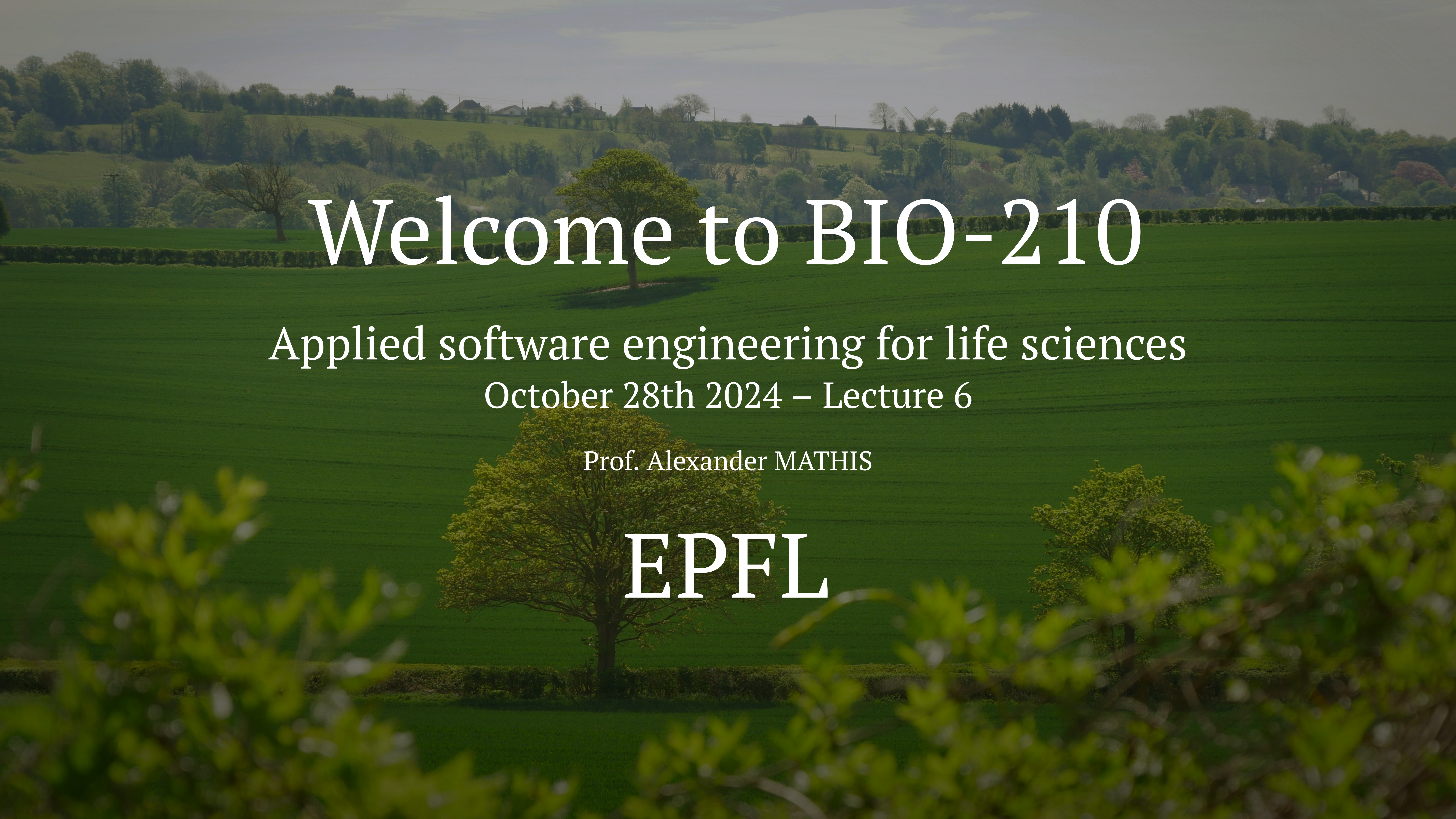


Welcome to BIO-210

Applied software engineering for life sciences

October 28th 2024 – Lecture 6

Prof. Alexander MATHIS

EPFL

Announcements I

- Congrats on your excellent quiz results: Quiz 1: class average 9.7/10;
- For Quiz 2 we have: 8.6/12. Let’s keep studying Python!

Announcements II

- Today is the *second* in person quiz: please come *in time*, there will be no extra time. Submission closes at 13:35. To start, you’ll need to sign in. Bring your Camipro. No notes are allowed. If you switch to a different tab from Moodle’s quiz or communicate with somebody, you’ll receive 0 points.
- Monday 15:15 - 16: my office hours at SV 2811

Announcements III

- Final room assigment here. NOTE: you need your EPFL login to see it!
- v2 of your project was due at 10am today (not graded/checked), check out release guide.
  - try to do the majority in the exercise session (we share the problem set at least by Friday, so you can prepare better)
  - what if you get stuck? -> discuss with your teammates, ask on ED, … and *release* your best version on Monday at 10am
  - make sure you get feedback about your latest version on Monday from the SA/TAs! You can release a bugfix/patch/update, e.g. `` `v2.1` ``, see details on releases
- We will provide code review for your v2!

Quiz

You have a dataset representing the expression levels of 5 genes across 4 tissue samples. Each row corresponds to a gene, and each column corresponds to a tissue sample.

```
import numpy as np
expression_levels = np.array([
    [5.1, 2.3, 3.4, 6.5],  # Gene 1
    [1.5, 3.5, 2.4, 4.6],  # Gene 2
    [3.2, 5.1, 1.6, 3.8],  # Gene 3
    [4.1, 3.2, 4.5, 2.2],  # Gene 4
    [2.8, 1.5, 3.1, 5.0],  # Gene 5
])

```

Write a program to calculate the standard deviation of the expression levels of each gene.

```
In [1]: np.std(expression_levels,axis=1)
Out[1]: array([1.60370664, 1.16404467, 1.25772612, 0.88600226, 1.2509996 ])

In [2]: np.sqrt(np.var(expression_levels,axis=1)) #if you don't know the std command
Out[2]: array([1.60370664, 1.16404467, 1.25772612, 0.88600226, 1.2509996 ])
```

Useful conventions for developing a project

New feature development:

- for new features make branches. Give the branch a good name, e.g. *your_name/novel_featurename*
- once you’re ready you make a pull request and assign your collaborators for review (e.g., see this example PR)
- here is an example for the demo project

Recommended project workflow:

1. Develop a feature (on a branch)
2. Merge main/master into it [when you’re done]
3. Test it, again after you merge: `` `git push` ``
4. create a pull request and assign your teammates as reviewers

Alternative workflow: sometimes 2) and 3) are done by the reviewer. i.e. they merge!

