Introduction to Python Day 6

Verjinia Metodieva and Daniel Parthier

2025-04-01

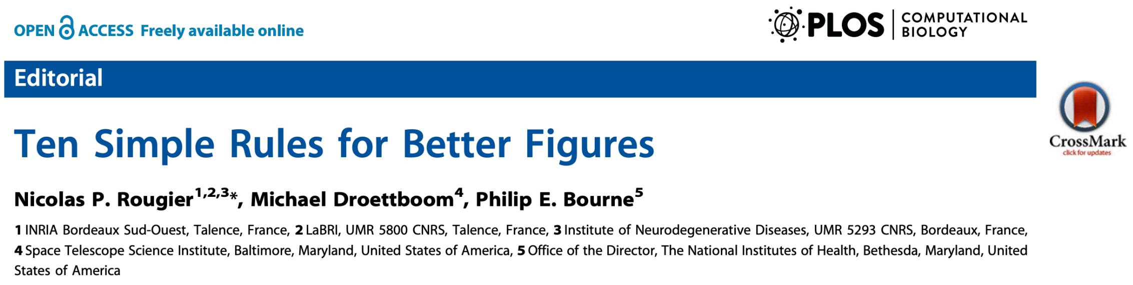
"Visualization is a method of computing. It transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations. Visualization offers a method for seeing the unseen. It enriches the process of scientific discovery and fosters profound and unexpected insights. In many fields it is already revolutionizing the way scientists do science.

~ McCormick, B.H., T.A. DeFanti, M.D. Brown, Visualization in Scientific Computing, Computer Graphics Vol. 21.6, November 1987

-

"Sometimes the most effective way to describe, explore, and summarize a set of numbers - even a very large set - is to look at those numbers"

~ The Visual Display of Quantitative Information, Edward Tufte, 1983


OPEN ACCESS Freely available online

PLOS | COMPUTATIONAL BIOLOGY

Editorial

Ten Simple Rules for Better Figures

Nicolas P. Rougier[1,2,3]*, Michael Droettboom[4], Philip E. Bourne[5]

1 INRIA Bordeaux Sud-Ouest, Talence, France, 2 LaBRI, UMR 5800 CNRS, Talence, France, 3 Institute of Neurodegenerative Diseases, UMR 5293 CNRS, Bordeaux, France, 4 Space Telescope Science Institute, Baltimore, Maryland, United States of America, 5 Office of the Director, The National Institutes of Health, Bethesda, Maryland, United States of America

