Scientific Visualization: Python + Matplotlib

Nicolas P. Rougier

SCIENTIFIC PYTHON VOLUME II

SCIENTIFIC VISUALIZATION

PYTHON & MATPLOTLIB

NICOLAS P. ROUGIER

SCIENTIFIC VISUALISATION, PYTHON & MATPLOTLIB

First Edition: November 2021.
ISBN 978-2-9579901-0-8

This book is typeset in *Roboto*, *Source Serif Pro* & *Source Code pro*. It has been written in reStructuredText, converted to LaTeX using docutils and exported to Portable Document Format using XeLaTeX.

Printing:
1 2 3 4 5 6 7 8 9 10

Scientific Visualisation

Python & Matplotlib

Nicolas P. Rougier

Scientific Python – Volume II

In memory of ***John D. Hunter*** *(1968 — 2012), creator of the Matplotlib library &* ***Maxim Shemanarev*** *(1966 — 2013), creator of the antigrain geometry library. Two brilliant minds that are dearly missed.*

Contents

Preface

About the author

Nicolas P. Rougier is a full-time researcher in computational cognitive neuroscience, located in Bordeaux, France. He's doing his research at Inria (the French institute for computer science) and the Institute of Neurodegenerative Diseases where he investigates decision making, learning and cognition using computational models of the brain and distributed, numerical and adaptive computing, a.k.a. artificial neural networks and machine learning. His research aims to irrigate the fields of philosophy with regard to the mind-body problem, medicine to account for the normal and pathological functioning of the brain and the digital sciences to offer alternative computing paradigms. Beside neuroscience and philosophy, he's also interested in open and reproducible science (he has co-founded ReScience C with Konrad Hinsen and ReScience X with Etienne Roesch), scientific visualization (he created glumpy, co-created VisPy), Science outreach (e.g. The Conversation) and computer graphics (especially digital typography).

Nicolas P. Rougier has been using Python for more than 20 years and Matplotlib for more than 15 years for modeling in neuroscience, machine learning and for advanced visualization. Nicolas P. Rougier is the author of several online resources and tutorials and he's teaching Python, NumPy and scientific visualisation at the University of Bordeaux as well as at various conferences and schools worldwide.

About this book

This open access book has been written in reStructuredText converted to LaTeX using docutils and exported to Portable Document Format using XeLaTeX. Sources are available at github.com/rougier/python-scientific-visualisation

How to contribute

If you want to contribute to this book, you can:

- Review chapters & suggest improvements
- Report issues & correct my English
- Star the project on GitHub & buy the printed book

Prerequisites

This book is not a Python beginner guide and you should have an intermediate level in Python and ideally a beginner level in NumPy. If this is not the case, have a look at the bibliography for a curated list of resources.

Conventions

We will use usual naming conventions. If not stated explicitly, each script should import NumPy, SciPy and Matplotlib as:

```
import scipy
import numpy as np
import matplotlib.pyplot as plt
```

We'll use up-to-date versions (at the date of writing, June 2019) of the different packages:

```
>>> import sys; print(sys.version)
3.7.4 (default, Jul  9 2019, 18:13:23)
[Clang 10.0.1 (clang-1001.0.46.4)]
>>> import numpy; print(numpy.__version__)
1.16.4
>>> import scipy; print(scipy.__version__)
1.3.0
>>> import matplotlib; print(matplotlib.__version__)
3.1.0
```

License

Acknowledgments

I would like to thank all my sponsors & supporters for their great contributions, comments and questions that help to improve the book and the associated code. It's really great to have these people on your side you when you're writing a book.

Sponsors (in no specific order)

- Scott Lasley (@selasley)
- Steven Armour (@GProtoZeroW)
- Daniel Gomez (@dangom)
- Nathan Howell (@neh)
- Aymen-lng (@Aymen-lng)
- Paniterka (@paniterka)
- t. m. k. (@teehemkay)
- Alberto Mario Ceballos-Arroyo (@alceballosa)
- Panagiotis Simakis (@sp1thas)
- Anton Emelyanov (@emertonyc)
- Florent Langenfeld (@FloLangenfeld)
- Andrei Berceanu (@berceanu)
- Robin Mattheussen (@romatthe)
- Robert Crim (@ottbot)
- Jonas Bernoulli (@tarsius)

Supporters (in no specific order)