Comments in Python code

We already learned that "#" allows to put comments in code.

```python
# Hello world <--- this will not be interpreted!
# "#" also allows multi-line comments

# we have also seen inline comments...
a=3       # you can also make inline comments!
b=4       # assigning b to 4.

''' single quotes
You can also make long comments ... everything is "ignored" by python!
a=f2d123ee1505
b=123
'''
c=a+b

""" quotation marks
Alternative,
multiline comment
"""

```

Some guidelines (not rules)

From pep 8 = Style Guide for Python Code

- Comments should be complete sentences. The first word should be capitalized, unless it is an identifier that begins with a lower case letter (never alter the case of identifiers, aka keywords, module names, etc.).
- Ensure that your comments are clear and easily understandable to other speakers of the language you are writing in.
- You can look for typos by using the pip-package codespell.
- **Comments that contradict the code** are worse than no comments. Always make a priority of keeping the comments up-to-date when the code changes!

P.S.: PEP stands for Python Enhancement Proposal. A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment. The PEP should provide a concise technical specification of the feature and a rationale for the feature – from PEP 1.

Quiz: How do you create a np.array …

… starting at 12, ending at 176 containing every third number?

```
In [1]: import numpy as np
In [2]: np.arange(12,177,3)
Out[2]:
array([ 12,  15,  18,  21,  24,  27,  30,  33,  36,  39,  42,  45,  48,
        51,  54,  57,  60,  63,  66,  69,  72,  75,  78,  81,  84,  87,
        90,  93,  96,  99, 102, 105, 108, 111, 114, 117, 120, 123, 126,
       129, 132, 135, 138, 141, 144, 147, 150, 153, 156, 159, 162, 165,
       168, 171, 174])
```

But what if you forgot how to use `np.arange`?

```
help(np.arange)
np.arange?
```

The displayed help is actually the `docstring`

```
In [3]: help(np.arange)

Help on built-in function arange in module numpy:

arange(...)
    arange([start,] stop[, step,], dtype=None, *, like=None)

    Return evenly spaced values within a given interval.

    Values are generated within the half-open interval ``[start, stop)``
    (in other words, the interval including `start` but excluding `stop`).
    For integer arguments the function is equivalent to the Python built-in
    `range` function, but returns an ndarray rather than a list.

    When using a non-integer step, such as 0.1, the results will often not
    be consistent.  It is better to use `numpy.linspace` for these cases.

    Parameters
    ----------
    start : integer or real, optional
        Start of interval.  The interval includes this value.  The default
        start value is 0.
    stop : integer or real
        End of interval.  The interval does not include this value, except
```

Essential documentation: Docstrings

A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the `__doc__` special attribute of that object.

- Docstrings provide help for your code (so you (and others) can re-use it in the future!)

```
In [4]: def myfun(x):
    ...:   ''' identity function '''        # Docstrings are defined like comments!
    ...:   return x
    ...:
    ...: print(myfun.__doc__)       # Docstrings are assinged to the attribute `__doc__`
    ...:
 identity function

 In [5]: help(myfun)                # They become accessible via help!
 Help on function myfun in module __main__:

 myfun(x)
     identity function
```

Python’s recommendations for docstrings

- Write docstrings for all public modules, functions, classes, and methods.
- Docstrings are not necessary for non-public methods, but you should have a comment that describes what the method does. This comment should appear after the def line.
- PEP 257 immortalizes Python’s docstring conventions
- For mathematical functions (like in our projects) the detailed numpy style guide is excellent to follow

What should be contained in a docstring?

- A **Short summary** (for basic, simple functions)

```python
def add(a, b):
    """The sum of two numbers.

    """
```

- **Extended summary** contains among others:
  - a simple description (clarify functionality, not implementation details those belong to Notes)
  - parameters
  - returns
  - examples
  - notes
  - references

Details are available in the Numpy doc style guide

Example (*shortened* from np.arange)

```
In [13]: np.arange?
Return evenly spaced values within a given interval.
!!OMMITTED for space reasons!!
Parameters
----------
start : integer or real, optional
    Start of interval.  The interval includes this value.  The default
    start value is 0.
stop : integer or real
!!OMMITTED for space reasons!!

Returns
-------
arange : ndarray
    Array of evenly spaced values.
!!OMMITTED for space reasons!!

Examples
--------
>>> np.arange(3)
array([0, 1, 2])
Type:      builtin_function_or_method
```

Documentation

- Note that the (numpy) docstrings are also (html-rendered) on the web, e.g., for np.arange
- this is all automatically generated with Sphinx, see https://github.com/numpy/doc

Docstrings in action

- Today you will work on docstrings for your functions!
- Compare to the demo-project

Quiz: How do you define a function that can offset the output by a specific parameter with default 2?

```
In [1]: def f(x,offset = 2):
   ...:     return x+offset
   ...:

# Testing our function:
In [2]: f(0)
Out[2]: 2

In [3]: f(3)
Out[3]: 5

In [4]: f(2,5)
Out[4]: 7
```

Quiz: How do you write docstrings for this function?

```
def offsetter(x,offset = 2):        # use a good name!
    """ Function that offsets input by default value (offset)
    Parameters
    ----------
    x      : array or float
    offset : float, optional
              default 2
    Returns
    -------
    numpy array or float
         x + offset
    Examples
    --------
    >>> offsetter(2)
    4
    >>> offsetter(2,3)
    5
    """
    return x+offset
```

Questions?

Visualization

- is crucial in science and beyond ("a picture is worth 1000 words…")
- python has strong support for plotting with maptlotlib (our focus), seaborn (neat interface on top), Majavi (esp. 3D visualization), Plotly (esp. web), Bokeh (esp. web), Pandas, gnuplot, …

P.S. A formula is worth a thousand pictures… (by Edsger W. Dijkstra)

