ÖZYEĞİN UNIVERSITY

EE393 Python for Engineers

Dr. Orhan Gökçöl
orhan.gokcol@ozyegin.edu.tr

online

30.11.2020
2020-2021 Fall Semester

1

Agenda

- Review –use of numpy & scipy for scientific/engineering computing
- NumPy
- SciPy
- An extended intro to matplotlib
  - Simple plot
  - Managing different types of plots

(Download to your local)

2

30.11.2020 | LMS resources

30 November - 6 December

- Online lecture session (30.11.2020)
- 30.11.2020 handout
- codes and data
- 30.11.2020 -lecture recording
- HW: Investigating real roots of f(x)=0 in a given interval
- Discussion forum for 30.11.2020 HW

codes and data

- figure.ipynb
- matplotlib.ipynb
- plot2.ipynb
- week9-recap.ipynb

3

Learning objectives for 23.11.2020

- Understands how to use Python in science and engineering
- Applies scipy and numpy to a wide range of engineering tasks
- Knows how to generate (x,y), histogram, pie, bar, scatter and line plots

4

MIDTERM EXAM

DECEMBER 21, 2020; 08:40

Online
(Details will be announced later)

FINAL EXAM

JANUARY 15, 2021; 09:00

Online
(Details will be announced later)

5

Last week

- numpy
- scipy
- examples

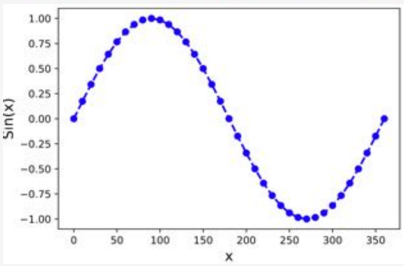
6

Scientific/Engineering Python Uses

- Extra features required:
  - fast, multidimensional arrays
    - With homogenous elements inside!!!! e.q. all numbers!
  - libraries of reliable, tested <u>**scientific functions**</u>
  - plotting tools
- **NumPy** is at the core of nearly every scientific Python application or module since it provides a fast **N-d array** datatype that can be manipulated in a vectorized form.

Finding values of complex functions and simple x-y plot

```
#------------------------
#numpy arrange and linspace are very convenient to create points (generally, x coordinates)
#then, proper numpy methods or your own functions could be used to find
#values of functions at those points. plotting values using matplotlib is also an easy task
#------------------------
import numpy as np
x = np.arange(0, 361,10)
y = np.sin(x*np.pi/180)
y = np.around(y,8)
print (x)
print (y)

#plot
import matplotlib.pyplot as plt
fig = plt.figure()
plt.plot(x,y,
         linestyle='--', linewidth=2,
         marker='o', color='b')
plt.xlabel('x', fontsize=14)
plt.ylabel('Sin(x)', fontsize=14)
fig.savefig("ee393.pdf", format='pdf', dpi=200)
plt.show()
```



week9-recap.ipynb

```
[  0  10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160 170
 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320 330 340 350
 360]
[ 0.          0.17364818  0.34202014  0.5         0.64278761  0.76604444
  0.8660254   0.93969262  0.98480775  1.          0.98480775  0.93969262
  0.8660254   0.76604444  0.64278761  0.5         0.34202014  0.17364818
  0.         -0.17364818 -0.34202014 -0.5        -0.64278761 -0.76604444
 -0.8660254  -0.93969262 -0.98480775 -1.         -0.98480775 -0.93969262
 -0.8660254  -0.76604444 -0.64278761 -0.5        -0.34202014 -0.17364818
 -0.         ]
```

Numerical integration

$$I = \int_a^b f(x)\mathrm{d}x$$

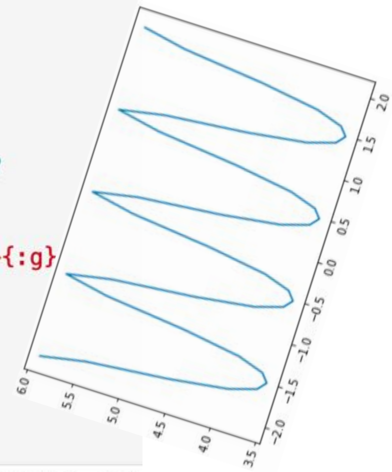
- Scientific Python provides a number of integration routines. A general purpose tool to solve integrals *I* of the kind is provided by the **quad()** function of the **scipy.integrate** module.
- It takes as input arguments the function ***f*(*x*)** to be integrated (the "integrand"), and the lower and upper limits ***a*** and ***b***.
- It returns two values (in a tuple): the first one is the computed **result**s and the second one is an estimation of the **numerical error** of that result.

recap

9

```python
from numpy import cos, exp, pi
from scipy.integrate import quad
import numpy as np

# function we want to integrate
def f(x):
    return exp(cos(-2 * x * pi)) + 3.2

# call quad to integrate f from -2 to 2
res, err = quad(f, -2, 2)

print("The numerical result is {:f} (+-{:g}"
    .format(res, err))

import matplotlib.pyplot as plt
x = np.arange(-2,2.01,0.1)
plt.plot(x,f(x))
plt.show()
```

```
The numerical result is 17.864264 (+-1.55113e-11)
```



recap

10

Self study (no grading!)

- Write a function with name **plotquad** which takes the same arguments as the quad command (*i.e. f*, *a* and *b*) and which

  (i) creates a plot of the integrand $f(x)$ and

  (ii) computes the integral numerically

using the quad function.

The return values should be as for the quad function.

recap

Differential Equations

- To solve an ordinary differential equation of the type with a given $y(t_0)=y_0$

$$\frac{dy(t)}{dt} = f(y,t)$$

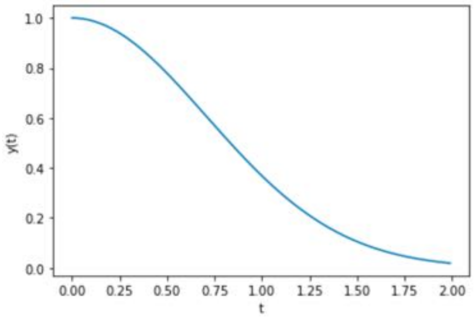
- We can use **scipy**'s **odeint** function. Here is a (self explaining) example program to find

  y(t) for t∈[0,2]

given this differential equation:

- $dy(t)/dt = -2yt$ with $y(0)=1$ (initial contition)

```
from scipy.integrate import odeint
import numpy as np

#this is the rhs of the ODE
#to integrate, i.e. dy/dt=f(y,t)
def f(y, t):
    return -2*y*t

y0 = 1    # initial value
a  = 0    # integration limits
b  = 2    # for t

t = np.arange(a, b, 0.01) # values of t for
                          # which we require
                          # the solution y(t)
# actual computation of y(t) -it is a numpy array
y = odeint(f, y0, t)

import matplotlib.pyplot as plt   # plotting of results
plt.plot(t, y)
plt.xlabel('t'); plt.ylabel('y(t)')
plt.show()
```



13

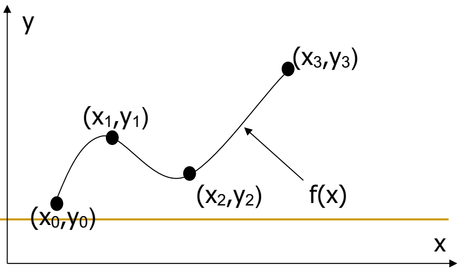
Interpolation

- Many times, data is given only at discrete points such as

$$(x_0, y_0), (x_1, y_1), (x_2, y_2), \quad \ldots\ldots\ldots\ldots \quad (x_{n-1}, y_{n-1}), \ (x_n, y_n)$$

So, how then does one find the value of *y* at any other value of *x* ?

- A continuous function *f(x)* may be used to represent the *n+1* data values with *f(x)* passing through the n+1 points.
- Then one can find the value of *y* at any other value of *x*. This is called *interpolation*.



recap

14

Polynomial interpolation?

