterHub**: Más que un servicio online, JupyterHub es una de herramienta más dentro del ecosistema Jupyter, enfocado a proporcionar un servidor multiusuario para poder ser utilizado en clase o por un equipo de investigación, por ejemplo. Sobre esta herramienta dedicamos un capítulo aparte en este mismo manual por lo que su discusión se pospone a ese punto.

Exportar a otros formatos

Aunque los documentos generados por Jupyter Notebook son fácilmente accesibles a través de un navegador, en ocasiones es necesario o útil disponer de otros formatos, por ejemplo, utilizar un archivo Python ejecutable. Con el fin de facilitar la conversión de los *notebooks*, Jupyter dispone desde la versión 1.0 (versión actual a día de hoy) de la herramienta **nbconvert**, que permite exportar los documentos hechos en Jupyter Notebook a otros formatos. Concretamente, la herramienta incluida en la instalación de Jupyter Notebook denominada **nbconvert** proporciona la conversión a documentos estáticos (es decir, las celdas no pueden ejecutarse), en formato HTML, LaTeX, Markdown, reStructuredText, script de Python ejecutable y, finalmente, en formato presentación. El uso de esta herramienta requiere la instalación de un nuevo paquete denominado *Pandoc*, que puede instalarse a través del gestor de paquetes de las distintas distribuciones Linux o bien, a través de su página web [9].

El funcionamiento de esta herramienta puede realizarse mediante una terminal o consola, o bien directamente desde el documento, a través del menú `File ->Download as...`, aunque para explotar todas sus capacidades indicamos a continuación la sintaxis para su uso en una terminal. Esta es,

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

donde `FORMAT` es el formato que se desea obtener. Por ejemplo, si queremos convertir nuestro documento a un archivo en LaTeX, escribiríamos:

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

Una vez generado el archivo .tex podemos convertirlo a pdf mediante, por ejemplo, pdflatex o cualquier editor de LaTeX. Si queremos realizar realizar la conversión a formato pdf directamente, podemos añadirlo a la anterior instrucción, quedando,

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

También es posible modificar las plantillas de nuestro archivo LaTeX generado a formato libro por ejemplo añadiendo `--template book`. Es necesario hacer notar que el formato presentación no genera un archivo PowerPoint o similar, sino una presentación Reveal.js en HTML, que requiere para su visualización que el ordenador muestre la presentación en el navegador. Este requisito se cumple si añadimos a la línea anterior la opción `--post serve`. Este tipo de presentaciones, aunque más difíciles de mostrar por la necesidad de que el ordenador actúe como servidor de esa página en HTML, dan un mayor número de opciones que las presentaciones tradicionales y son más sencillas de compartir a través de diferentes sistemas operativos y plataformas.

Para más información, se pueden consultar los siguientes enlaces [10, 11].

Recursos en la red

La cantidad de recursos en la red dedicada a Jupyter Notebooks no para de crecer y de modificarse con el tiempo. Es por ello que la siguiente lista ha de tomarse como una muestra, para nada completa, y válida únicamente en las fechas en las que se ha generado este documento. Sin embargo, y dada la importancia de los sitios web a los que se refieren los siguientes enlaces, es esperable que, aunque nuevos sitios de referencia puedan surgir, los resaltados a continuación puedan seguir siendo recursos útiles en el futuro.

1. Documentación de Jupyter Notebook [12]
2. Subforo en Reddit dedicado a Jupyter Notebooks [13]
3. Galería con una selección de Jupyter Notebooks [14]
4. Página de desarrollo de CoCalc [15]

Dado que el lenguaje principal de los documentos mostrados en el Anexo ha sido Python, a continuación se muestran algunas de sus características, especialmente comparando con otros programas de cálculo como Matlab, también ampliamente utilizado en los estudios universitarios de carácter científico.

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[a] as the computational kernel.

[a]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
    ```
    [b]

`print` command.

[b] 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[c]

    ```
    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
    ```

[c] 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].