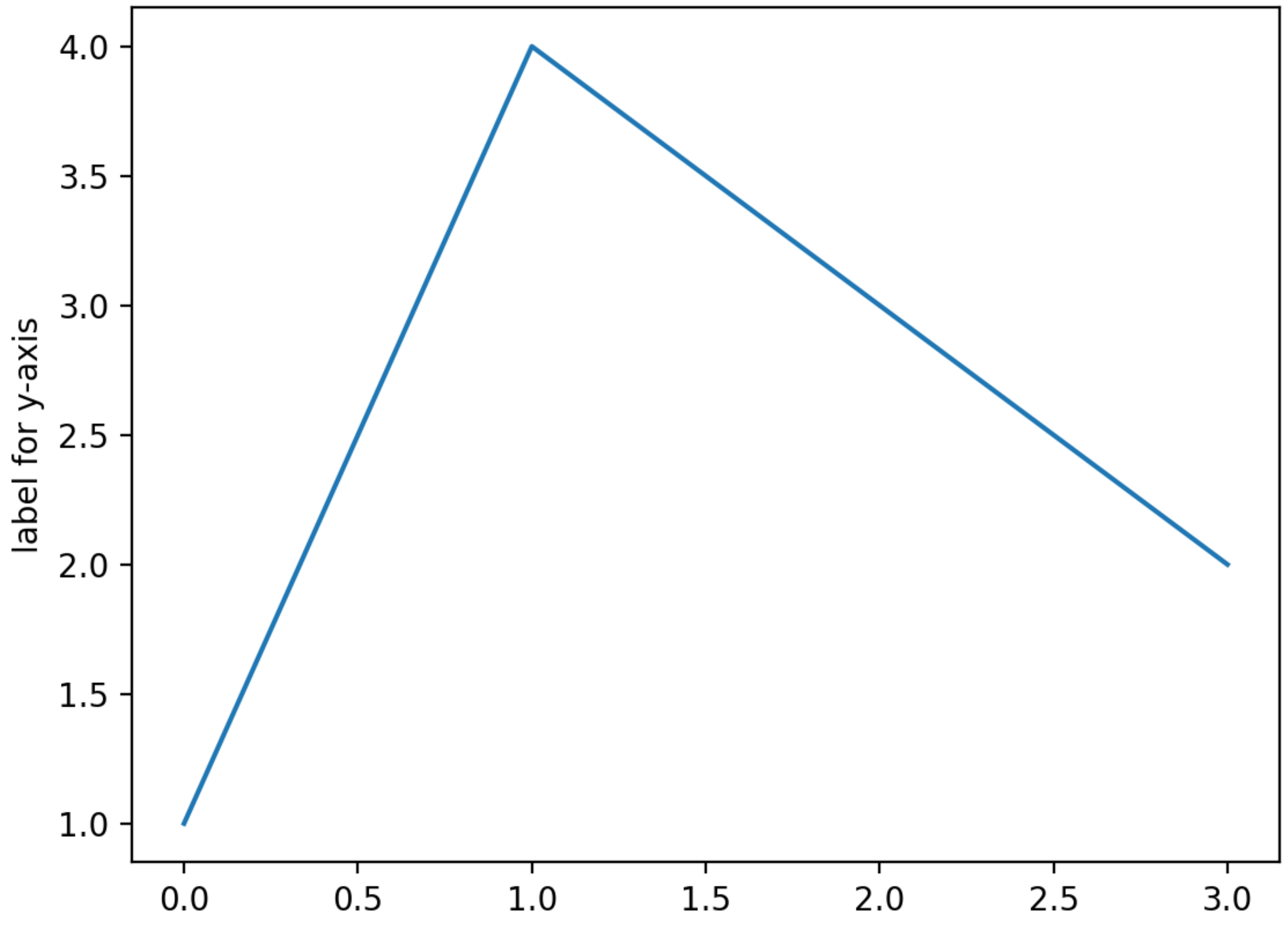
P.P.S. Just like Numpy, maptlotlib is a library you need to install --> `` `pip install matplotlib` ``

Pyplot: simple plotting in matplotlib

```python
import matplotlib.pyplot as plt     # Importing matplotlib.pyplot
# Note: we use all functions from this library with plt.XYZ
plt.plot([1, 4, 3, 2])              # Plotting x vs. y data
plt.ylabel('label for y-axis')      # Making a label for y
plt.show()                          # Display all open figures
```

Pyplot: simple plotting in matplotlib

```python
import matplotlib.pyplot as plt     # Importing matplotlib.pyplot
# Note: we use all functions from this library with plt.XYZ
plt.plot([1, 4, 3, 2])              # Plotting x vs. y data
plt.ylabel('label for y-axis')      # Making a label for y
plt.show()                          # Display all open figures
```

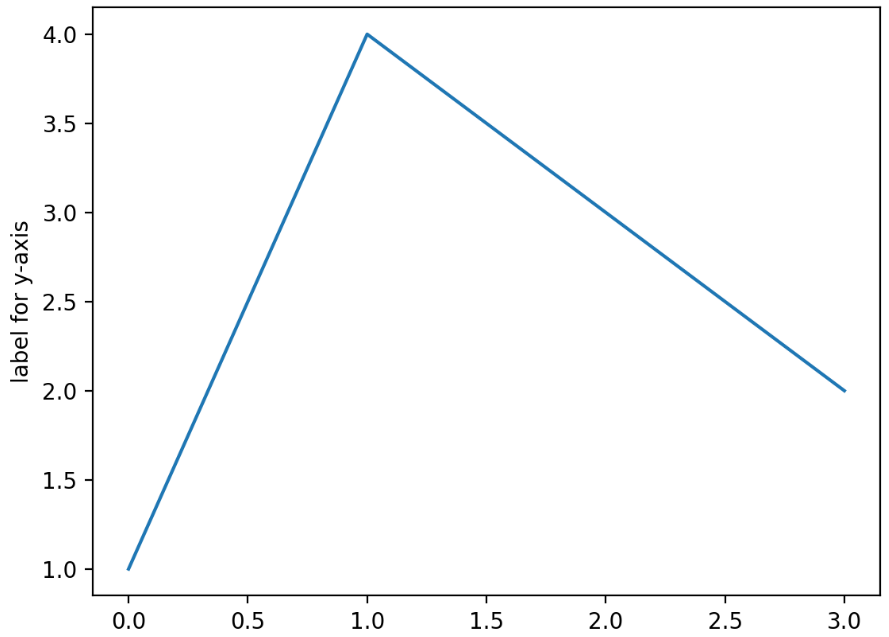


Quiz: Why does the plot look like this?

```python
import matplotlib.pyplot as plt     # Importing matplotlib.pyplot
# Note: we use all functions from this library with plt.XYZ
plt.plot([1, 4, 3, 2])              # Plotting x vs. y data
plt.ylabel('label for y-axis')      # Making a label for y
plt.show()                          # Display all open figures
```