- audience, message, caption, choice of viz, no misleading plotting

Matplotlib

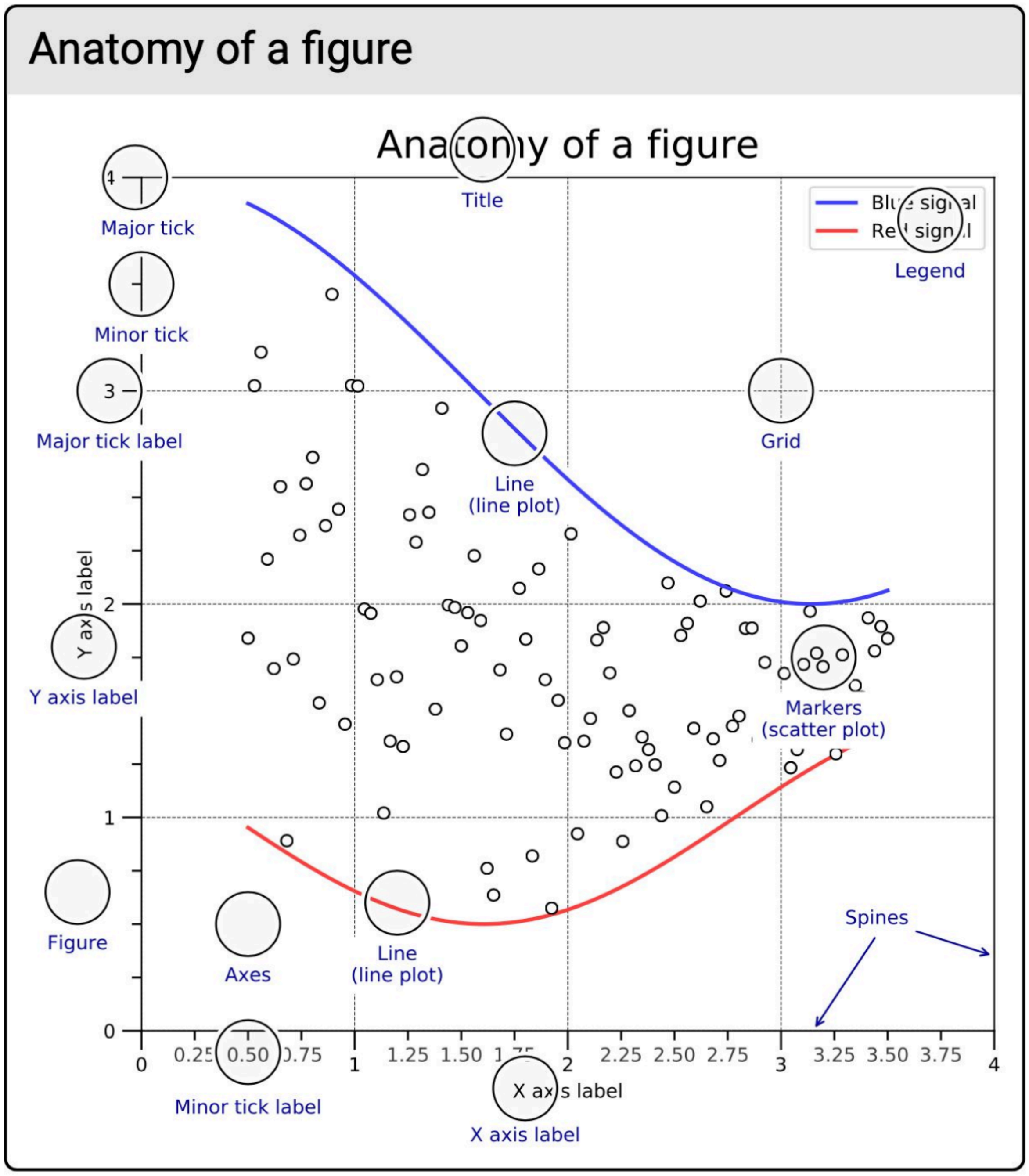
- library for visualization with Python

```
import matplotlib.pyplot as plt
```

- other plotting libraries: Seaborn, Plotly, Bokeh, Altair

Anatomy of a figure

[Find the complete cheatsheet on GitHub in the ‘img/’ folder]



mapping data info into visual info

- necessary: data (y-axis elements)
- x-axis elements

- type of plot: scatter plot, line plot, histogram, bargraph, violin plot, etc.
- shape, size, and color specification
- axis ticks and labels
- legend
- title

Interfaces/ Styles of Use

- implicit pyplot interface
  - example: plt.xlabel
  - methods are generally applied to the current axes or figure
- **object-oriented interface**
  - example: (ax.set_xlabel)
  - uses methods on a figure or axes object to create other artists, and
  - allows to build a visualization step by step