- Of course, if new ‘x’ falls outside the range of *x* for which the data is given, it is no longer interpolation but instead is called *extrapolation*.
- So what kind of function *f(x)* should one choose? A polynomial is a common choice for an interpolating function because polynomials are easy to
  - evaluate,
  - differentiate, and
  - integrate

  relative to other choices such as a trigonometric and exponential series
- Polynomial interpolation involves finding a polynomial of order *n* that passes through the *n+1* points

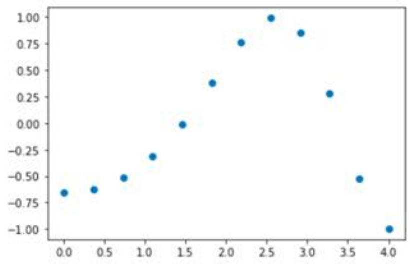
15

scipy interpolation

```
#Interpolation
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt

x = np.linspace(0, 4, 12)
y = np.cos(x**2/3+4)
print (x,y)
```

```
[0.         0.36363636 0.72727273 1.09090909 1.454545
 2.18181818 2.54545455 2.90909091 3.27272727 3.636363
5364362 -0.61966189 -0.51077021 -0.31047698 -0.007154
  0.76715099  0.99239518  0.85886263  0.27994201 -0.5
```

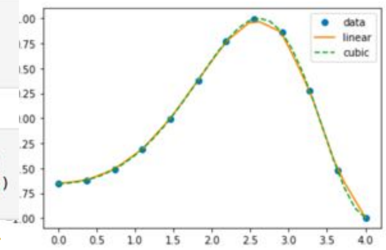
```
#plot points
plt.plot(x, y,'o')
plt.show()
```



```
#linear and cubic interpolations
f1 = interpolate.interp1d(x, y, kind = 'linear')
f2 = interpolate.interp1d(x, y, kind = 'cubic')
print (f2(3.7))
```

```
-0.6503130110905226
```

```
xnew = np.linspace(0, 4,30)
plt.plot(x, y, 'o', xnew, f1(xnew), '-', xnew, f2(xnew), '--')
plt.legend(['data', 'linear', 'cubic','nearest'], loc = 'best')
plt.show()
```



recap

16

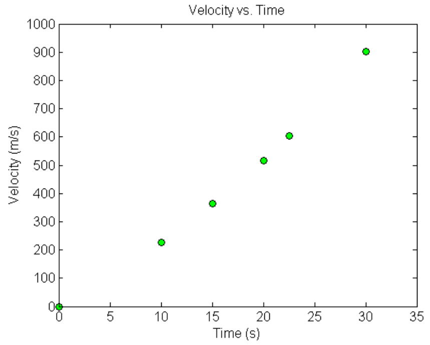
Self-study; no grading

- The upward velocity of a rocket is given as a function of time in the table given below. Corresponding graph is shown in the figure

Velocity as function of time

Time (s)	Velocity (m/s)
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67

Graph of Velocity



- Determine the value of the velocity at **<u>t = 16</u>** seconds using interpolation.

17

Root finding

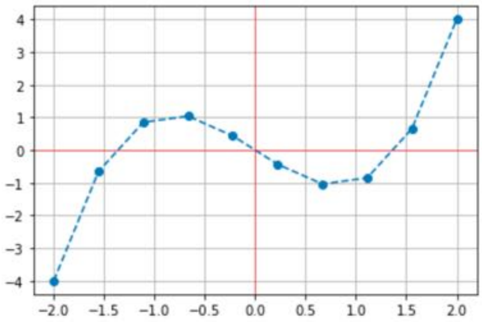
- If you try to find an x such that $f(x)=0$, then this is called *root finding*.
- Note that problems like $g(x)=h(x)$ fall in this category as you can rewrite them as
  $f(x)=g(x)-h(x)=0$.
- A number of root finding tools are available in **scipy**'s **<u>optimize</u>** module.

18

Example

- Find roots for f(x)=$x^3 - 2x^2 = 0$

```
#ROOT finding
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    return x**3 - 2*x

x = np.linspace(-2,2,10)
y = f(x)
plt.plot (x,y,"o--")
plt.axhline(y=0, color='r',
            linewidth=0.5, linestyle='-')
plt.axvline(x=0, color='r',
            linewidth=0.5, linestyle='-')
plt.grid()
plt.show()
```



p.s. there are other methods such as bisection to find a root

```
from scipy.optimize import fsolve
#we need to specify an initial value
x = fsolve(f, 3)          # one root is at x=2.0
print("The root x is approximately x=",x)
```

```
The root x is approximately x= [1.41421356]
```

19

HW (see LMS)

- Root finding algorithms require a starting point to approach the root.
- Modify the code in the previous slide so that it finds all the roots in that interval
- For this HW, for the tasks that you're required to complete, see the HW area in LMS resources for 30.11.2020

20

Common numpy functions useful for engineers

```
#common mathematical functions
x = np.arange(1,5)
result = np.sqrt(x) * np.pi
print (result)
print (np.power(2,4)) #much faster than python equivalent
print (x.max() - x.min())

#exponential & log
arr = np.array([10,8,4])
print(np.exp(arr)) #e^x
print (np.log(arr)) #ln(x), base is e
print ("e :", np.e, "pi:", np.pi)
print (np.log10(arr)) #log(x), base is 10
print (np.log2(arr)) #log(x), base is 2

#rounding
print ("\nROUNDING")
arr = np.array([20.8999,67.89899,54.23409])
print(np.around(arr,2)) #round off with 2 decimals
print(np.floor(arr)) #largest integer less than input number
print(np.ceil(arr))  #smallest integer greater than input num

#trigonometric
print ("\nTRIGONOMETRIC FUNCTIONS")
arr = np.array([0, 30, 60, 90, 120, 150, 180])
print(np.sin(arr * np.pi / 180)) #sine function
print(np.cos(arr * np.pi / 180)) #cosine
```

Algebraic
Rounding
Logarithm
Trigonometry
Complex numbers

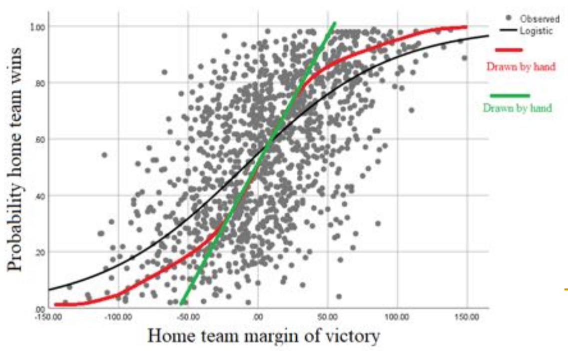
week9-recap.ipynb

See the extended list of examples in the pynb

21

Curve fitting

- **Curve fitting** is the process of constructing a curve, or mathematical function, that has the best fit to a series of data points, possibly subject to constraints.
- Curve fitting can involve either interpolation, where an exact fit to the data is required, or smoothing in which a "smooth" function is constructed that approximately fits the data. (Wikipedia)

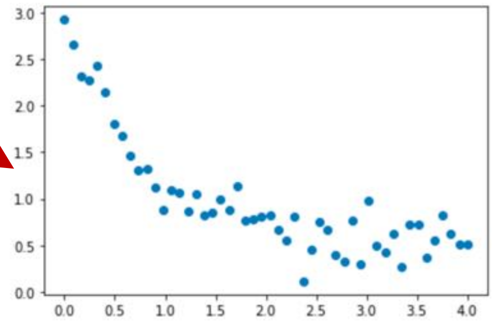


(image is from google search)

22

Example

```python
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

def f(x, a, b, c):
    """Fit function y=f(x,p) with parameters p=(a,b,c). """
    return a * np.exp(- b * x) + c

#create fake data
x = np.linspace(0, 4, 50)
y = f(x, a=2.5, b=1.3, c=0.5)
#add noise
yi = y + 0.2 * np.random.normal(size=len(x))
plt.plot(x, yi, 'o', label='data $y_i$')
plt.show()
```

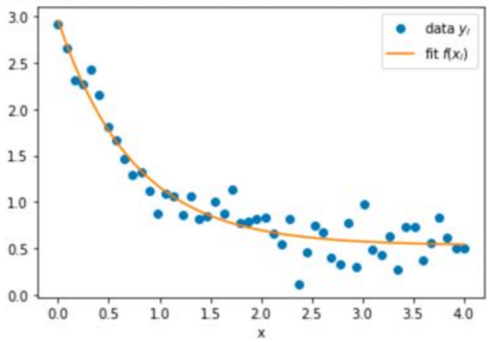


There seems to be a nice curve-like change in the data

23

```python
#call curve fit function
popt, pcov = curve_fit(f, x, yi)
a, b, c = popt
print("Optimal parameters are a={}, b={}, and c={}".format(a, b, c))

#plotting
import pylab
yfitted = f(x, *popt)   # equivalent to f(x, popt[0], popt[1], popt[2])
plt.plot(x, yi, 'o', label='data $y_i$')
plt.plot(x, yfitted, '-', label='fit $f(x_i)$')
plt.xlabel('x')
plt.legend()
plt.show()
```

```
Optimal parameters are a=2.428038004224287, b=1.349105878333664, and c=0.5306636349319742
```



LOOK!
week9-recap.ipynb

24

Linear equation systems

```
#scipy linalg
#solve linear equations systems
#importing the scipy and numpy packages
from scipy import linalg
import numpy as np

#Declaring the numpy arrays
a = np.array([[1, 3, 5], [2, 5, 1], [2, 3, 8]])
b = np.array([10, 8, 3])

#Passing the values to the solve function
x = linalg.solve(a, b)

#printing the result array
print (x)
```

```
[-9.28  5.16  0.76]
```