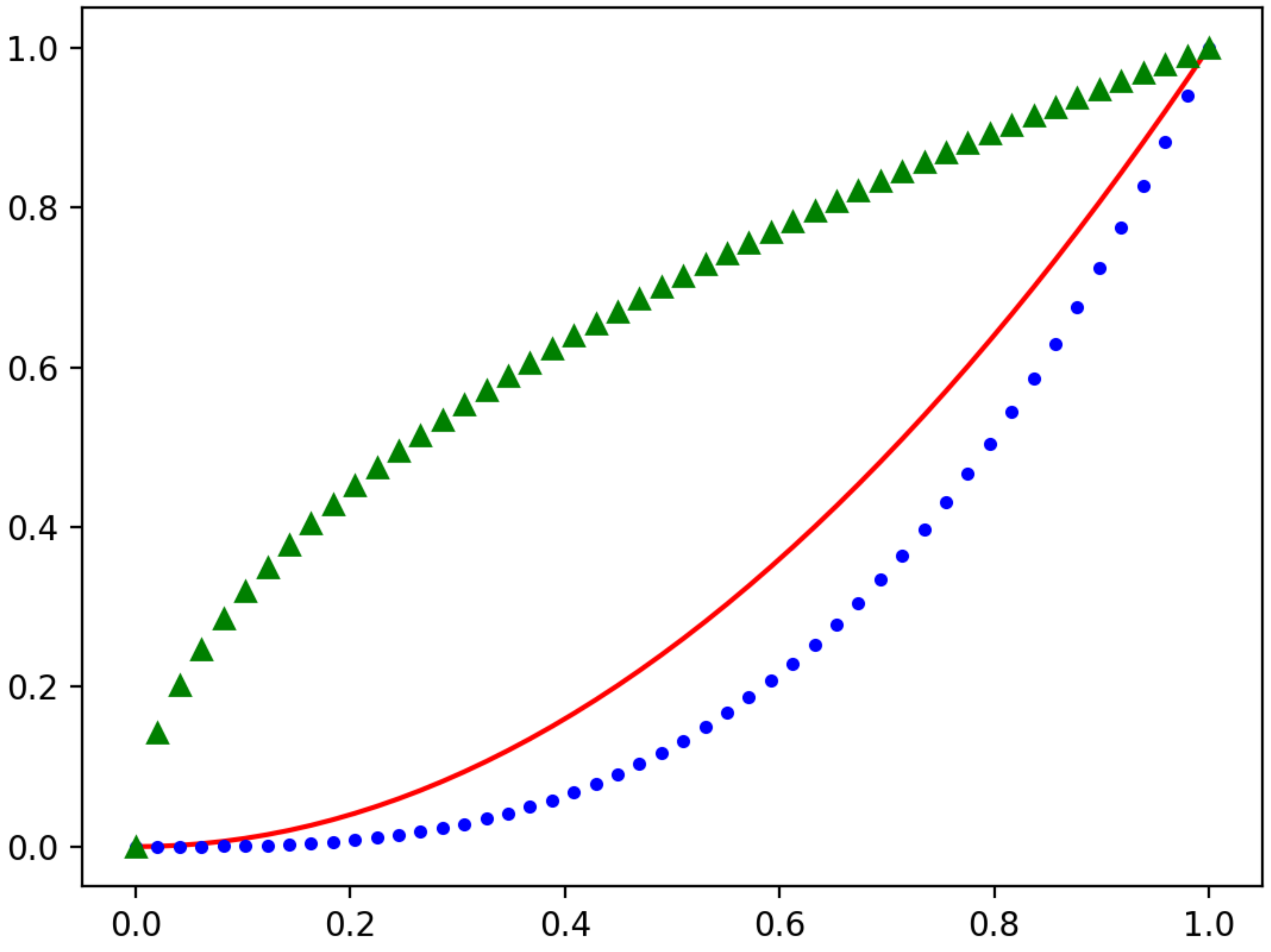
Quiz: Why does the plot look like this?

If one gives just one array `` `a` ``, this array is interpreted as y-axis values. By default the x-values are just enumerated 0 to len(a)-1. All those points are connected by line segments.