Useful links

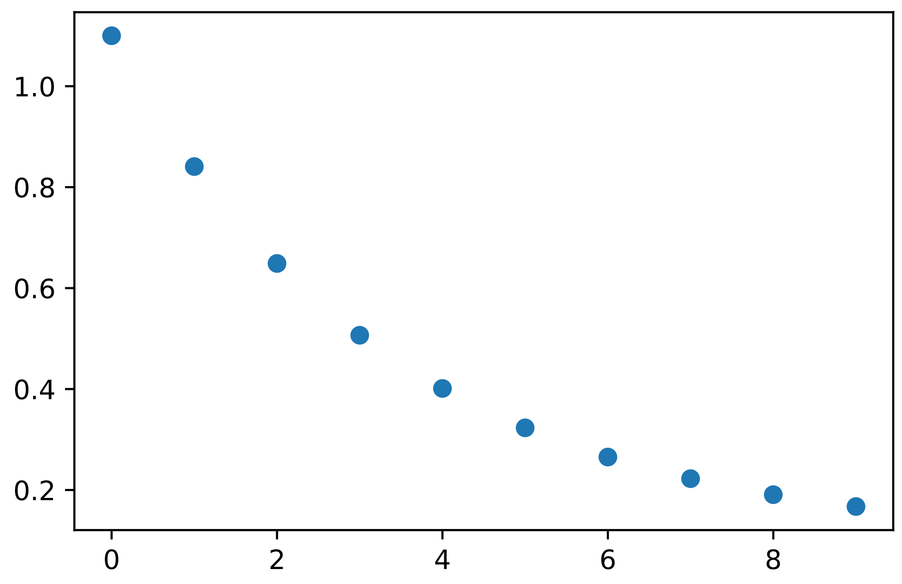
https://datavizcatalogue.com/index.html https://matplotlib.org/stable/gallery/index.html
https://matplotlib.org/cheatsheets/ https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.10

Recap

Specify a function

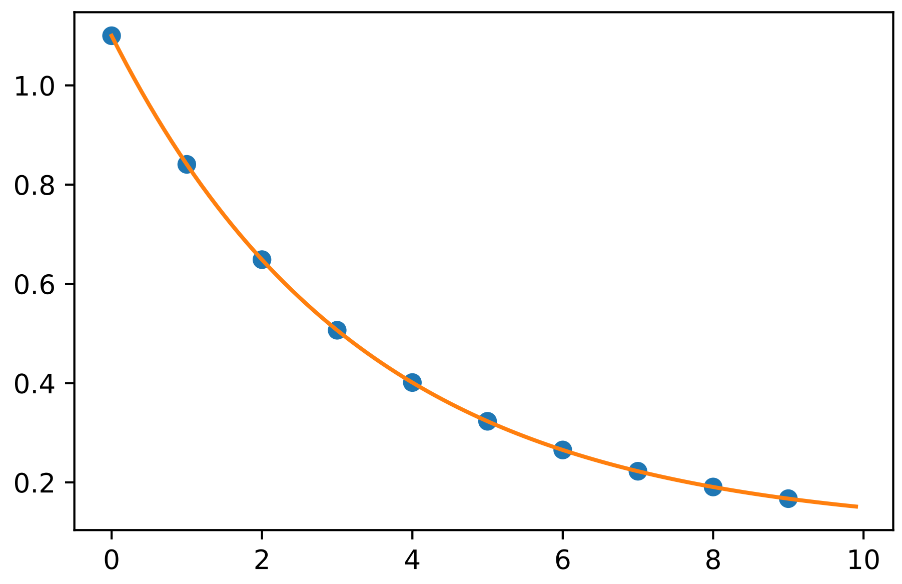
```python
import numpy as np
def func(x, a, b, c):
    return a * np.exp(-b * x) + c

x = np.arange(0,10,1)
y = func(x, 1,0.3,0.1)
plt.plot(x, y, 'o')
plt.show()
```



Fit function

```
from scipy import optimize
popt, pcov = optimize.curve_fit(func, x, y, p0=(1,1,1))
x_fit = np.arange(0,10,0.1)
plt.plot(x, y, 'o')
plt.plot(x_fit, func(x_fit, *popt))
plt.show()
print(popt)
```



```
[1.  0.3 0.1]
```

Convolve and Deconvolve

- Signals can be distorted
- Multiplying something to signal with `convolve`
- Remove known distortion from signal `deconvolve`
- Where do you see use cases?