25

Numpy – statistics

More functions are available in **scipy**

```
#statistics
a = np.array([1, 4, 3, 8, 9, 2, 3], float)
print ("median:",np.median(a))
b = np.array([[1, 2, 1, 3], [5, 3, 1, 8]], float)
c = np.corrcoef(b)
print ("Correlation:", c)
d = np.corrcoef(a,a)
print ("Correlation:", d)
a = np.array([1,2,3,4,6,7,8,9])
b = np.array([2,4,6,8,10,12,13,15])
c = np.array([-1,-2,-2,-3,-4,-6,-7,-8])
print (np.corrcoef([a,b,c]))
print ("Covariance: ", np.cov(a)) #covariance
print ("Variance: ", np.var(a)) #covariance
print ("Standard deviation: ", np.std(a)) #covariance
```

```
median: 3.0
Correlation: [[1.         0.72870505]
 [0.72870505 1.        ]]
Correlation: [[1. 1.]
 [1. 1.]]
[[ 1.          0.99535001 -0.9805214 ]
 [ 0.99535001  1.         -0.97172394]
 [-0.9805214  -0.97172394  1.        ]]
Covariance:  8.571428571428571
Variance:  7.5
Standard deviation:  2.7386127875258306
```

26

Optimization

```python
from scipy.optimize import minimize

def eqn(x):
  return x**2 + x + 2

#use Broyden–Fletcher–Goldfarb–Shanno algorithm
mymin = minimize(eqn, 0, method='BFGS')
print(mymin)
```

```
      fun: 1.75
 hess_inv: array([[0.50000001]])
      jac: array([0.])
  message: 'Optimization terminated successfully.'
     nfev: 8
      nit: 2
     njev: 4
   status: 0
  success: True
        x: array([-0.50000001])
```

27

Matplotlib

- We'll first investigate an important python library: **matplotlib**

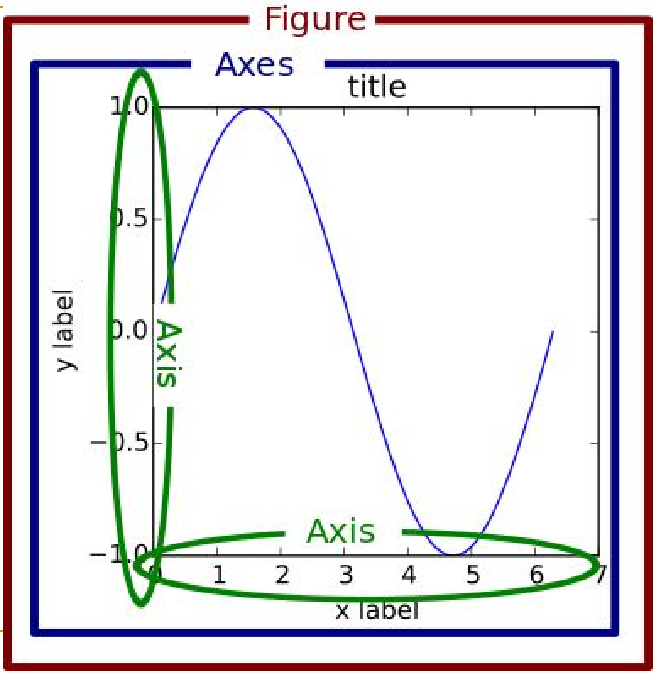
matplotlib

LOOK!

matplotlib.ipynb

28

What is a 2D Plot?



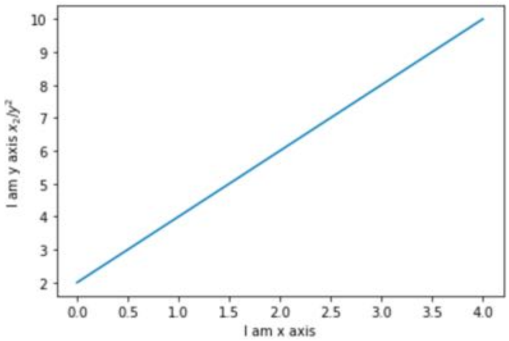
29

matplotlib

Simple Plot. The most basic plot(), with text labels

```
#simple graphics
#only y values are specieied. x becomes the index values!
plt.plot([2,4,6,8,10])
plt.ylabel('I am y axis $x_2/{y^2}$') #TeX in labels is possible
plt.xlabel('I am x axis')
plt.show()
```

Only y data is provided. x automatically becomes [0,1,2,3]



LaTeX is possible in texts

See that data is given as a list

30

matplotlib

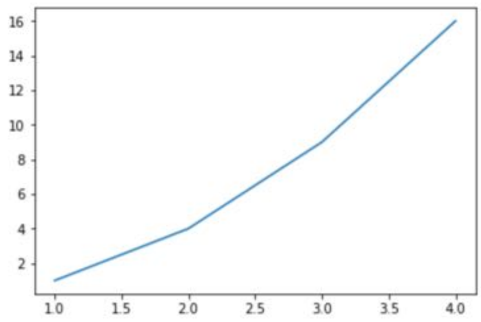
plot() is a versatile command, and will take an arbitrary number of arguments. For example, to plot x versus y, you can issue the command:

plt.plot([1,2,3,4], [1,4,9,16])

there is an optional third argument which is the format string that indicates the color and line type of the plot.

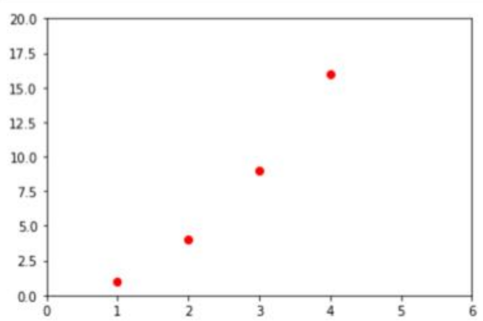
```
plt.plot([1,2,3,4], [1,4,9,16])
```

```
[<matplotlib.lines.Line2D at 0x1135c3860>]
```



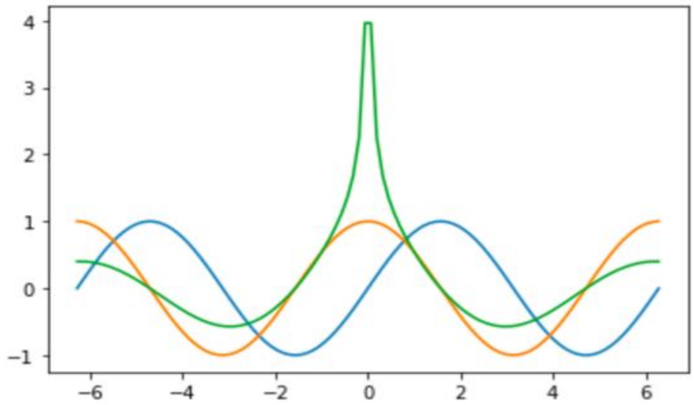
matplotlib –miscellaneous examples

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



r ➔ red
o ➔ circle

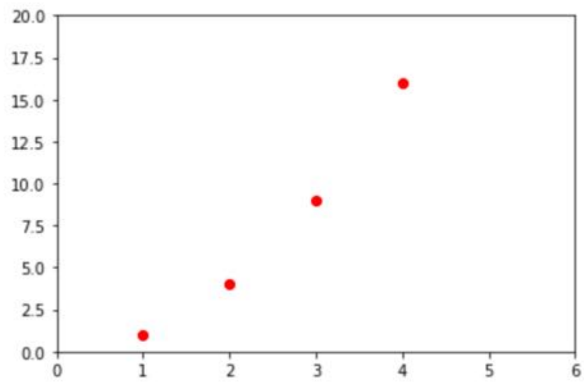
```
#many simple plots at once
import numpy as np
def f(x):
    return np.cos(x)/np.sqrt(np.abs(x))

x = np.linspace(-2*np.pi,2*np.pi,100)
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x));
plt.plot(x, f(x));
plt.show()
```



33

plot parameters -example

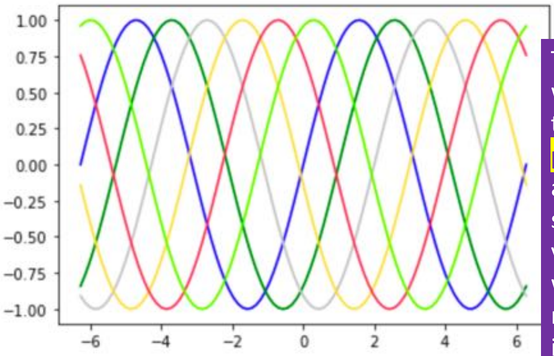
```
#plot command takes parameters to arrange many plot parameters
import matplotlib.pyplot as plt
#put o (circle) symbol on data points. Make them in red
plt.plot([1,2,3,4], [1,4,9,16], color="red", marker="o", linestyle="")
plt.axis([0, 6, 0, 20])
plt.show()
```