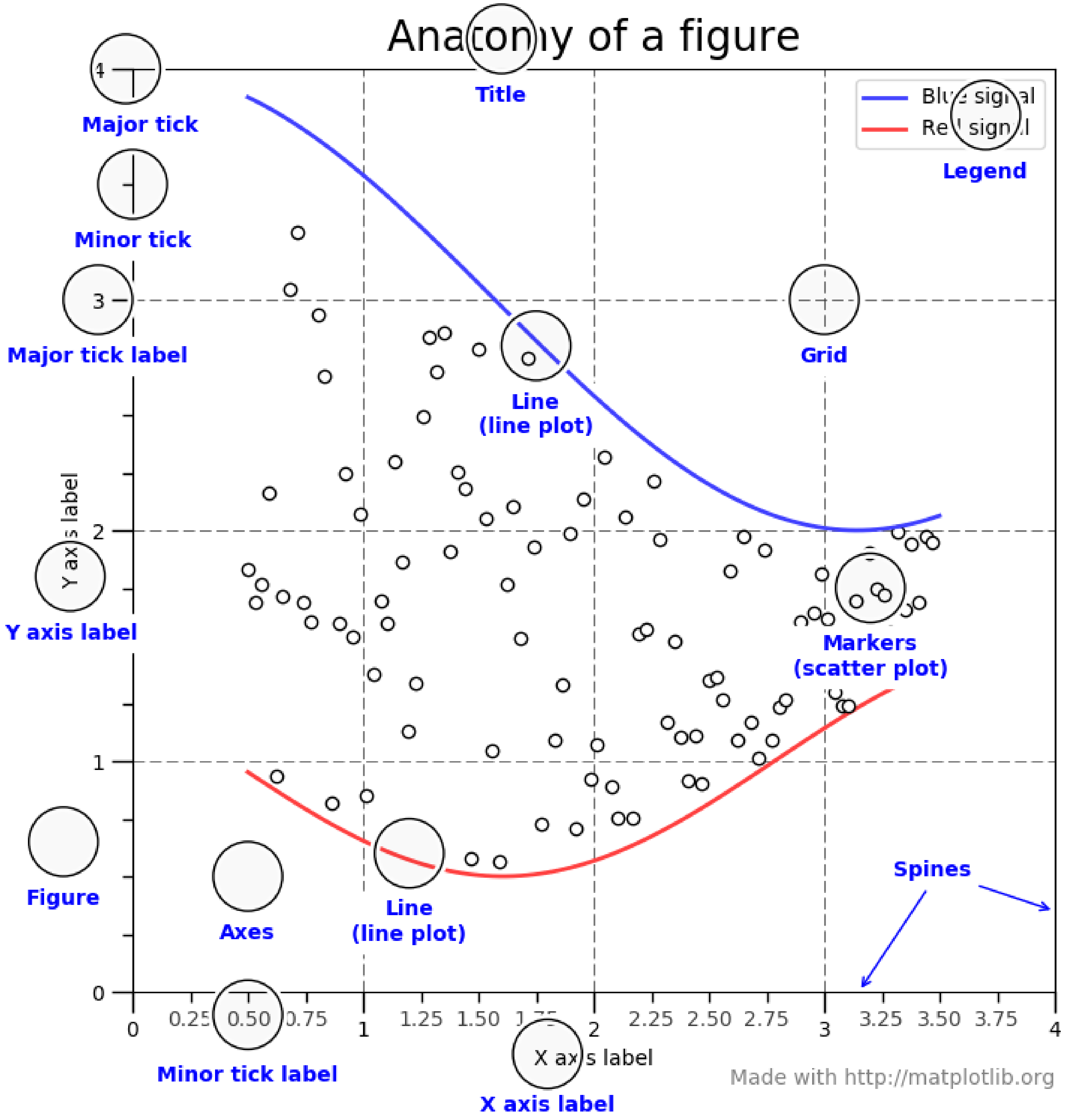


Example 2: formatting the style of plot

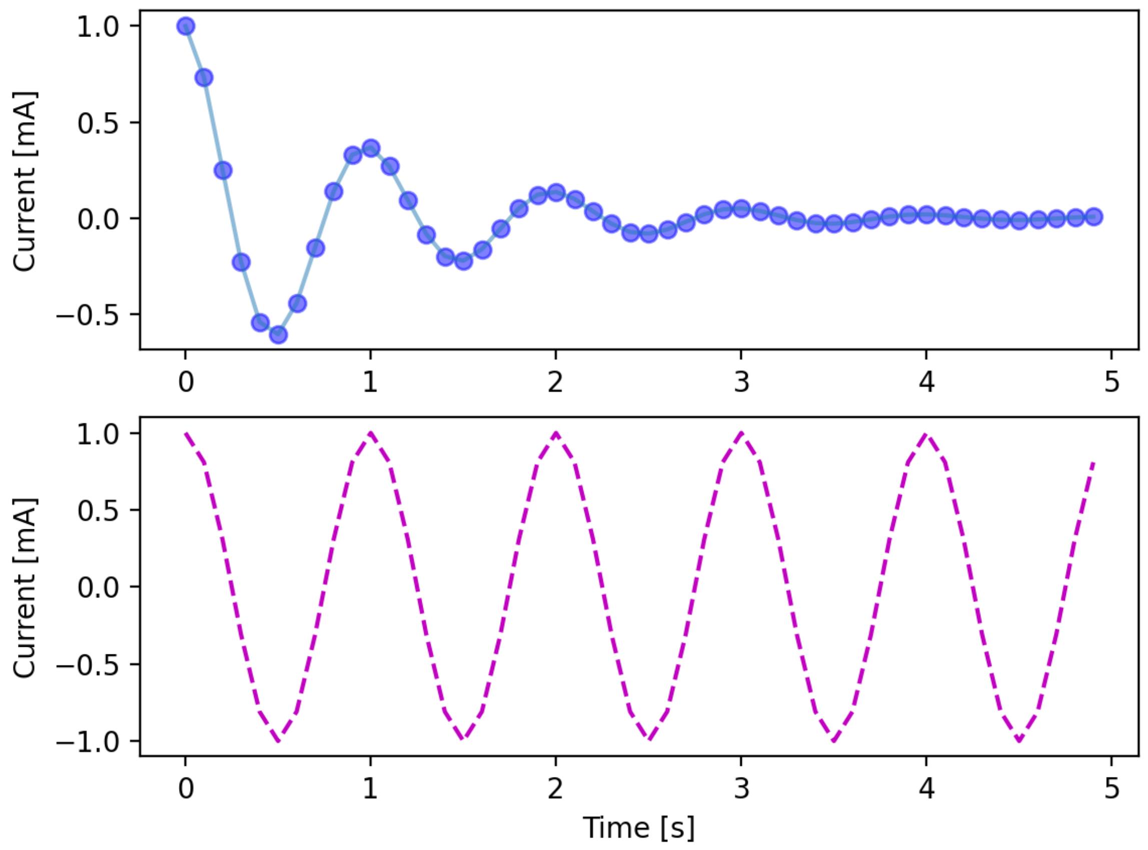
```
import numpy as np
x = np.linspace(0,1,50)
# Plotting x vs. y data  (for multiple functions/ x-y pairs with their own style)
plt.plot(x,x**2,'r',x,x**3,'b.',x,np.sqrt(x),'g^')
plt.show()
```



Anatomy of a matplotlib figure

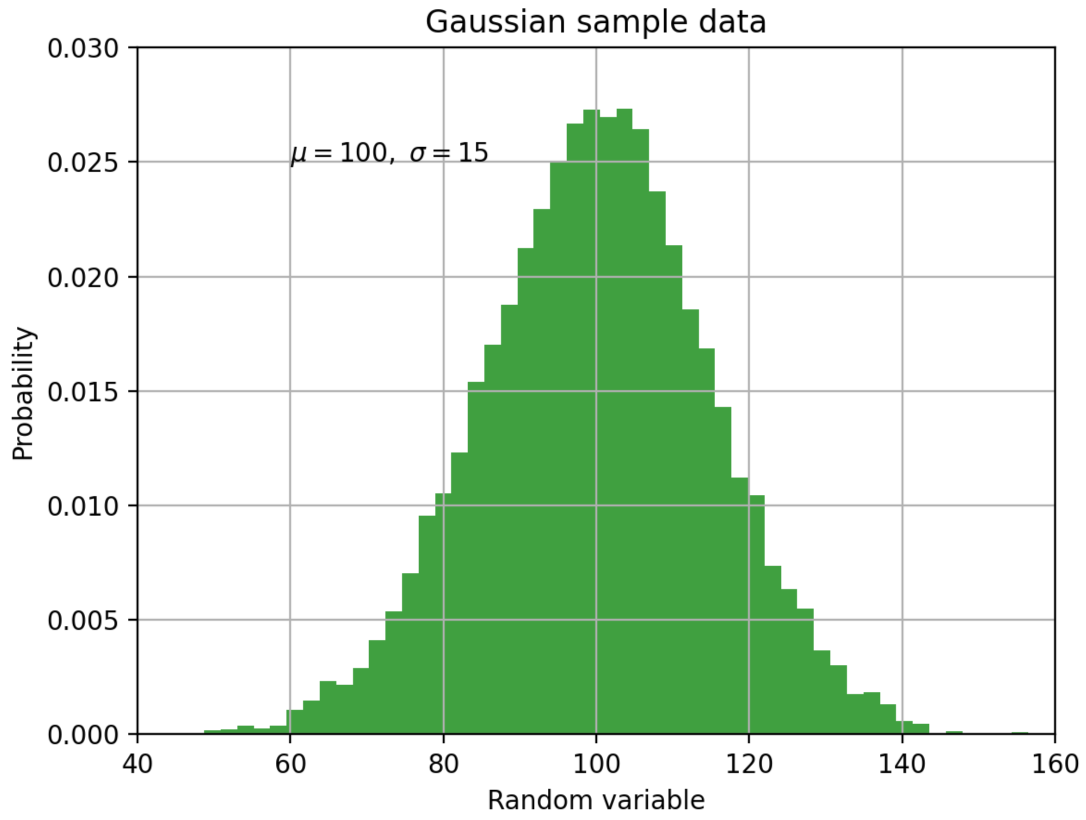


```python
# Creating figures with subplots (here nrows = 2, ncols = 1)
T = np.arange(0.0, 5.0, 0.1)
Y = np.exp(-T) * np.cos(2*np.pi*T)    # Vectorized computation!
plt.figure()                          # Creating a new figure (or activate existing)
plt.subplot(211)                      # subplot(nrows, ncols, index)
plt.plot(T, Y, 'bo',T,Y,'-',alpha=.5)
plt.ylabel("Current [mA]")            # y-label
plt.subplot(212)                      # creating index = 2
plt.plot(T, np.cos(2*np.pi*T), 'm--')
plt.xlabel("Time [s]")
plt.ylabel("Current [mA]")
plt.show()
```