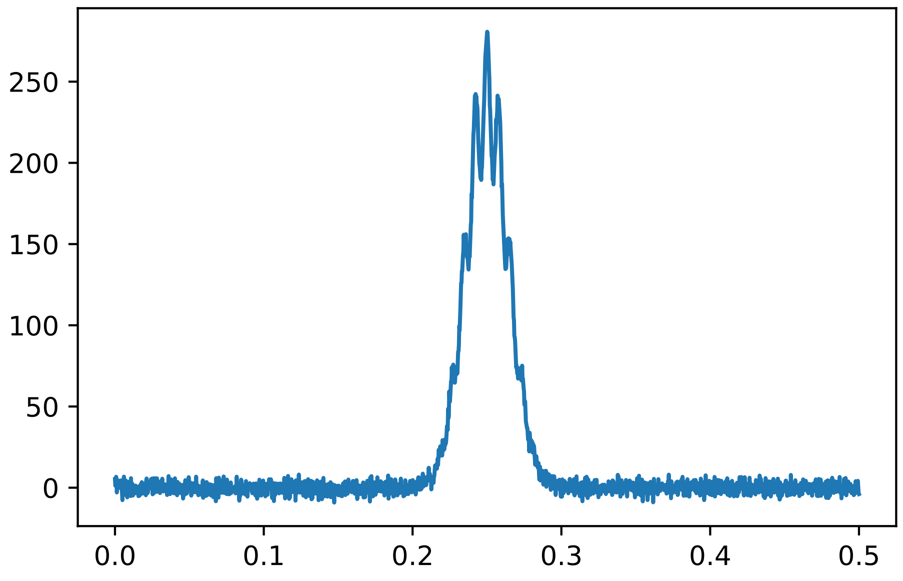
Signal analysis (power)

```
time = np.linspace(start=0, stop=0.5, num=2000)
mu, sigma = 0.25, 0.01
sinewave = np.sin(time * 250 * np.pi)
gaussian = (1 / (np.sqrt(2 * np.pi * np.square(sigma)) *
            np.exp(-(np.square(time - mu) /np.square(2 * sigma))))
ripple = gaussian * sinewave
```

Signal analysis (power)

```
np.random.seed(0)
trace = ripple + gaussian*6 + np.random.normal(0, 3, size=time.shape)
```

```
plt.plot(time, trace)
plt.show()
```



Signal analysis (power)

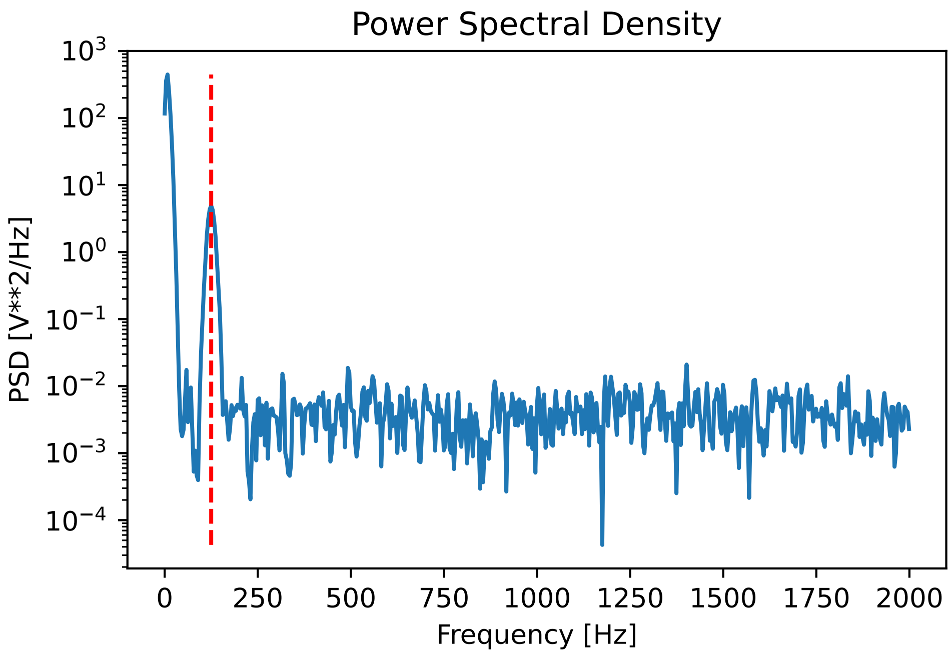
- Module `signal`
- Power spectrum to analyse frequencies
- e.g. `welch`