Gestión de cursos

Jupyter Notebook permite generar documentos interactivos para el alumno que le ayuden a explorar y profundizar en los distintos aspectos tratados en una asignatura de ámbito científico. En este sentido, suponen una herramienta muy útil de apoyo para el trabajo individual del estudiante y una ayuda para el profesor para extender las explicaciones en clase a otros niveles. Sin embargo, la gestión de un curso basado totalmente o parcialmente parte en este tipo de documentos obliga a disponer de un servidor capaz de ofrecer el acceso a estos cuadernos interactivos. Además, si incluimos ejercicios en este formato, requeriremos de alguna herramienta para corregirlos y para distribuir dicha corrección entre todos los estudiantes.

En este capítulo abordamos dos posibilidades para la gestión de un curso utilizando Jupyter Notebooks. En primer lugar, un servicio externo, con la gran ventaja de disponer de una plataforma con nulo trabajo para ser puesta en marcha. En segundo lugar, una solución local, en la que el profesor configura ese servidor de Jupyter Notebooks. La ventaja en ese último caso es el control completo de todo el proceso por parte del profesor, aunque con el coste de una mayor complejidad para su configuración.

CoCalc

Cocalc [5] , anteriormente conocido como SageMathCloud, proporciona una plataforma completa de trabajo en la nube para la computación científica. Dentro de sus características, destaca, además de poder ejecutar Jupyter Notebooks, disponer de una terminal completa (Linux), un editor de texto, editor de LaTeX con previsua-

lización instantánea, gestor de tareas y un completo gestor de cursos. El software instalado, todo ello basado en herramientas libres, es muy amplio, cubriendo multitud de lenguajes de programación, pudiendo extenderse fácilmente instalando en el espacio del usuario otros programas en Linux.

El espacio personal en CoCalc se organiza en diferentes proyectos, cada uno de los cuales con sus archivos, los cuales pueden ser generados en la propia plataforma o bien subidos desde el ordenador del usuario. Por otro lado, también dispone de tutoriales de distntos aspectos tanto de uso de la plataforma, como de programación en distintos lenguajes.

En lo referente a la gestión de cursos, el profesor puede añadir estudiantes por su dirección de correo, asignar tareas copiando automáticamente a todos los integrantes del curso una carpeta determinada, llevar un registro de la corrección de dichas tareas y devolver una copia corregida a todos los estudiantes pulsando un único botón. Por otro lado, su acceso es gratuito, ofreciendo un espacio de 3GB para estas cuentas, aunque existen paquetes de cursos para distinto número de alumnos con un cierto coste, comenzando por 199$ para 25 estudiantes [d]. Estos paquetes ofrecen acceso reservado a recursos computacionales logrando por tanto una mejor experiencia de uso.

Otra de las características más útiles de CoCalc desde el punto de vista docente es que permite la edición colaborativa simultánea de los documentos Jupyter Notebook al estilo de Google Documents, gracias a una versión propia de la interfaz web de Jupyter Notebook. Este añadido, no presente en la versión por defecto de Jupyter Notebook, facilita enórmemente el trabajo en grupo sobre una tarea, y se complementa con un chat integrado en la propia plataforma.

Todas estas opciones hacen de CoCalc una solución completa y sencilla de usar para la integración de Jupyter Notebooks en un curso académico. Además, su carácter completamente libre ha propiciado la aparición de una versión local de la plataforma completa, pudiendo ser instalada en un ordenador cualquiera. Esta versión local, a modo de imagen Docker, puede ser descargada desde su página de desarrollo en GitHub [19] y permite, si se dispone de un ordenador suficientemente

[d] Ofrecido por CoCalc a fecha de Abril de 2018

potente, utilizar recursos propios para la impartición de un curso basado en Jupyter Notebooks.

JupyterHub

JupyterHub es una de las herramientas ofrecidas por el proyecto Jupyter, y consiste en un servidor multiusuario cada uno de los cuales accede a su propio Dashboard y gestiona sus propios Jupyter Notebooks. Más detalladamente, JupyterHub se encarga de presentar una página de identificación a quien accede con el navegador a la página del servicio, gestiona la identificación del usuario y una vez el acceso ha sido permitido, presenta un servidor de notebooks específico para ese usuario, donde puede copiar desde su ordenador cualquier tipo de archivo, descargarlos, crear nuevos Jupyter Notebooks y, por supuesto, trabajar con ellos. También permite abrir una terminal del sistema y, en el caso de tener instaladas extensiones, activarlas y desactivarlas o hacer uso de sus funcionalidades. En este sentido, y dentro del contexto de gestión de un curso académico, cabe destacar la extensión denominada Nbgrader la cual permite a un profesor crear, gestionar y evaluar ejercicios para los estudiantes (o en el caso en que sea un estudiante quien acceda, trabajar con las tareas pendientes y enviarlas al profesor).