- Andrew (@ahuang11)
- Alister Burt (@alisterburt)
- @argapost

- @artinus
- Avihay Bar (@avihaybar)
- Georgios Bakirtzis (@bakirtzisg)
- @BCMarquez
- Behrooz Bashokooh (@BehroozBashokooh)
- Bill Little (@bjlittle)
- Benjamin Morgan (@bjmorgan)
- Sébastien Boisgérault (@boisgera)
- Brandon Rohrer (@brohrer)
- Brian Hamilton (@bsxfun)
- Christopher Anderson (@chrisLanderson)
- Chris Short (@ChristopherShort)
- @ciscostud
- Chris Morgan (@cmorgan)
- Onuralp (@cx0)
- Jochen Schröder (@cycomanic)
- David Ignacio Cortes (@davidcortesortuno)
- Danylo Malyuta (@dmalyuta)
- @earlev4
- Eitan Lees (@eitanlees)
- Javier González Monge (@Enterprixe)
- Folgert Karsdorp (@fbkarsdorp)
- Federico Vaggi (@FedericoV)
- Francisco (@fpozon)
- Giovanni d'Ario (@gdario)
- Georg Wille (@georgwille)
- Luciano Gerber (@gerberl)
- Jeff Borisch (@horshacktest)
- @imsalte
- @jafvert
- Jonathan Whitmore (@jbwhit)
- Julien Hillairet (@jhillairet)
- Joseph Szymborski (@jszym)
- @ltosti
- Matthieu Leroy (@m-leroy)
- Markus Degen (@MarkusDegen)
- Michael Dick (@midick)
- Niru Maheswaranathan (@nirum)

- @ofrighil
- @oszaar
- @paulgoulain
- Paul Nakroshis (@paulnakroshis)
- @qsandi
- Rik Huygen (@rhuygen)
- Rich Teague (@richteague)
- Rocco Meli (@RMeli)
- @s7oneghos7
- Sébastien Le Maguer (@seblemaguer)
- Andrew Slabko (@slabko)
- wonjun (@sleepyeye)
- Serge Toropov (@sombr)
- @samdani-1729
- Shen Zhou (@szsdk)
- Thomas Lentali (@tlentali)
- VO-PY (@VO-PY)
- Xavier Olive (@xoolive)
- Izaak "Zaak" Beekman (@zbeekman)
- zguo (@zguoch)
- Zhang Zhou (@zznature)

A special thanks to **Eitan Lees** who reviewed several chapters to correct my poor English. All remaining typos are my own. I also would like to thank **Sébastien Boisgérault** who helped me a lot with the automated build process (well, actually, he setup everything).

Introduction

The Python scientific visualisation landscape is huge (see figure 1). It is composed of a myriad of tools, ranging from the most versatile and widely used down to the more specialised and confidential. Some of these tools are community based while others are developed by companies. Some are made specifically for the web, others are for the desktop only, some deal with 3D and large data, while others target flawless 2D rendering.

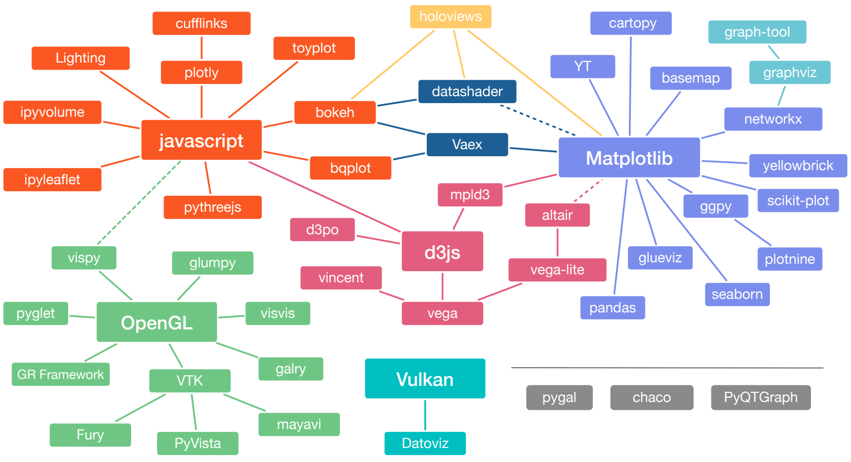

Figure 1
Python scientific visualisation landscape in 2018 (not exhaustive). Adapted from the original idea of Jake Vanderplas. **Sources:** github.com/rougier/python-visualization-landscape

Facing such a large choice, it may be thus difficult to find the package that best suit your needs, simply because you may not even be aware that this or that package exists. To help you in your choice, you can start by asking yourself a few questions:

- Do you target desktop or web rendering?
- Do you need complex 3D rendering?
- Do you need publication quality?
- Do you have very large data?
- Is there an active community?
- Are there documentation and tutorials?