Power spectrum

```
from scipy import signal
f, Pxx_den = signal.welch(trace, fs=1/time[1], nperseg=1024, average='median')
plt.semilogy(f, Pxx_den)
plt.vlines(125, Pxx_den.min(), Pxx_den.max(),color='r', linestyle='--')
plt.xlabel('Frequency [Hz]')
plt.ylabel('PSD [V**2/Hz]')
plt.title('Power Spectral Density')
plt.show()
```

```
Text(0.5, 0, 'Frequency [Hz]')
```

```
Text(0, 0.5, 'PSD [V**2/Hz]')
Text(0.5, 1.0, 'Power Spectral Density')
```



- Returns frequencies `f` and power densities `Pxx_den`

Filtering data

- Filtering data is used to remove different components from signal
- Lowpass, highpass, bandpass
- Different types of filters (Butterworth, Chebyshev, FIR, etc.)

Filter construction

- Need different parts
  - Nyquist frequency
  - Filter frequencies (0 to 1 - frequency/Nyquist)
  - Type of filter
  - Order ('strength')
- Outputs are filter coefficients

```
sampling_rate = 1/time[1]
nyquist = sampling_rate / 2
b, a = signal.butter(N=5, Wn=[100/nyquist, 250/nyquist], btype='band')
```

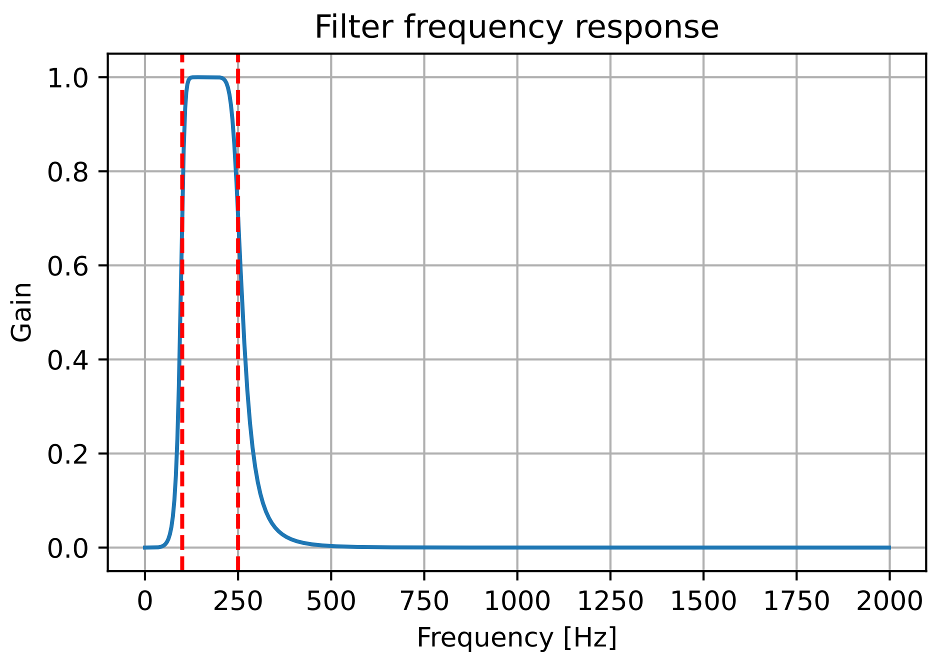
Filter construction

```
w, h = signal.freqz(b, a, fs=sampling_rate, worN=2000)
plt.plot(w, abs(h))
plt.title('Filter frequency response')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Gain')
plt.grid()
plt.axvline(100, color='r', linestyle='--')
plt.axvline(250, color='r', linestyle='--')
plt.show()
```

```
Text(0.5, 1.0, 'Filter frequency response')
```

```
Text(0.5, 0, 'Frequency [Hz]')
```