Por ello JupyterHub posee un gran potencial para su uso docente, al facilitar un servidor de notebooks para cada uno de los estudiantes de un curso, y en donde el profesor tiene el control absoluto del software instalado y puede monitorizar el acceso de cada estudiante.

La instalación y configuración de JupyterHub no es difícil, pero es necesario tener en cuenta algunas consideraciones:

- Acceso externo: Evidentemente, si JupyterHub se utiliza para poner a disposición de los estudiantes material de un curso, es necesario que se pueda acceder desde el exterior para facilitar el trabajo fuera de clase. Este acceso hace necesaria cierta seguridad para proteger las contraseñas de los estudiantes, como es configurar que JupyterHub utilice una conexión segura mediante el uso del protocolo https.

- Docker : Cada estudiante o usuario accede a un servidor propio en donde se dispone de una terminal completa, ejecutándose en el ordenador elegido por el profesor. Esta terminal puede suponer un riesgo de seguridad importante en caso de no poner límites a su funcionamiento. No solo eso, la ejecución de comandos del sistema puede ser llevada a cabo a su vez desde un Jupyter Notebook. Es por ello que, especialmente cuando el servidor es utilizado por parte del profesor para otras tareas, se hace necesario encapsular de algún modo el funcionamiento de JupyterHub del resto del sistema.

  Afortunadamente, realizar este proceso es relativamente sencillo gracias a Docker, el cual proporciona contenedores, con un consumo bajo de recursos, capaces de aislar del resto del sistema aplicaciones individuales. El equipo del proyecto Jupyter tiene disponible para su descarga y ejecución directa uno de estos contenedores docker (denominada *imagen*) para JupyterHub [20] así como un tutorial con todos los pasos para configurarlo.

- Escalamiento: El uso simultáneo de diferentes Jupyter Notebooks por parte de muchos alumnos puede imponer una gran carga de trabajo al servidor elegido para desarrollar el curso. Es necesario por tanto realizar una estimación previa de dicha carga, en función del número de alumnos y de los cálculos que requieran las tareas propuestas por parte del profesor. Como referencia, los autores de este manual han desarrollado sin problemas un curso con acceso simultáneo de 25 estudiantes a Jupyter Notebooks (sin requerir un manejo masivo de datos) utilizando como servidor un ordenador con 8 núcleos y 16 GB de RAM donde se instaló JupyterHub. Mayores necesidades impondrían el uso de ordenadores más potentes o bien de clusters.

JupyterHub proporciona por tanto la base para poder ofrecer a los estudiantes un entorno unificado de acceso a Jupyter Notebooks a través de su navegador, sin requerir una instalación local. Sin embargo, para gestionar un curso, es necesario además una herramienta para generar ejercicios, poderlos corregir y proporcionar una retroalimentación a los estudiantes. Esta herramienta es la extensión Nbgrader que se describe a continuación.

Nbgrader

Nbgrader es una extensión a Jupyter Notebook que gestiona la generación, corrección e intercambio de tareas entre el profesor y los estudiantes. Permite incluir tareas tanto basadas en código (utilizando Python) como ejercicios con respuestas libres de texto. Para ello dispone de dos herramientas: por un lado, incluye una barra de opciones adicional a cada celda de Jupyter Notebook para elegir si dicha celda es el enunciado de un ejercicio, si va a incluir la respuesta del estudiante, o bien si es una celda de corrección con la posibilidad de incluir códigos que permitan la autocorrección de esos ejercicios. Por otro lado, genera una nueva pestaña denominada *Formgrader* en la página de acceso a Jupyter Notebook (Jupyter Dashboard) con una doble funcionalidad. Para el profesor, permite la asignación de tareas, validar los tests en caso de disponer de celdas de autocorrección, recoger los ejercicios enviados por los estudiantes, corregirlos y distribuir las versiones corregidas de nuevo a los alumnos. Para estas tareas, aparte de la interfaz web