Histograms and working with text

```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)     # creating 10k samples with mu 100 and std 15
# Creating histogram of the data
n, bins, patches = plt.hist(x, 50, density=1, facecolor='g', alpha=0.75)
plt.xlabel('Random variable')
plt.ylabel('Probability')
plt.title('Gaussian sample data')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')      # putting text at location (60,0.025)
plt.axis([40, 160, 0, 0.03])                      # setting the axis limits
plt.grid(True)                                    # making grid
```



Remember, use docstrings to get help!

```
In [15]: plt.axis?
Signature: plt.axis(*args, emit=True, **kwargs)
Docstring:
Convenience method to get or set some axis properties.

Call signatures::

  xmin, xmax, ymin, ymax = axis()
  xmin, xmax, ymin, ymax = axis([xmin, xmax, ymin, ymax])
  xmin, xmax, ymin, ymax = axis(option)
  xmin, xmax, ymin, ymax = axis(**kwargs)

Parameters
----------
xmin, xmax, ymin, ymax : float, optional
    The axis limits to be set.  This can also be achieved using ::

        ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))

option : bool or str
    If a bool, turns axis lines and labels on or off. If a string,
    possible values are:

    ======== ==========================================================
```

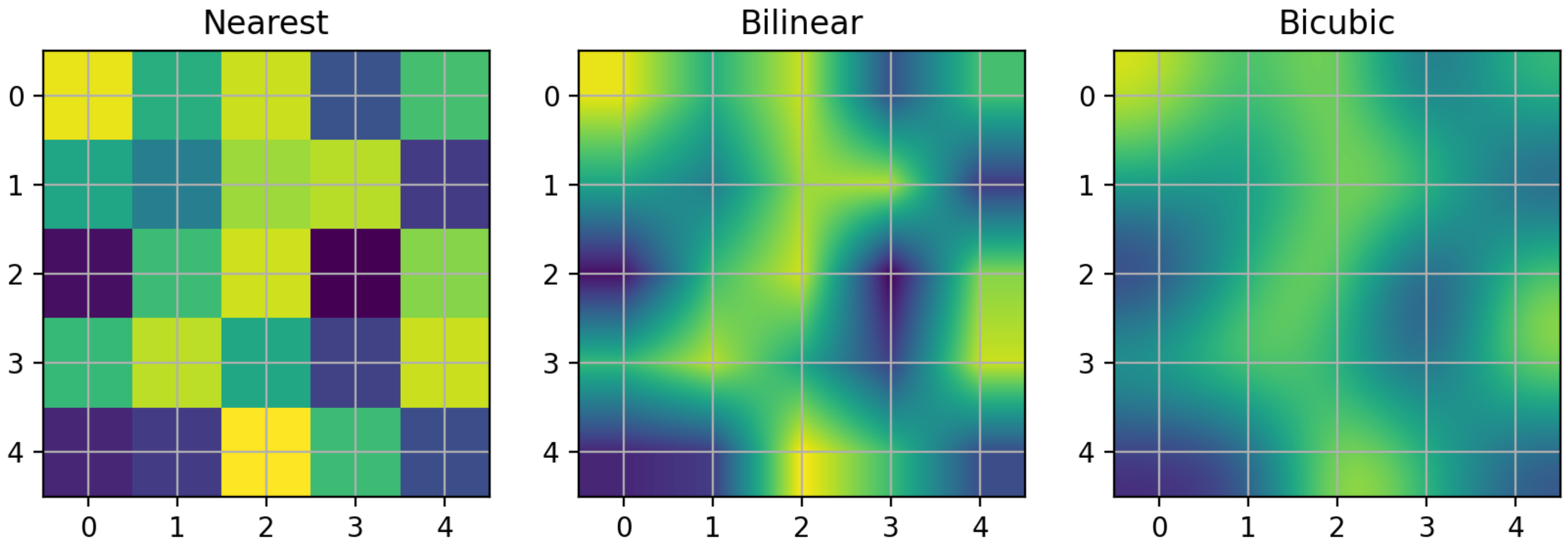
Questions?

Matplotlib has two interfaces

- `` `matplotlib.pyplot` `` is a state-based interface to matplotlib
  - this is what we saw so far
  - Pyplot tutorial
- it also has an object-oriented (OO) interface. In this case, we utilize an instance of `` `axes.Axes` `` in order to render visualizations on an instance of `` `figure.Figure` ``.
  - more details what that means with a nice example plotting financial data
  - all plots we saw so far, you can all also do this way
  - lots of examples

Plotting images `imshow`

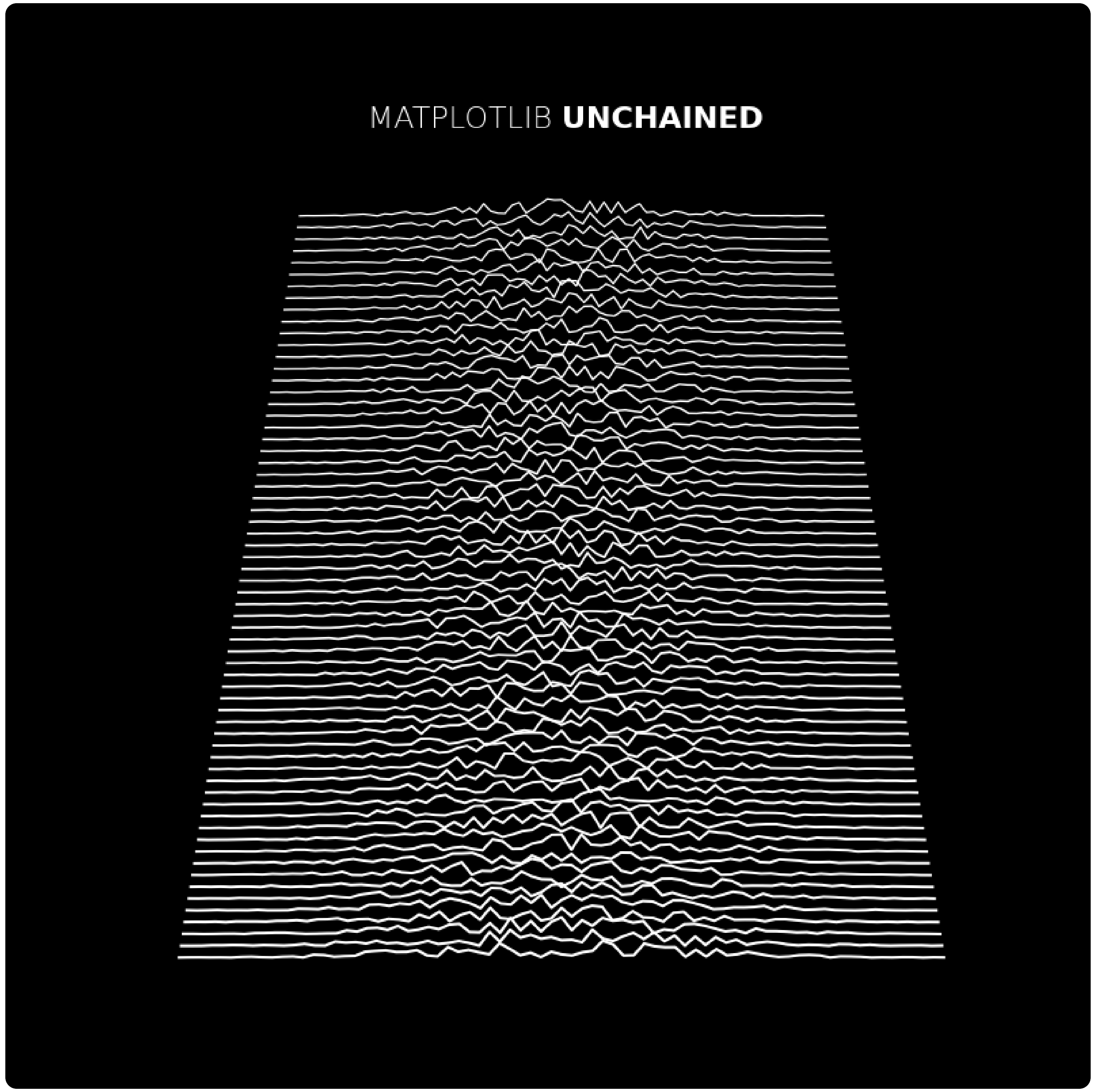
```python
A = np.random.rand(5, 5)          # creating a random 5 x 5 array (uniform)
fig, axs = plt.subplots(1, 3, figsize=(10, 3))      # creating a figure object
for ax, interp in zip(axs, ['nearest', 'bilinear', 'bicubic']):
    ax.imshow(A, interpolation=interp)              # plotting `image` A
    ax.set_title(interp.capitalize())
    ax.grid(True)

plt.show()
```