34

Adjusting the plot: line colors

```
#Adjusting plots
#COLORS
import matplotlib.pyplot as plt
plt.plot(x, np.sin(x - 0), color='blue')        # specify color by name
plt.plot(x, np.sin(x - 1), color='g')           # short color code (rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75')        # Grayscale between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44')     # Hex code (RRGGBB from 00 to FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 to 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
```



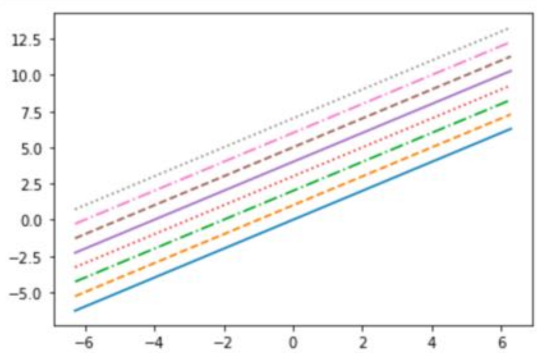
The first adjustment you might wish to make to a plot is to control the line colors and styles. The plt.plot() function takes additional arguments that can be used to specify these. To adjust the color, you can use the color keyword, which accepts a string argument representing virtually any imaginable color. The color can be specified in a variety of ways:

If no color is specified, Matplotlib will automatically cycle through a set of default colors for multiple lines.

35

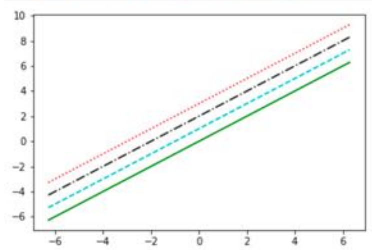
Adjusting the plot: line styles

```
#Adjusting plots
#LINE STYLES
import matplotlib.pyplot as plt
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');

# For short, you can use the following codes:
plt.plot(x, x + 4, linestyle='-')  # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':');  # dotted
```



If you would like to be extremely pythonic, these **linestyle** and **color** codes can be combined into a single non-keyword argument to the plt.plot() function:

```
#It is possible to combine line style and color
plt.plot(x, x + 0, '-g')  # solid green
plt.plot(x, x + 1, '--c') # dashed cyan
plt.plot(x, x + 2, '-.k') # dashdot black
plt.plot(x, x + 3, ':r');  # dotted red
```



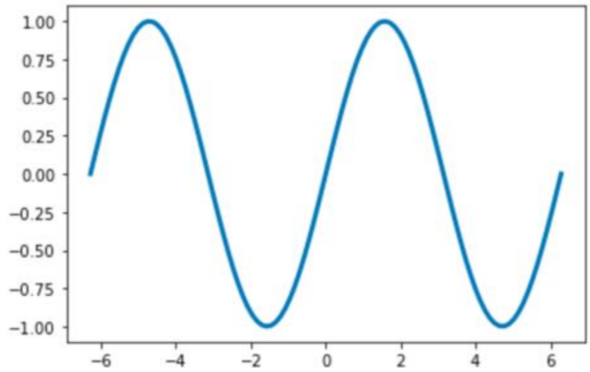
These single-character color codes reflect the standard abbreviations in the RGB (Red/Green/Blue) and CMYK (Cyan/Magenta/Yellow/blacK) color systems, commonly used for digital color graphics.

36

Adjusting the plot: line width

```
#Adjusting plots
#line width
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2*np.pi,2*np.pi,100)
plt.plot(x, np.sin(x), linewidth=3) #or, lw=3
```

```
[<matplotlib.lines.Line2D at 0x7fd3dd7cf730>]
```

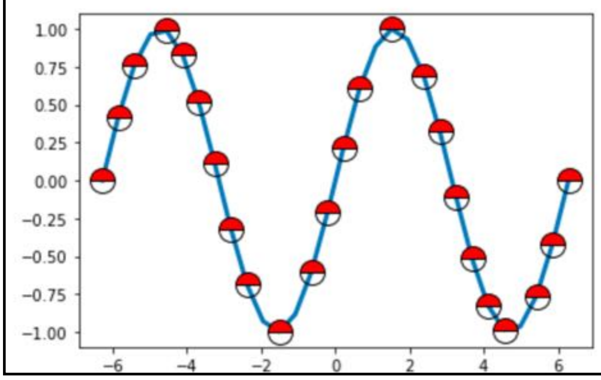


37

Adjusting the plot: marker

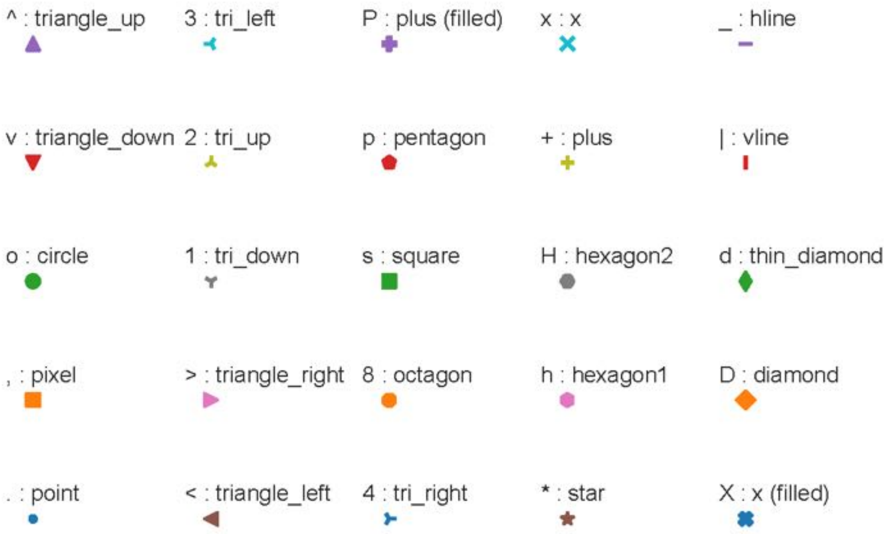
```
#Adjusting plots
#line marker
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2*np.pi,2*np.pi,30)
plt.plot(x, np.sin(x), linewidth=3,
         marker="o", markersize=16, markerfacecolor="red",
         markeredgecolor="black",
         fillstyle="top",
         markevery=0.09) #markevery has many other options
```

```
[<matplotlib.lines.Line2D at 0x7fd3dff75370>]
```



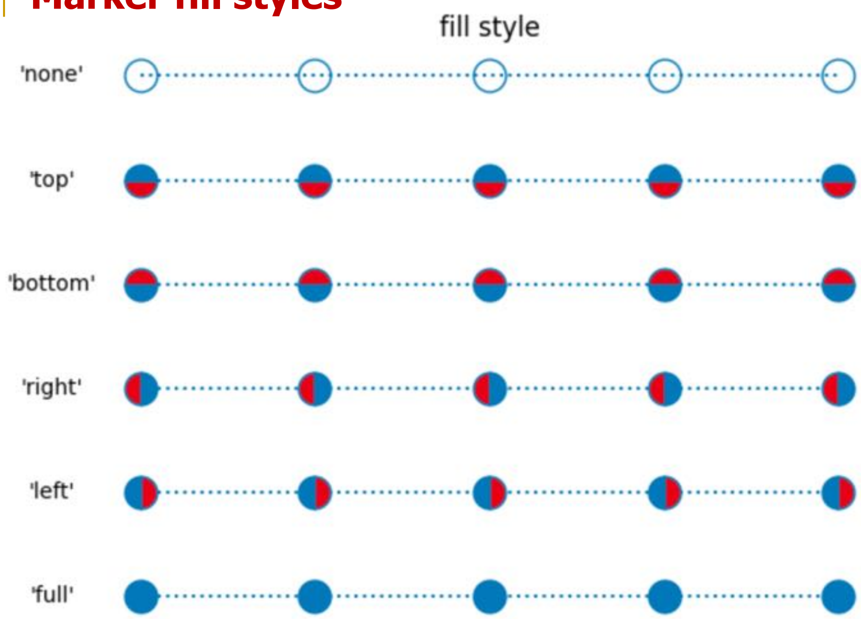
38

Marker list



39

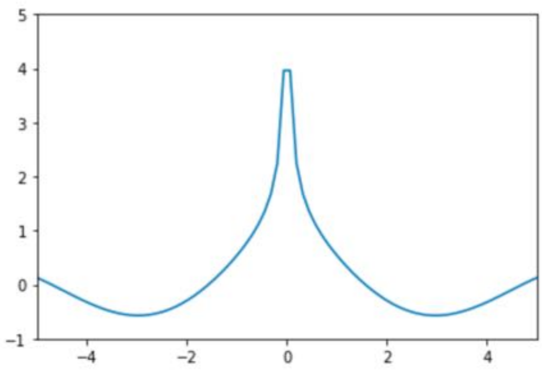
Marker fill styles



40

Adjusting the plot: axes

```
#Adjusting plots
#axes limits
import numpy as np
def f(x):
    return np.cos(x)/np.sqrt(np.abs(x))

x = np.linspace(-2*np.pi,2*np.pi,100)
plt.plot(x, f(x));
plt.xlim(-5, 5)
plt.ylim(-1, 5);
plt.show()
```



```
plt.xlim(xmin, xmax)
plt.ylim(ymin, ymax)
```

