

COMP 204: Computer Tools for Life Sciences

Data visualization with Matplotlib

Yue Li

based on material from Mathieu Blanchette, Christopher J.F.
Cameron and Carlos G. Oliver

Matplotlib

Visualization is an important way for humans to understand data. Python programs can generate plots about the data they are handling.

This is commonly done using the Matplotlib module.

https://matplotlib.org/devdocs/api/pyplot_summary.html

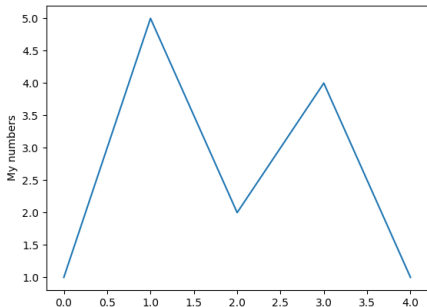
To use Matplotlib, you first need to import the module within your program.

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

This imports the functions of the matplotlib module, and gives the module a shorter name: plt.

Example 1: plot1.py

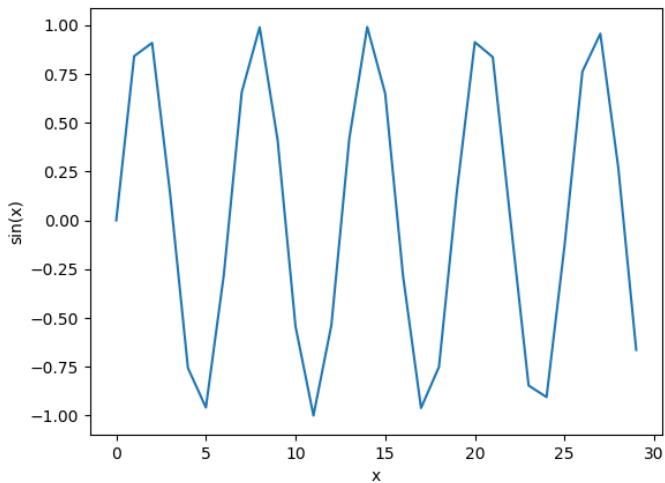
```
1 import matplotlib.pyplot as plt
2
3 my_numbers=[1,5,2,4,1]
4 plt.plot(my_numbers)
5 plt.ylabel("My numbers")
6 plt.show() # displays figure
```



Example 2: plot2.py

```
1 import matplotlib.pyplot as plt
2 import math
3
4 my_x = range(0,30)
5
6 # calculate the value of sin(x) for all x in my_x
7 # this uses the approach of list comprehension
8 my_sin = [math.sin(x) for x in my_x]
9
10 # here plot takes two arguments: the list of x
   ↪ coordinates
11 # and the list of y coordinates
12 plt.plot(my_x, my_sin)
13 plt.xlabel("x")
14 plt.ylabel("sin(x)")
15 plt.show() # displays figure
```

A second example



Saving a figure: `plt.savefig()` in `plot3.py`

With Wing101, you can save a figure when it pops-up during the program's execution.

But you can also save the figure programmatically (without the need for user intervention):

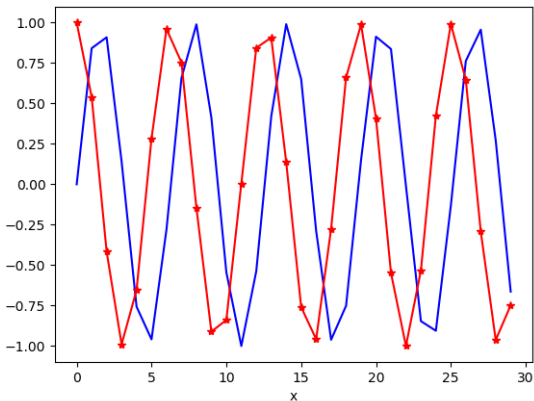
Use the `plt.savefig()` function: [online doc](#)

```
1 import matplotlib.pyplot as plt
2 import math
3
4 my_x = range(0,30)
5 my_sin = [math.sin(x) for x in my_x]
6
7 plt.plot(my_x, my_sin)
8 plt.xlabel("x")
9
10 # this won't show the figure, but will save it
11 # in a file named my_sin.png
12 plt.savefig("my_sin.png")
```

Colors and markers

We can select the color of the plots, the style/size of markers, etc.
Useful when multiple data are being plotted!

See `plot()` [documentation](#) for details.



Plotting two plots in one figure: plot4.py

```
1 import matplotlib.pyplot as plt
2 import math
3
4 my_x = range(0,30)
5 my_sin = [math.sin(x) for x in my_x]
6 my_cos = [math.cos(x) for x in my_x]
7
8 # plots my_sin with a blue line
9 plt.plot(my_x, my_sin, "b")
10
11 # plots my_cos with a red line and marker *
12 plt.plot(my_x, my_cos, "r*-")
13 plt.xlabel("x")
14
15 plt.show()
```


More about colors

Matplotlib functions can handle many different colour codes:

1. character:

- ▶ 'b': blue
- ▶ 'r': red
- ▶ 'k': black
- ▶ ...