Como se puede observar por las anteriores características descritas, Nbgrader cubre todo el proceso de gestión de de un curso. Para su instalación, es recomendable de nuevo utilizar la distribución de Python científico Anaconda pues la extensión se instalará y se activará en un único paso. Instalaciones manuales requerirán realizar la activación posteriormente además de la instalación (vease la documentación de Nbgrader en [21]. En el caso, de elegir Anaconda, su gestor de paquetes de Python realiza todo el trabajo por nosotros con el comando `conda install -c conda-forge nbgrader`.

Para comenzar a utilizar la extensión en un curso, pueden ser necesarias ciertas instrucciones previas para su configuración. Sin embargo, también es posible realizar un comienzo automático con el comando `nbgrader quistart course_id` donde *course_id* es el nombre que queremos asignar a nuestro curso.

Propuestas Docentes

Consideraciones generales

Las Comisiones que han elaborado los protocolos del Nuevo Espacio Europeo de Educación Superior (EEES) destacan la importancia de la innovación docente para la mejora de las clases presenciales teóricas y prácticas, complementadas con una fuerte participación en seminarios y ejercicios realizados fuera del aula. En particular, el desarrollo de seminarios interactivos permite al alumno adquirir un conocimiento teórico a la vez que realiza actividades prácticas al respecto. En el área de ciencias es importante, además, dotar a los estudiantes de recursos de programación para favorecer el aprendizaje de técnicas numéricas imprescindibles en problemas científicos de alta complejidad, optimizando el tiempo que se emplea en la adquisición de conocimientos y minimizado el empleado en los cálculos que demuestran los aspectos teóricos fundamentales. En este marco, hemos propuesto Jupyter Notebook para el diseño de seminarios interactivos, constituyendo una nueva herramienta docente utilizable en el aula y puesta a disposición a través del Campus Virtual. Destacamos que en esta plataforma de cálculo simbólico y numérico, las líneas de texto explicativo, las líneas de comandos y la representación gráfica coexisten en el mismo entorno, facilitando así que los estudiantes puedan manejarlo aún sin conocer en detalle todas sus funcionalidades. El uso de estos seminarios puede realizarse a diferentes niveles de profundidad según las necesidades docentes y la motivación del alumno.

- El primer nivel consistirá en el uso de las explicaciones, apoyadas con figuras obtenidas de las simulaciones.

- En un siguiente nivel se plantearán ejercicios en los que el estudiante utilizará las simulaciones como una *caja negra*, es decir, cambiando los parámetros y obteniendo los resultados sin conocer el método de cálculo.

- En el nivel más avanzado los estudiantes podrán modificar los códigos para ampliar las simulaciones mostradas, adquiriendo así nuevas competencias en el cálculo numérico.

Jupyter Notebook como herramienta docente, aunque puede ser de utilidad en cualquier disciplina universitaria, es de especial relevancia en titulaciones de naturaleza científica, técnica o relativas a ciencias de la salud, donde usualmente se requiere una potente herramienta de cálculo numérico, simbólico o estadístico. Sin embargo, no es menos importante destacar otra de las ventajas principales para la comunidad universitaria, y es que establece una pasarela hacia el conocimiento de herramientas de software libre, cuya filosofía (y no solo por su menor coste económico) consideramos más afín a la enseñanza universitaria.

Nos gustaría destacar que Jupyter Notebook no circunscribe su aplicación únicamente a las tareas docentes. Los módulos y funciones que se utilizan son también válidos para el desarrollo de programas más avanzados orientados a la investigación, pudiendo sustituir en gran parte el uso de otros programas comerciales de cálculo numérico y simbólico. Así pues, su aprendizaje resulta útil para la adquisición de competencias y capacidades para el ejercicio profesional futuro. En la web [22] se puede encontrar información sobre la gran cantidad de proyectos científicos y docentes que usan Python y la herramienta Jupyter Notebook para realizar y compartir sus cálculos. Entre ellos, cabe destacar el sistema *Ganga*, desarrollado en el CERN para el control de trabajos en red de los experimentos en el LHC, o en aplicaciones docentes como el curso de biología computacional en la Universidad de Michigan State, basado en Jupyter Notebook. Por todo esto, el aprendizaje de esta herramienta y la base del lenguaje de programación Python permite a los alumnos y profesores interaccionar con otros proyectos científicos y docentes muy activos en otras partes del mundo.