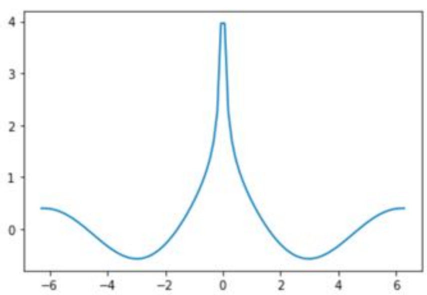
Or;

```
plt.axis(xmin,xmax,ymin,ymax)
```

41

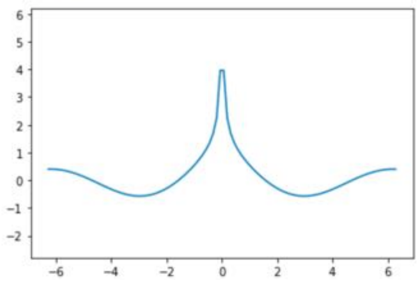
tighten the axes

```
#tighten the axes
plt.plot(x, f(x));
plt.axis('tight');
plt.show()
```



equal the axes

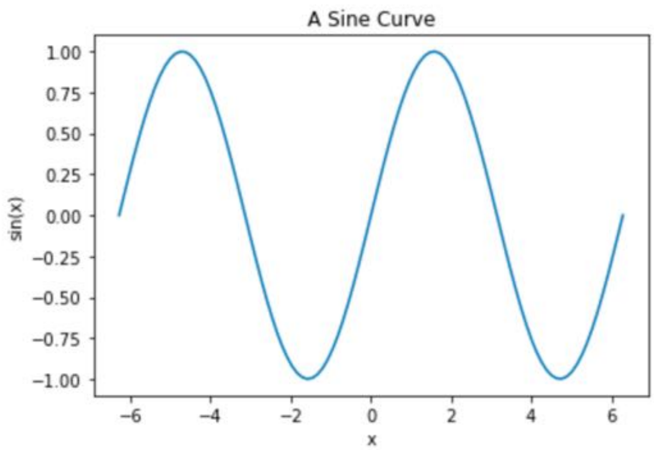
```
#equal the axes (both axes have same limits)
plt.plot(x, f(x));
plt.axis('equal');
plt.show()
```



42

Titles and axes labels

```python
#Adjusting plots
#Setting labels
plt.plot(x, np.sin(x))
plt.title("A Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)");
```



It is possible to write LaTeX compatible math for titles and labels

43

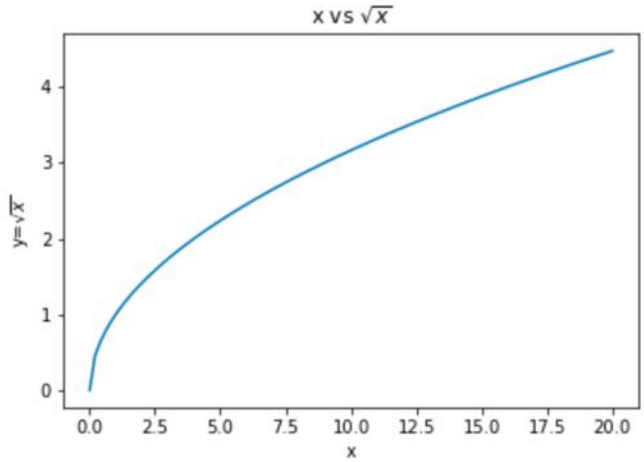
EXTRA: Using TeX to write formula

`$f(x) =\frac{d}{dx} \int_a^x f(t) \ dt$`	$f(x) = \frac{d}{dx} \int_a^x f(t)\ dt.$
`$\cos{\theta_1} + i\sin{\theta_1}$`	$\cos{\theta_1} + i\sin{\theta_1}$
`$x^n + y^n$`	$x^n + y^n$
`$\sum_{n=1}^{\infty} 2^{-n}$`	$\sum_{n=1}^{\infty} 2^{-n}$

description	code	examples
Greek letters	`\alpha \beta \gamma \rho \sigma \delta \epsilon`	$\alpha\ \beta\ \gamma\ \rho\ \sigma\ \delta\ \epsilon$
Binary operators	`\times \otimes \oplus \cup \cap`	$\times \otimes \oplus \cup \cap$
Relation operators	`< > \subset \supset \subseteq \supseteq`	$< \ > \subset \ \supset \ \subseteq \ \supseteq$
Others	`\int \oint \sum \prod`	$\int \oint \sum \prod$

44

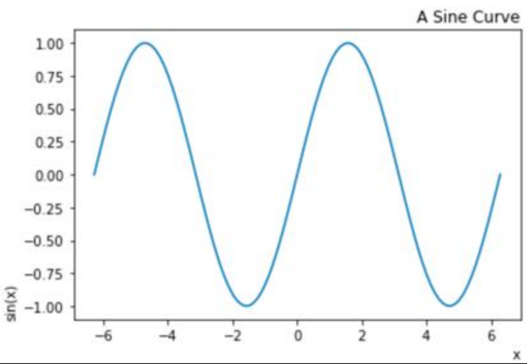
```
x = np.linspace(0,20,100)
y = np.sqrt(x)
plt.plot(x,y)
plt.title("x vs $\sqrt{x}$")
plt.xlabel("x")
plt.ylabel("y=$\sqrt{x}$")
plt.show()
```



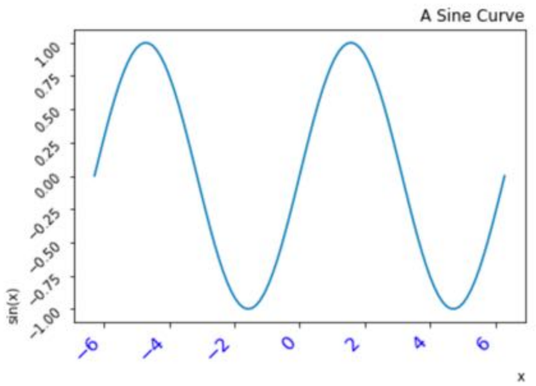
Math formula as labels and titles

The position, size, and style of these labels can be adjusted using optional arguments to the function.

```
#Adjusting plots
#Setting labels' locations
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2*np.pi,2*np.pi,100)
plt.plot(x, np.sin(x))
plt.title("A Sine Curve", loc="right") #left, right
plt.xlabel("x", loc="right") #left, right
plt.ylabel("sin(x)", loc="bottom") # top, bottom
plt.show();
```



```python
#Adjusting plots
#tickmarks
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2*np.pi,2*np.pi,100)
plt.plot(x, np.sin(x))
plt.title("A Sine Curve", loc="right") #left, right
plt.xlabel("x", loc="right") #left, right
plt.ylabel("sin(x)", loc="bottom") # top, bottom
plt.xticks(rotation=45, fontsize=14, color="b", ha="right")
plt.yticks(rotation=45);
plt.show();
```



Tick marks can be adjusted using many options.