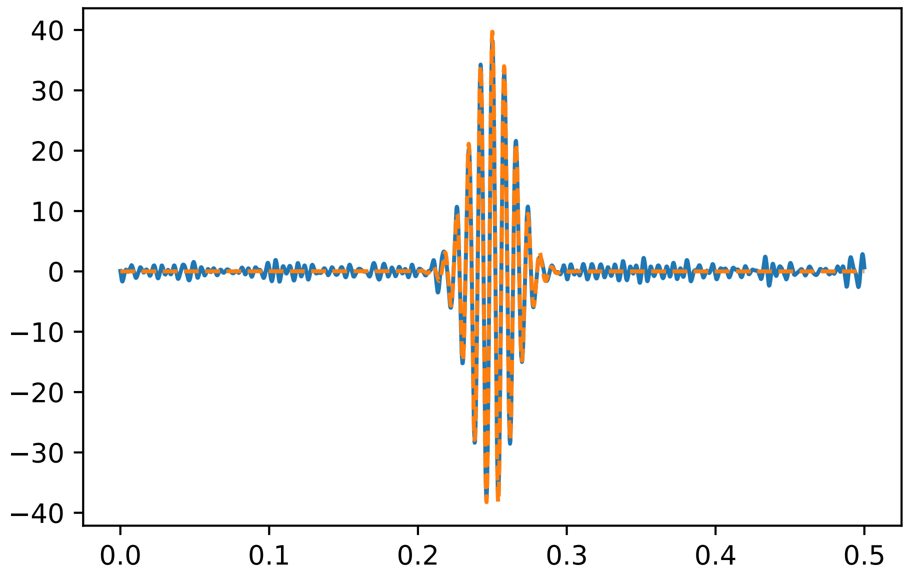
```
Text(0, 0.5, 'Gain')
```



Apply filter

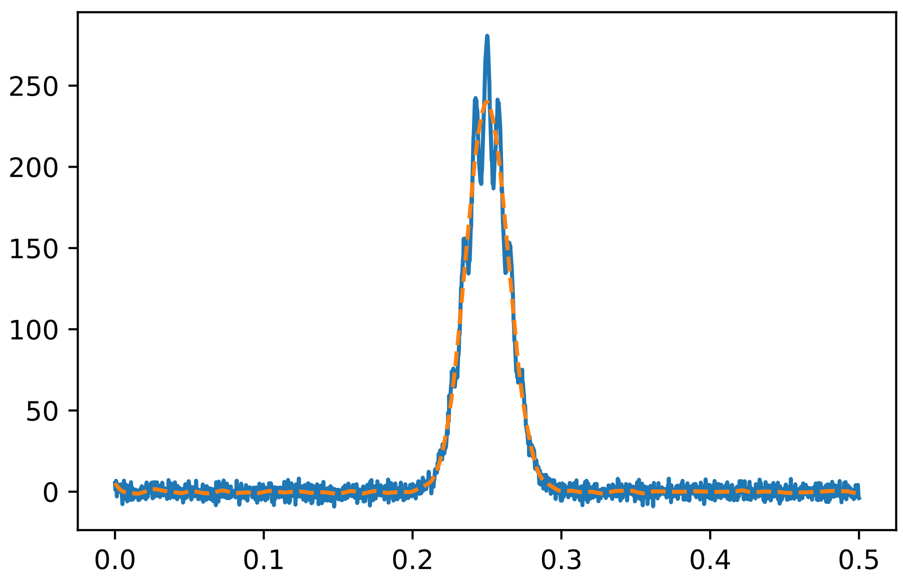
- Different ways of filtering (IIR/FIR)
- One way or two ways (filtfilt)

```
ripple_filtered_trace = signal.filtfilt(b, a, trace)
plt.plot(time, ripple_filtered_trace)
plt.plot(time, ripple, '--')
plt.show()
```



Filter for lowpass

```
b, a = signal.butter(5, 80/nyquist, btype='low')
lowpass_filtered_trace = signal.filtfilt(b, a, trace)
plt.plot(time, trace)
plt.plot(time, lowpass_filtered_trace, '--')
plt.show()
```



250
200
150
100
50
0
0.0
0.1
0.2
0.3
0.4
0.5