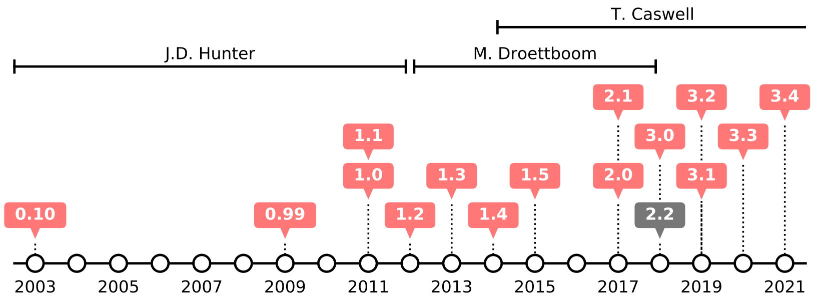

Figure 2
Matplotlib has been originally written by John D. Hunter and the first public version was released in 2003. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012, joined in 2014 by Thomas Caswell who is now (2021) the lead-developer. The latest version is 3.4 (at the time of writing), and is Python 3 only while the 2.2 version is a long term support version compatible with Python 2 and Python 3. **Sources:** introduction/matplotlib-timeline.py .

Depending on your answers, you may be able to decide which package to use and to invest some time learning it. For example, if you need interactive visualization in the browser with seamless integration with jupyter, bokeh might be an answer. If you have very large data and needs 3D on the desktop, vispy or mayavi might be an option. If you're interested in a very intuitive tool to rapidly build beautiful figures, then seaborn and altair are your friends. However, if you're working in geosciences, then you cannot overlook cartopy, etc. I cannot list them all and I'm sure that between the writing of this chapter and the actual publication of the book, new visualization libraries will have been created. A good source of information is the pyviz website (Python tools for data visualization) that offers a lot of pointers and has an up-to-date list of active packages (as opposed to dormant).

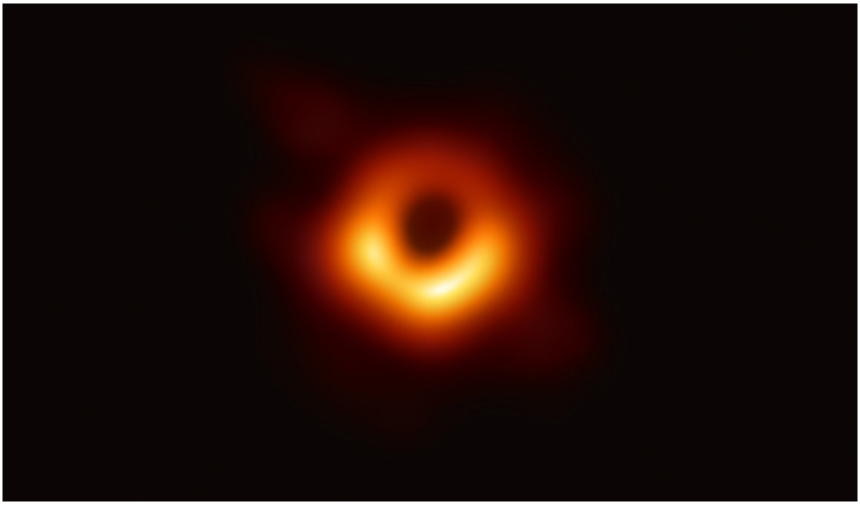

Figure 3
The supermassive black hole at the core of supergiant elliptical galaxy Messier 87, with a mass ~7 billion times the Sun's, as depicted in the first image released by the Event Horizon Telescope (10 April 2019). **Source:** Wikipedia

In this landscape, Matplotlib has a very special place. It was originally created by John D. Hunter in 2003 in order to visualize electrocorticography data. Here is the official announcement posted on the Python mailing list on May 23, 2003[1].

```
Matplotlib

  Matplotlib is a pure python plotting package for python and
  pygtk. My goal is to make high quality, publication quality
  plotting easy in python, with a syntax familiar to matlab
```