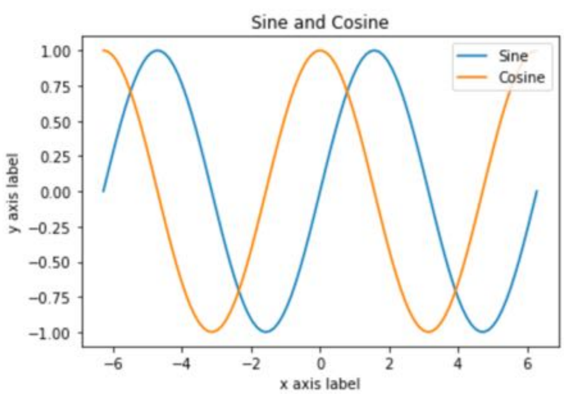
ha ➔horizontal alignment of tickmark according to value

47

Adding Legend

```python
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'], loc="upper right")
```

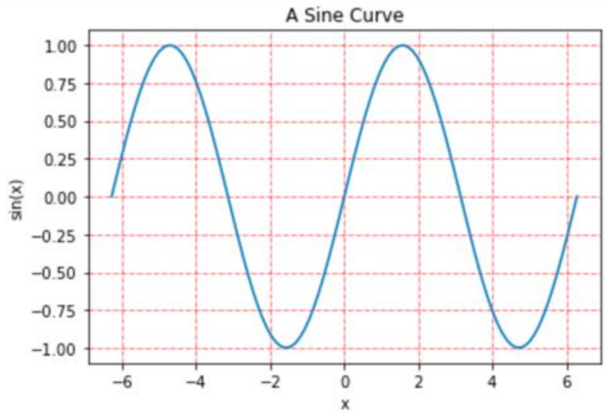
```
<matplotlib.legend.Legend at 0x7fd3e0fafdf0>
```



48

GRID

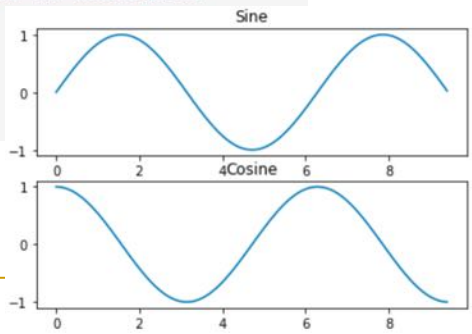
```
#Setting up grids
#Adjusting plots
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2*np.pi,2*np.pi,100)
plt.plot(x, np.sin(x))
plt.title("A Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.grid(color='r', ls = '-.', lw = 0.5)
```



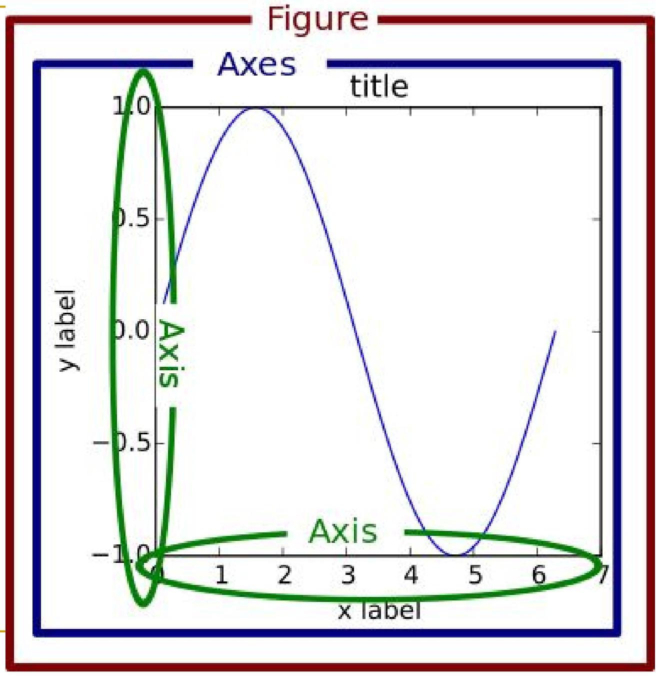
Sub plots

```
# SUBPLOTS
# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```

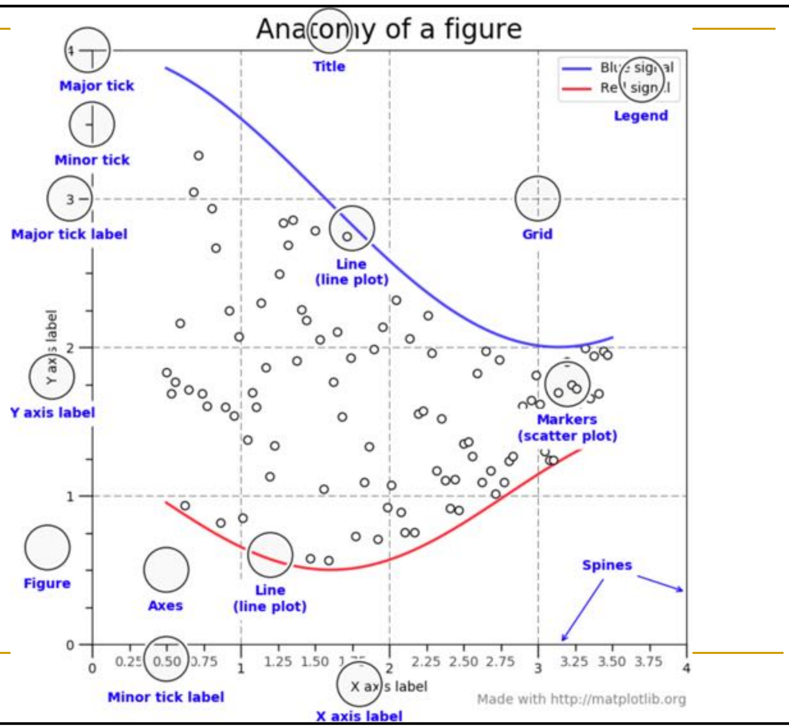
```
plt.subplot(nrows, ncols, index)
```



What is a 2D Plot?



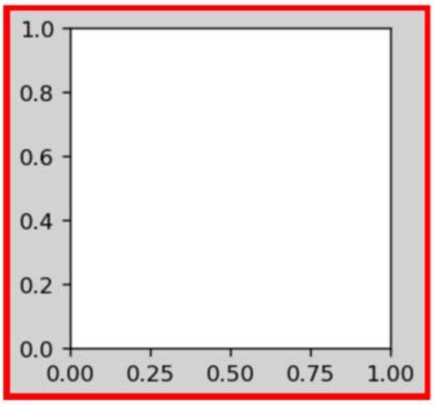
51



52

```python
#matplotlib figure
import matplotlib.pyplot as plt
import matplotlib.lines as lines

#create figure object with given properties
#fig size is in inch, given as tuble
fig = plt.figure(figsize=(2,2), dpi=120,
                 facecolor="lightgrey", edgecolor="red", linewidth=5)
#add axis x-y
#A 4-length sequence of [left, bottom, width, height] quantities.
ax=fig.add_axes([0,0,1,1])
plt.show()
```

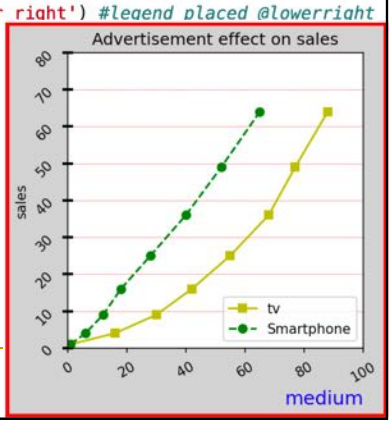
plot.ipynb



- The **matplotlib.figure** module contains the **Figure** class.
- It is a top-level container for all plot elements.
- The Figure object is instantiated by calling the **figure()** function from the pyplot module –
- The **figure()** function in **pyplot** module of matplotlib library is used to create a new figure.

53

```python
import matplotlib.pyplot as plt

#generate some data
y = [1, 4, 9, 16, 25,36,49, 64]
x1 = [1, 16, 30, 42,55, 68, 77,88]
x2 = [1,6,12,18,28, 40, 52, 65]

#create figure object
fig = plt.figure(figsize=(3,3), dpi=150,
                 facecolor="lightgrey", edgecolor="red", linewidth=5)
ax = fig.add_axes([0,0,1,1]) #axes
l1 = ax.plot(x1,y,'ys-') # solid line with yellow colour and square marker
l2 = ax.plot(x2,y,'go--') # dash line with green colour and circle marker
ax.legend(labels = ('tv', 'Smartphone'), loc = 'lower right') #legend placed @lowerright
ax.set_title("Advertisement effect on sales")
ax.set_xlabel('medium', color="b",
              fontsize=14, loc="right")
ax.set_ylabel('sales')
ax.set_xlim(0,100); ax.set_ylim(0,80)
ax.tick_params(axis='x', pad=5, rotation=30)
ax.tick_params(axis='y', pad=5, rotation=45,
               width=2, length=8, direction="inout")
ax.grid(axis="y", color='r', ls = ':', lw = 0.5)
plt.show()
```

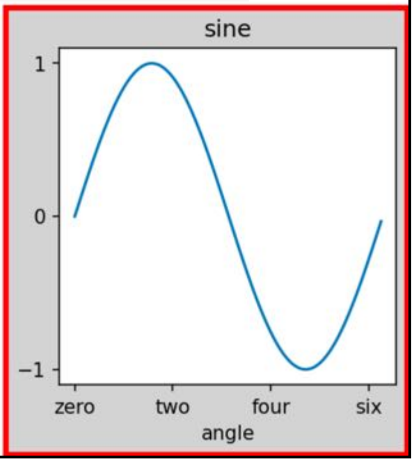


plot.ipynb

54