2. RGB (Red-Green-Blue)

- ▶ (0,1,0) = green
- ▶ (1,0,1) = purple
- ▶ (0,0,0) = black
- ▶ (0.5, 0.5, 0.5) = gray

See [colors_api](#) for more information.

A more interesting example: cancer.py

Suppose you have measured the expression of 5 genes in a set of healthy patients and a set of cancer patients:

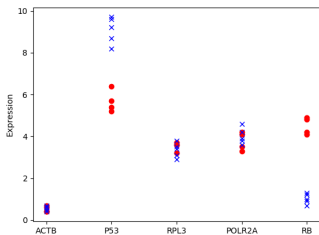
```
1 genes = ["ACTB", "P53", "RPL3", "POLR2A", "RB"]
2
3 #expression data in 4 healthy individuals
4 normals = [[0.4, 6.4, 3.2, 3.5, 4.1],
5             [0.6, 5.4, 3.6, 4.2, 4.9],
6             [0.7, 5.7, 3.7, 4.1, 4.2],
7             [0.4, 5.2, 3.6, 3.3, 4.8]]
8
9 #expression data in 5 cancer patients
10 cancer = [ [0.5, 9.2, 3.4, 3.6, 0.9],
11            [0.7, 8.7, 3.5, 4.6, 0.7],
12            [0.4, 8.2, 2.9, 4.2, 1.2],
13            [0.6, 9.7, 3.8, 3.9, 1.3],
14            [0.6, 9.6, 3.1, 3.8, 1.0]]
```

Goal: Visualize this data to learn which genes may be dysregulated in cancer.

A more interesting example: cancer.py

Idea: generate plot with x-axis = gene, y-axis = expression
Use dots of different colors for normals and cancer patients

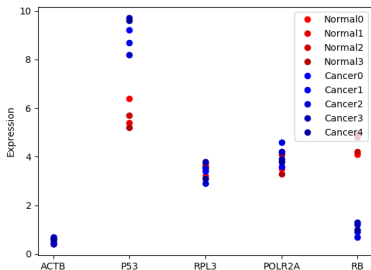
```
16 import matplotlib.pyplot as plt
17 for n in normals:
18     plt.plot(genes, n, "ro")
19 for c in cancer:
20     plt.plot(genes, c, "bx")
21
22 plt.ylabel("Expression")
23 plt.savefig("cancer1.png")
```



Add figure legend: cancer2.py

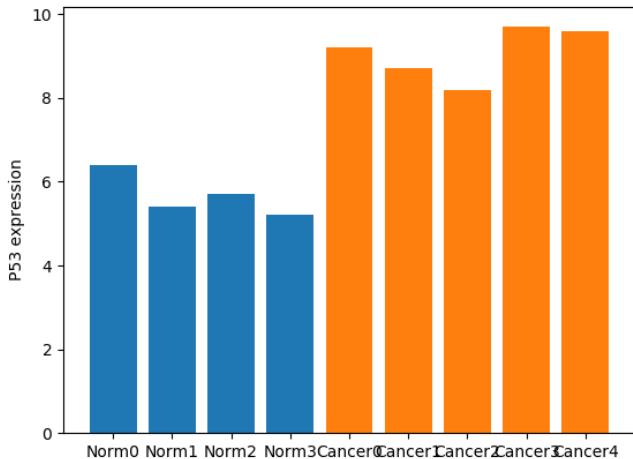
Show different individuals in different tones of red and blue

```
16 for index,n in enumerate(normals):
17     plt.plot(genes, n, "o",color=(1-0.1*index,0,0),
18             label="Normal"+str(index))
19
20 for index,c in enumerate(cancer):
21     plt.plot(genes,c, "o", color=(0,0,1-0.1*index),
22             label="Cancer"+str(index))
23 plt.ylabel("Expression")
24
25 plt.legend(loc="best") # displays legend
26 plt.savefig("cancer2.png")
```



Bar graph

Generate a bar graph of expression for P53.



Bar graph: cancer3.py

Generate a bar graph of expression for P53.

```
16 # extract data for P53
17 p53_exp_normals = [n[1] for n in normals]
18 p53_exp_cancer = [c[1] for c in cancer]
19
20 # generate identifiers for samples
21 normals_names = ["Norm"+str(i) for i in
   ↪ range(0,len(normals))]
22 cancer_names = ["Cancer"+str(i) for i in
   ↪ range(0,len(cancer))]
23
24 plt.bar(normals_names,p53_exp_normals)
25 plt.bar(cancer_names,p53_exp_cancer)
26 plt.ylabel("P53 expression")
27 plt.show()
28 plt.savefig("cancer3.png")
```

For more information

Tutorial:

<https://matplotlib.org/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py>

Documentation:

<https://matplotlib.org/devdocs/api>

Important: You don't need to know everything in Matplotlib! You just need to know how to read the document to figure out how to do what you want to do.