[1]Many thanks to Anthony Lee for pointing me to this archive.

```
  users. matplotlib is young, and several things need to be
  done for this goal is achieved. But it works well enough to
  make nice, simple plots.

Requirements

  python 2.2, GTK2, pygtk-1.99.x, and Numeric.

Download

  See the homepage - nitace.bsd.uchicago.edu:8080/matplotlib

Here are some of the things that matplotlib tries to do well

* Allow easy navigation of large data sets. Right click on
  figure window to bring up navigation tool bar for pan and
  zoom of x and y axes. This requires a wheel mouse. Place
  the wheel mouse over the navigation buttons and scroll away.
* Handle very large data sets efficiently by making use of
  Numeric clipping. I have used matplotlib in an EEG
  plotting application with 128 channels and several
  minutes of data sampled at 400Hz, eg, plotting matrices
  with dimensions 120,000 x 128.
* Choose tick marks and labels intelligently
* make easy things easy (subplots, linestyles, colors)
* make hard things possible (OO interface for full control)

Matplotlib is a class library that can be used to make plots
in pygtk applications. But there is a matlab functional
compatibility interface that you can get with, eg::

  from matplotlib.matlab import plot, subplot, show, gca

Example scripts and screenshots available at
http://nitace.bsd.uchicago.edu:8080/matplotlib

John Hunter
```

The initial goal was to replace the popular Matlab graphics engine and to support different platforms, to have high quality raster and vector output, to provide support for mathematical expressions and to work interactively from the shell. The first official release was made in 2003 (see figure 2) and more than 15 years later, the initial goals remains the same even though they have been further developed and polished. Today, the Matplotlib library is a *de facto* standard for Python scientific visualization. It has, for example, been used to display the first ever photography of a black hole (see figure 3) and to illustrate the existence of gravitational waves. Matplotlib is both a versatile and powerful library that allows you to design very high quality figures, suitable for scientific publishing. It offers both a simple and intuitive interface (`pyplot`) as well as an object oriented architecture that allows you to tweak anything within a figure. Note that, it can also be used as a regular graphic library in order to design non-scientific figures, as we'll see throughout this book. For example, the Matplotlib timeline figure (see figure 2) is simply made of a line with markers and some styled annotations.

This book is organized into 4 parts. The first part considers the fundamental principles of the Matplotlib library. This includes reviewing the different parts that constitute a figure, the different coordinate systems, the available scales and projections, and we'll also introduce a few concepts related to typography and colors. The second part is dedicated to the actual design of a figure. After introducing some simple rules for generating better figures, we'll then go on to explain the Matplotlib defaults and styling system before diving on into figure layout organization. We'll then explore the different types of plot available and see how a figure can be ornamented with different elements. The third part is dedicated to more advanced concepts, namely 3D figures, optimization, animation and toolkits. Lastly, the fourth and final part is a collection of showcases and their analysis.

I Fundamentals

1 Anatomy of a figure

A matplotlib figure is composed of a hierarchy of elements that, when put together, forms the actual figure as shown on figure 1.1. Most of the time, those elements are not created explicitly by the user but derived from the processing of the various plot commands. Let us consider for example the most simple matplotlib script we can write:

```
plt.plot(range(10))
plt.show()
```

In order to display the result, matplotlib needs to create most of the elements shown on figure 1.1. The exact list depends on your default settings (see chapter 7), but the bare minimum is the creation of a Figure that is the top level container for all the plot elements, an Axes that contains most of the figure elements and of course your actual plot, a line in this case. The possibility to not specify everything might be convenient but in the meantime, it limits your choices because missing elements are created automatically, using default values. For example, in the previous example, you have no control of the initial figure size since is has been chosen implicitly during creation. If you want to change the figure size or the axes aspect, you need to be more explicit:

```
fig = plt.figure(figsize=(6,6))
ax = plt.subplot(aspect=1)
ax.plot(range(10))
plt.show()
```

In many cases, this can be further compacted using the subplots method.

```
fig, ax = plt.subplots(figsize=(6,6),
                       subplot_kw={"aspect"=1})
ax.plot(range(10))
```