```python
#more on tick marks
import matplotlib.pyplot as plt
import numpy as np
import math
x = np.arange(0, math.pi*2, 0.05)
fig = plt.figure(figsize=(3,3), dpi=150,
                 facecolor="lightgrey", edgecolor="red", linewidth=5)
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
y = np.sin(x)
ax.plot(x, y)
ax.set_xlabel('angle')
ax.set_title('sine')
ax.set_xticks([0,2,4,6])
ax.set_xticklabels(['zero','two','four','six'])
ax.set_yticks([-1,0,1])
plt.show()
```

plot.ipynb

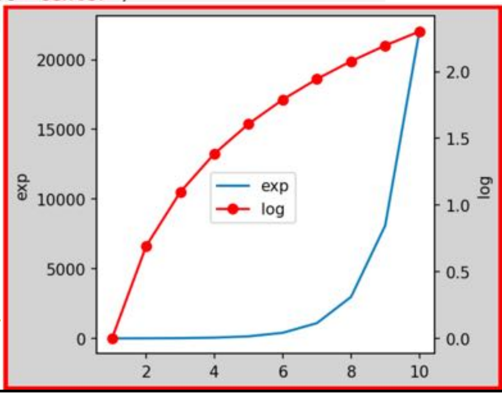


55

Twin axes

```python
#twin axes
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure(figsize=(3,3), dpi=150,
                 facecolor="lightgrey", edgecolor="red", linewidth=5)
a1 = fig.add_axes([0,0,1,1])
x = np.arange(1,11)
a1.plot(x,np.exp(x))
a1.set_ylabel('exp')
a2 = a1.twinx()
a2.plot(x, np.log(x),'ro-')
a2.set_ylabel('log')
fig.legend(labels = ('exp','log'),loc='center')
plt.show()
```

plot.ipynb

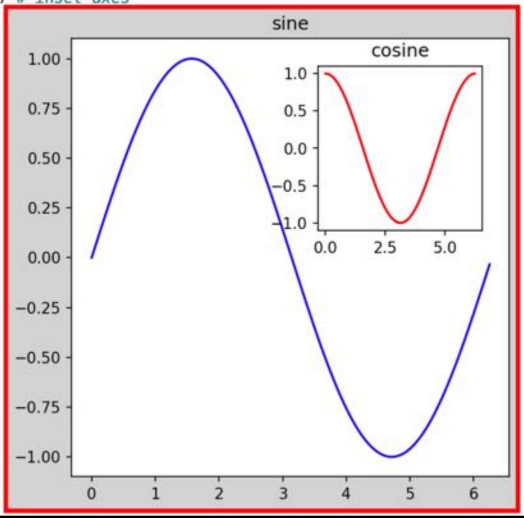


56

```
#use figure to manage multi plots
import matplotlib.pyplot as plt
import numpy as np
import math
x = np.arange(0, math.pi*2, 0.05)
#create figure object
fig = plt.figure(figsize=(5,5), dpi=150,
                 facecolor="lightgrey", edgecolor="red", linewidth=5)
axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])   # main axes
axes2 = fig.add_axes([0.55, 0.55, 0.3, 0.3]) # inset axes
y = np.sin(x)
axes1.plot(x, y, 'b')
axes2.plot(x,np.cos(x),'r')
axes1.set_title('sine')
axes2.set_title("cosine")
plt.show()
```

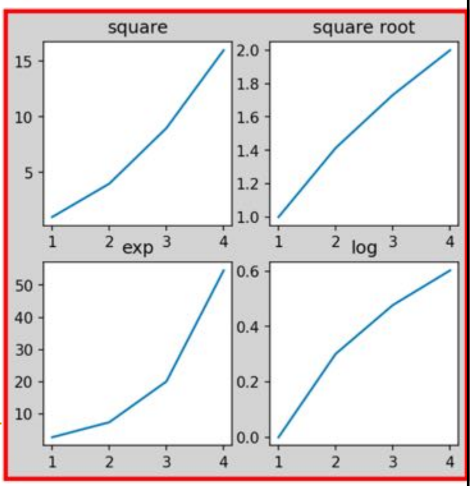
```
fig.savefig("ee393_2.png",
        format='png',
        dpi=200)
```

(to save the figure)



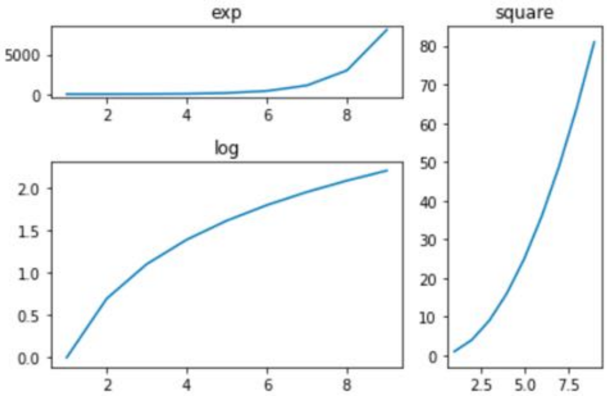
Matplotlib's **pyplot** API has a convenience function called **subplots()** which acts as a utility wrapper and helps in creating common layouts of subplots, including the enclosing figure object, in a single call.

```
#managing sub-plots
import matplotlib.pyplot as plt
import numpy as np

fig,ff =  plt.subplots(2, 2,   #2 rows, 2 columns
                  figsize=(5,5), dpi=150,
                  facecolor="lightgrey",
                  edgecolor="red", linewidth=5)
#ff is a axes subplot variable

x = np.arange(1,5) #x values
ff[0][0].plot(x,x*x) #plot at (0,0)
ff[0][0].set_title('square')
ff[0][1].plot(x,np.sqrt(x)) #plot at (0,1)
ff[0][1].set_title('square root')
ff[1][0].plot(x,np.exp(x)) #plot at (1,0)
ff[1][0].set_title('exp')
ff[1][1].plot(x,np.log10(x)) #plot at (1,1)
ff[1][1].set_title('log')
plt.show()
```



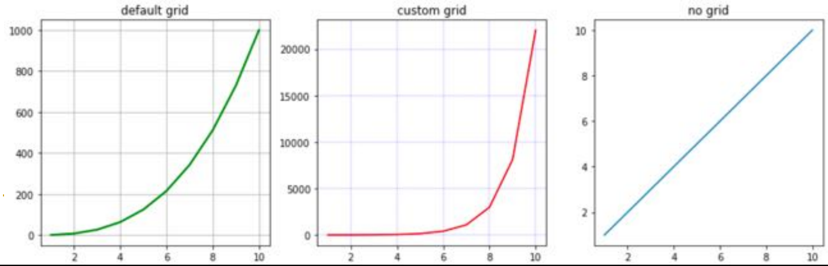
subplot2grid: This function gives more flexibility in creating an axes object at a specific location of the grid. It also allows the axes object to be spanned across multiple rows or columns. → plt.subplot2grid(shape, location, rowspan, colspan)

```
# a 3X3 grid of the figure object is filled with
# axes objects of varying sizes in row and column
# spans, each showing a different plot.
import matplotlib.pyplot as plt
a1 = plt.subplot2grid((3,3),(0,0),colspan = 2)
a2 = plt.subplot2grid((3,3),(0,2), rowspan = 3)
a3 = plt.subplot2grid((3,3),(1,0),rowspan = 2, colspan = 2)
import numpy as np
x = np.arange(1,10)
a2.plot(x, x*x)
a2.set_title('square')
a1.plot(x, np.exp(x))
a1.set_title('exp')
a3.plot(x, np.log(x))
a3.set_title('log')
plt.tight_layout()
plt.show()
```



59