Source / also works for images (loaded as arrays)

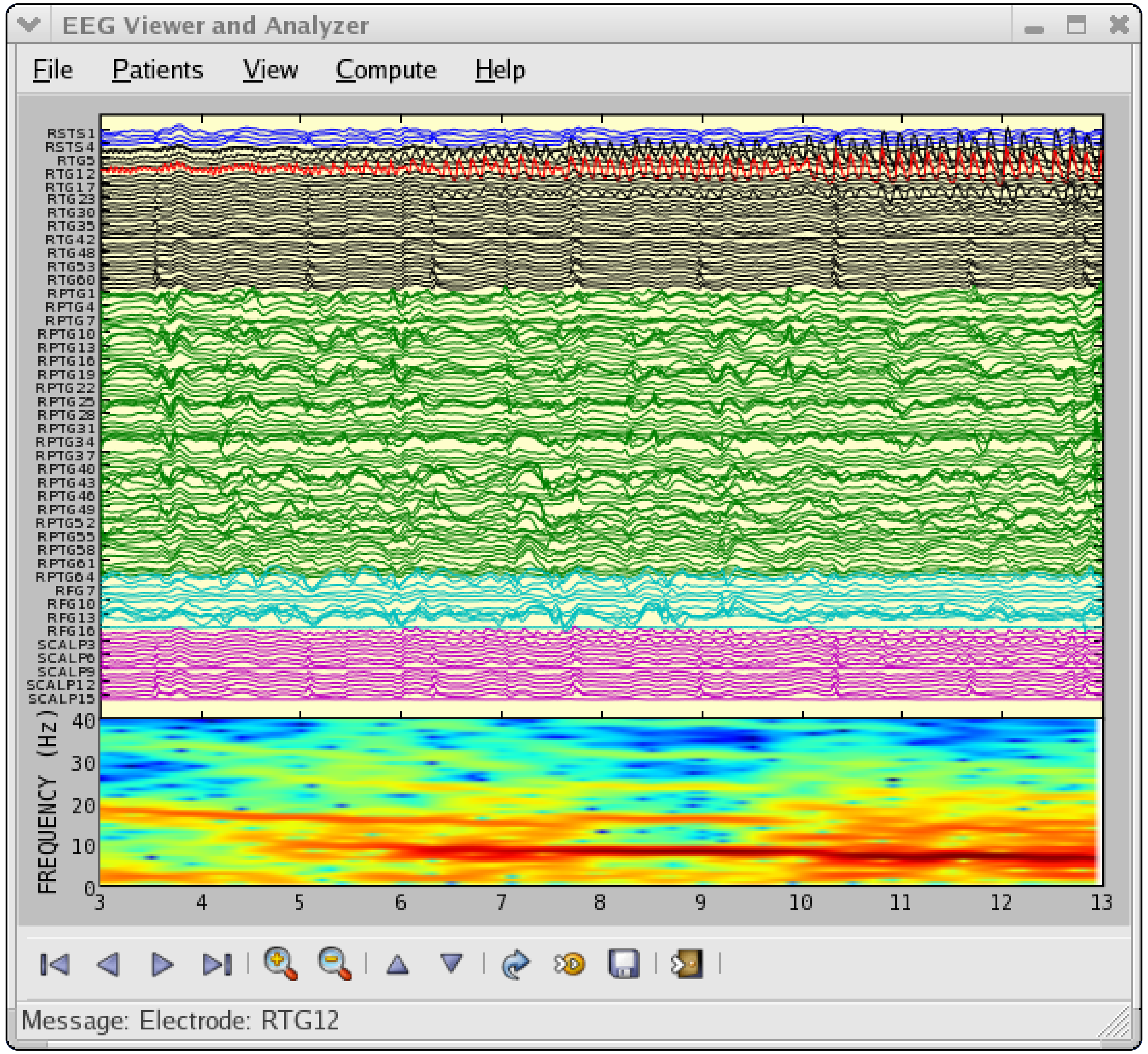
Check out the Matplotlib gallery

Tons of visual examples with code, e.g. matlab-unchained



Fun stuff I:

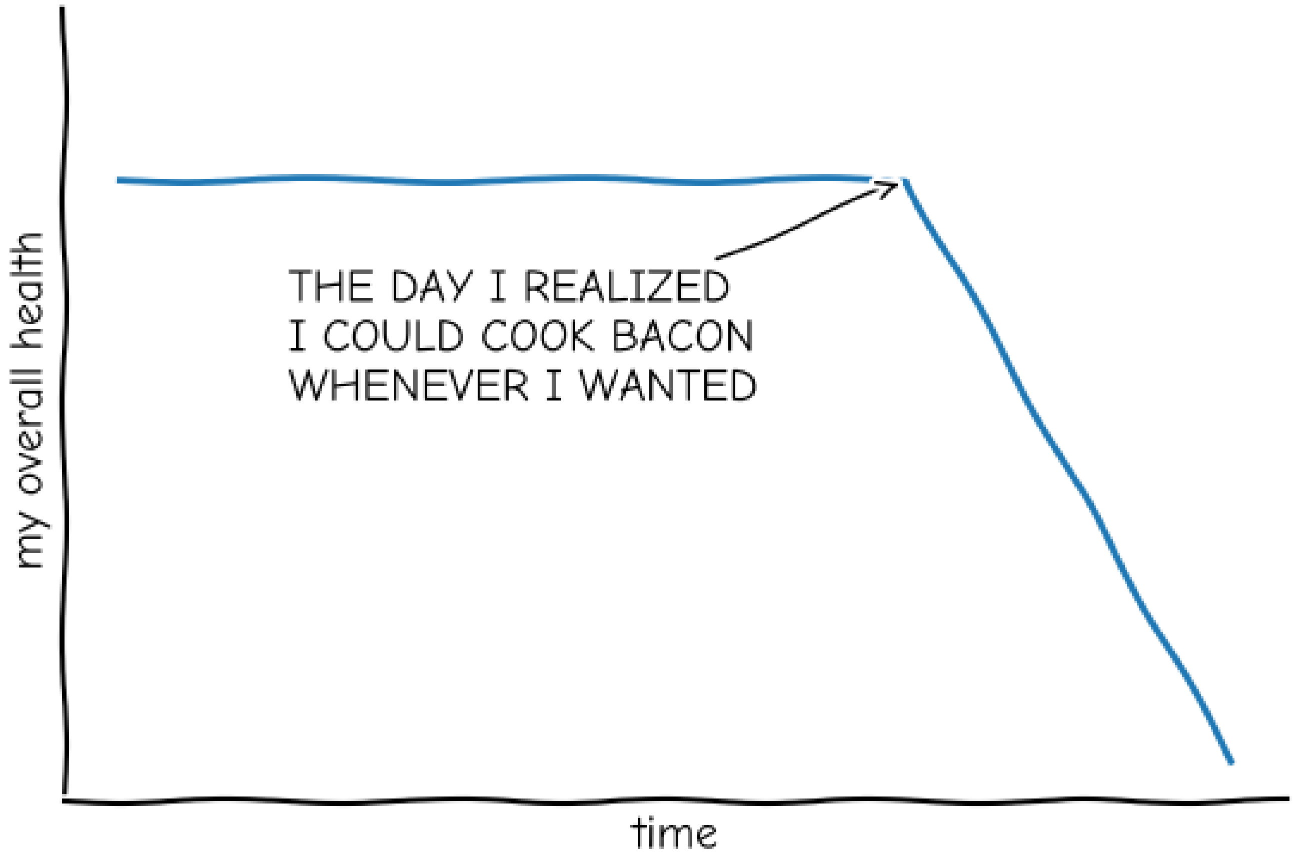
Matplotlib can be integrated in GUIs and make complex figures, e.g., here is a screenshot from pbrain



Message: Electrode: RTG12

Fun stuff II:

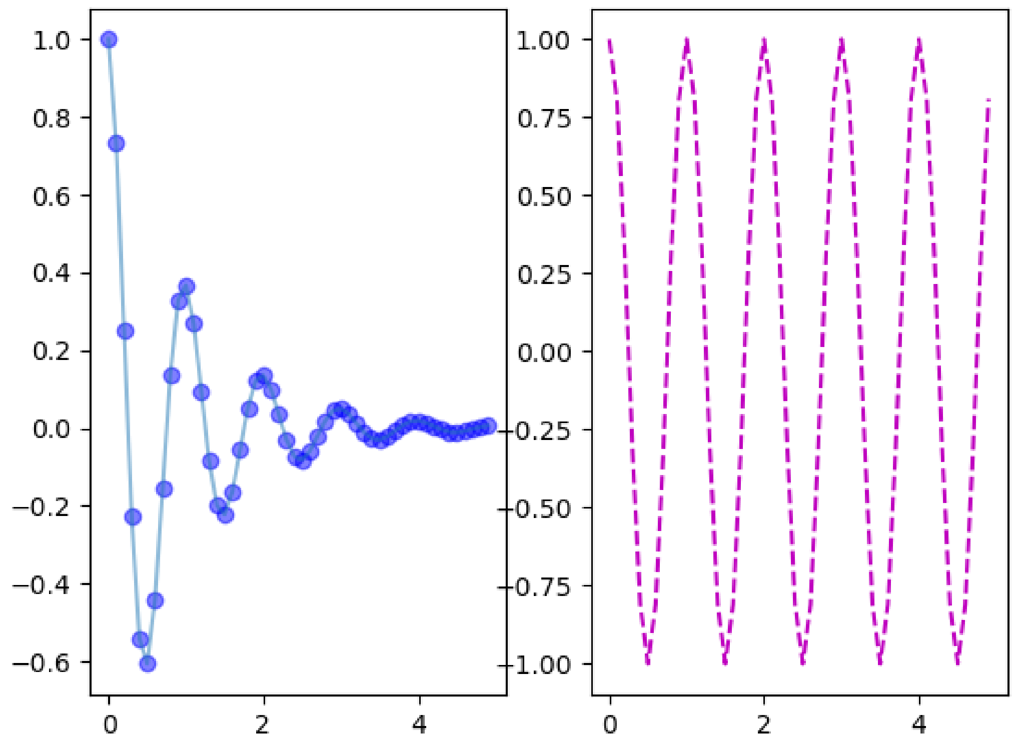
You can plot xkcd-comic style…



"Stove Ownership" from xkcd by Randall Munroe

Quiz: How do you make a plot with 2 columns and 1 row?

```
T = np.arange(0.0, 5.0, 0.1)
Y = np.exp(-T) * np.cos(2*np.pi*T)    # Vectorized computation!
plt.figure()                          # Creating a new figure (or activate existing)
plt.subplot(121)                      # subplot(nrows, ncols, index)
plt.plot(T, Y, 'bo',T,Y,'-',alpha=.5)
plt.subplot(122)                      # creating index = 2
plt.plot(T, np.cos(2*np.pi*T), 'm--')
plt.show()
```



Additional references

Remember, check out the Matplotlib gallery

- Matplotlib tutorial
- Excellent additional matplotlib resources
- Ten Simple Rules for Better Figures
- Review on Visualization of Biomedical Data

Questions?

Today's summary

- docstrings
- visualization with matplotlib

As always, try out the commands in the python shell/notebooks!

In the exercises you will add docstrings and visualizations to your project.

After lunch:

- Arrive early for the quiz (so you can start at 13:15)
- This week we will add visualizations and docstrings.
- Stay tuned for your code review, release v3 by Monday at 10 am
- Monday 15:15 - 16: my office hours at SV 2811