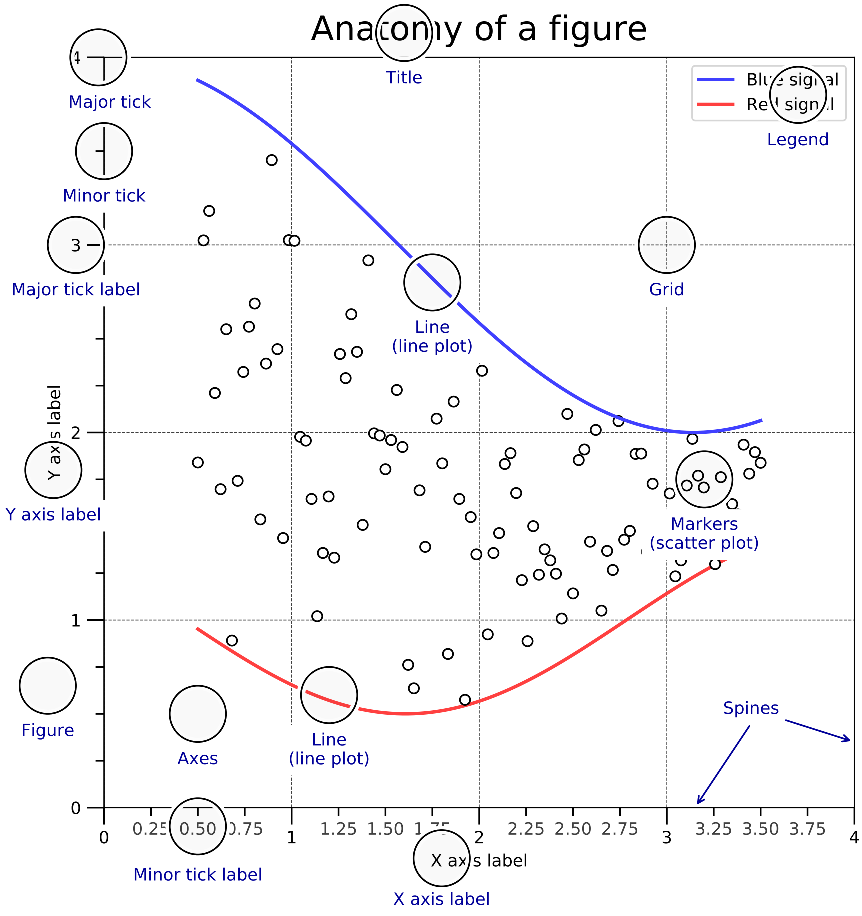

Figure 1.1
A matplotlib figure is composed of a hierarchy of several elements that, when put together, forms the actual figure (sources: anatomy/anatomy.py).

```
plt.show()
```

Elements

You may have noticed in the previous example that the plot command is attached to `ax` instead of `plt`. The use of `plt.plot` is actually a way to tell matplotlib that we want to plot on the current axes, that is, the last axes that has been created, implicitly or explicitly. No need to remind that *explicit is better than implicit* as explained in the The Zen of Python, by Tim Peters (`import this`). When you have choice, it is thus preferable to specify exactly what you want to do. Consequently, it is important to know what are the different elements of a figure.

Figure: The most important element of a figure is the figure itself. It is created when you call the `figure` method and we've already seen you can specify its size but you can also specify a background color (`facecolor`) as well as a title (`suptitle`). It is important to know that the background color won't be used when you save the figure because the savefig function has also a `facecolor` argument (that is white by default) that will override your figure background color. If you don't want any background you can specify `transparent=True` when you save the figure.

Axes: This is the second most important element that corresponds to the actual area where your data will be rendered. It is also called a subplot. You can have have one to many axes per figure and each is usually surrounded by four edges (left, top, right and bottom) that are called **spines**. Each of these spines can be decorated with major and minor **ticks** (that can point inward or outward), **tick labels** and a **label**. By default, matplotlib decorates only the left and bottom spines.

Axis: The decorated spines are called axis. The horizontal one is the **xaxis** and the vertical one is the **yaxis**. Each of them are made of a spine, major and minor ticks, major and minor ticks labels and an axis label.

Spines: Spines are the lines connecting the **axis** tick marks and noting the boundaries of the data area. They can be placed at arbitrary positions and may be visible or invisible.

Artist: Everything on the figure, including Figure, Axes, and Axis objects, is an artist. This includes Text objects, Line2D objects, collection