Beneficios para los estudiantes

Detallamos ahora los aspectos del apredizaje que son potenciados por los seminarios interactivos desarrollados con Jupyter Notebook y los beneficios que este tipo de seminarios conllevan para los estudiantes:

- Favorecen el aprendizaje autónomo de los estudiantes a diferentes niveles de complejidad según los intereses y las necesidades de cada alumno.
- Permiten un aprendizaje más atractivo, ya que estos seminarios digitales pueden incluir imágenes y vídeos de alta calidad y enlaces a otras webs de interés.
- Facilitan el proceso activo de adquisición de competencias en el cálculo numérico.
- Promueven el aprendizaje de código de texto en formato LaTeX que se utiliza para la elaboración de documentos científicos de amplia difusión, incluso en imprenta.
- Promueven el conocimiento y aprendizaje de nuevos lenguajes de programación basados en software libre que no conllevan coste económico y que utilizan formatos estándar. Es decir, pueden ser fácilmente utilizados por otras personas y exportados con mayor facilidad.
- Ofrecen nuevas posibilidades de uso del Campus Virtual, con el que los estudiantes están cada vez más familiarizados.
- Permiten realizar tareas de evaluación y autoevaluación en remoto de una forma más flexible y adecuada a las necesidades particulares de cada alumno. Esto liberará parte del tiempo de clase presencial que podrá ser usado para ampliar las explicaciones del docente.
- Facilitan su utilización por parte de un mayor número de alumnos, ya que están disponibles sin necesidad de instalar ningún software comercial y se puede acceder a ellos a través de dispositivos electrónicos más dinámicos, como los teléfonos inteligentes o tabletas.

Ejemplos de propuestas docentes

Recientemente hemos recolectado un dosier de los documentos docentes basados en Jupyter Notebook elaborados en diferentes proyectos de innovación educativa desarrollados en la Universidad Complutense desde el año 2012, visitar la web [23]. Estos documentos desarrollan algunos de los conceptos más relevantes de las asignaturas de Óptica Física y Óptica Biomédica impartidas en el Grado de Óptica y Optometría de la Facultad de Óptica y Optometría, y de la asignatura de Física de Estado Sólido del Grado de Física y del Grado de Ingeniería de Materiales impartidos en la Facultad de Ciencias Físicas, todos adscritos a la Universidad Complutense de Madrid. Estos documentos, sin pretender cubrir todos los conceptos explicados en los temas de referencia, presentan una pequeña muestra de las capacidades de Jupyter Notebook dentro de las tareas docentes. Además no deberían entenderse como documentos cerrados, sino, como cualquier material docente, en constante proceso de actualización y mejora para futuros cursos.

Conclusiones

Tras la evaluación del uso de documentos interactivos basados en la herramienta Jupyter Notebook para aplicaciones docentes, hemos mostrado que esta plataforma ofrece un entorno rico en contenidos que facilitan el aprendizaje, integrando de manera natural simulaciones realizadas en Python. Así, el estudiante adquiere competencias en cálculo numérico a la vez que profundiza en los conceptos explicados en el aula. El acceso remoto y la ausencia de coste asociado a su licencia facilitan sin lugar a dudas la implantación de estas herramientas en los contenidos curriculares de los Grados en Ciencias.

Los documentos interactivos se pueden utilizar como texto explicativo y ampliado de los conceptos presentados en el aula, como ejercicio que evalúe el profesor o como ejercicio de autoevaluación del alumno. Dependiendo de la respuesta del alumnado, podría incluso involucrarse a los estudiantes en la elaboración de apuntes dinámicos (a la manera de una *Wiki*) que puedan ser mejoras en cursos sucesivos.

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