```
#arranging grids in multi plots
import matplotlib.pyplot as plt
import numpy as np
fig, axes = plt.subplots(1,3, figsize = (12,4))
x = np.arange(1,11)
axes[0].plot(x, x**3, 'g',lw=2)
axes[0].grid(True)
axes[0].set_title('default grid')
axes[1].plot(x, np.exp(x), 'r')
axes[1].grid(color='b', ls = '-.', lw = 0.25)
axes[1].set_title('custom grid')
axes[2].plot(x,x)
axes[2].set_title('no grid')
fig.tight_layout()
plt.show()
```

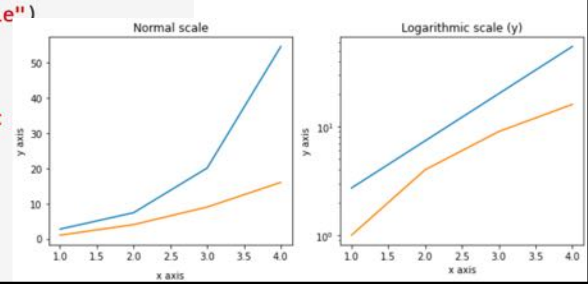


60

Formatting Axes: (EXTRA!)

- Sometimes, one or a few points are much larger than the bulk of data. In such a case, the scale of an axis needs to be set as logarithmic rather than the normal scale. This is the Logarithmic scale. In Matplotlib, it is possible by setting xscale or yscale property of axes object to '**log**'.
- It is also required sometimes to show some additional distance between axis numbers and axis label. The labelpad property of either axis (x or y or both) can be set to the desired value.
- Both the above features are demonstrated with the help of the following example. The subplot on the right has a logarithmic scale and one on left has its x axis having label at more distance.

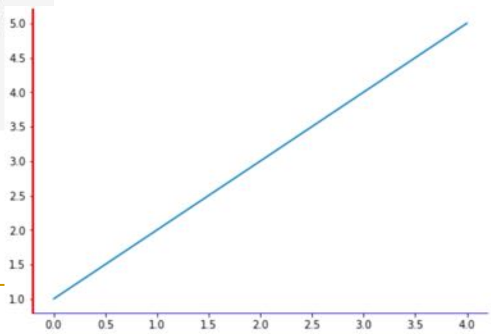
```
#working with axes
#(taken from tutorialpoints)
import matplotlib.pyplot as plt
import numpy as np
fig, axes = plt.subplots(1, 2, figsize=(10,4))
x = np.arange(1,5)
axes[0].plot( x, np.exp(x))
axes[0].plot(x,x**2)
axes[0].set_title("Normal scale")
axes[1].plot (x, np.exp(x))
axes[1].plot(x, x**2)
axes[1].set_yscale("log")
axes[1].set_title("Logarithmic
axes[0].set_xlabel("x axis")
axes[0].set_ylabel("y axis")
axes[0].xaxis.labelpad = 10
axes[1].set_xlabel("x axis")
axes[1].set_ylabel("y axis")
plt.show()
```



61

- **Axis spines** are the lines connecting axis tick marks demarcating boundaries of plot area.
- The axes object has spines located at top, bottom, left and right.
- Each spine can be formatted by specifying color and width. Any edge can be made invisible if its color is set to none.

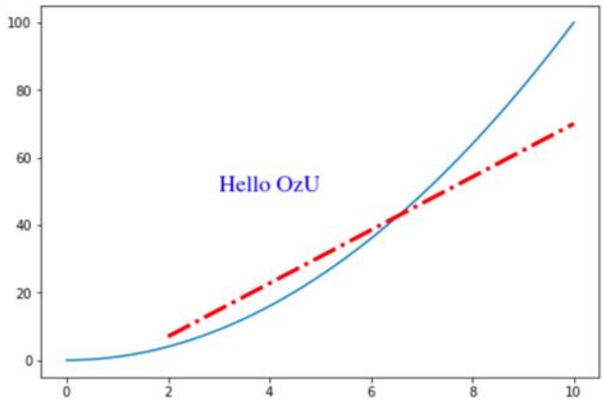
```
#axis spines
#(taken from tutorialpoints)
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.spines['bottom'].set_color('blue')
ax.spines['left'].set_color('red')
ax.spines['left'].set_linewidth(2)
ax.spines['right'].set_color(None)
ax.spines['top'].set_color(None)
ax.plot([1,2,3,4,5])
plt.show()
```



62

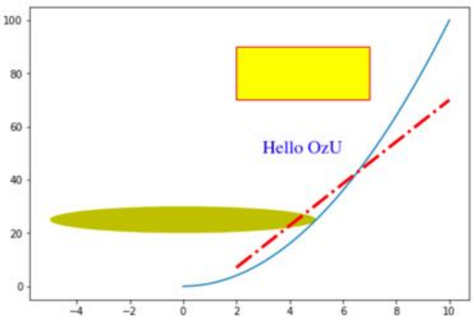
Adding lines and text

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])

x = np.linspace(0,10); y = x**2
p1 = [2,10]; p2 = [7,70]
ll=mlines.Line2D(p1, p2, color="r", ls = '-.', lw = 3)
ax.text(3,50, "Hello OzU", fontsize=18, fontfamily="Times", color="b")
ax.plot(x, y)
ax.add_line(ll)
plt.show()
```



63

add_patch : adding shapes



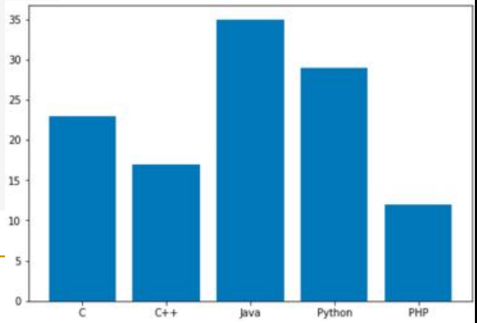

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
import matplotlib.patches as patches

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])

x = np.linspace(0,10); y = x**2
p1 = [2,10]; p2 = [7,70]
ll=mlines.Line2D(p1, p2, color="r", ls = '-.', lw = 3)
ax.text(3,50, "Hello OzU", fontsize=18, fontfamily="Times", color="b")
ax.plot(x, y)
ax.add_line(ll)

rect = patches.Rectangle((2,70),5,20,linewidth=1,edgecolor='r',facecolor='yellow')
circ = patches.Circle((0,25), 5, fc='y')
ax.add_patch(rect)
ax.add_patch(circ)
plt.show()
```

64

BAR PLOT: A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.
A bar graph shows comparisons among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value.

Matplotlib API provides the **bar()** function that can be used in the MATLAB style use as well as object oriented API. The signature of bar() function to be used with axes object is as follows : **ax.bar (x, height, width, bottom, align)**

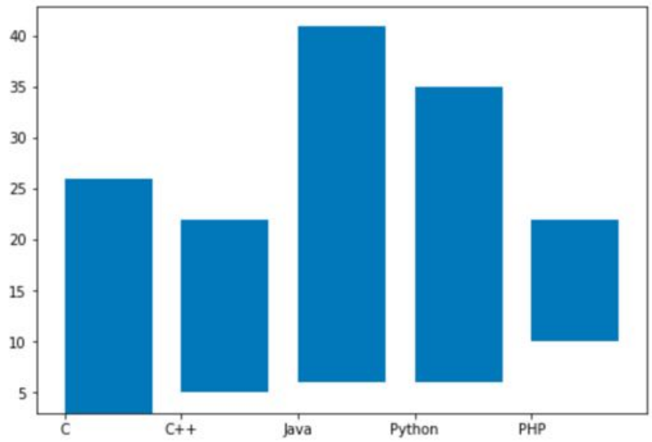
```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
langs = ['C', 'C++', 'Java',
         'Python', 'PHP']
students = [23,17,35,29,12]
ax.bar(langs,students)
plt.show()
```



LOOK!
plot2.ipynb

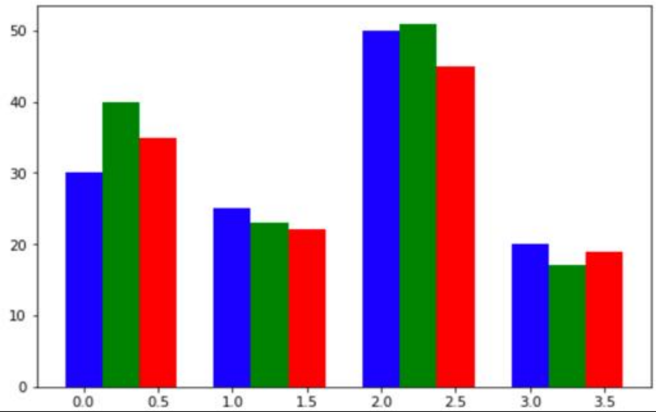
65

```
#bar plot example
#taken from tutorial point
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
langs = ['C', 'C++', 'Java',
         'Python', 'PHP']
students = [23,17,35,29,12]
ax.bar(langs,students, width=0.75, bottom=[3,5,6,6,10], align="edge" )
plt.show()
```



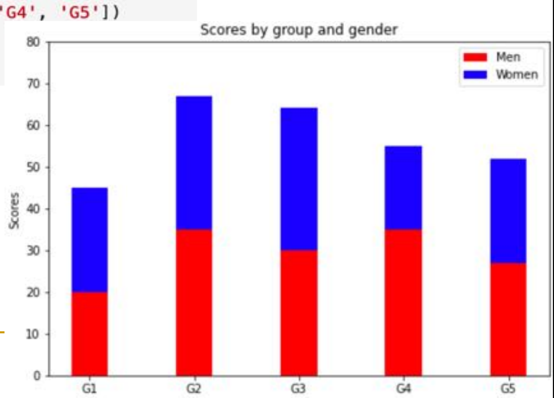
66

```
#bar chart series
import numpy as np
import matplotlib.pyplot as plt
data = [[30, 25, 50, 20],
[40, 23, 51, 17],
[35, 22, 45, 19]]
X = np.arange(4)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(X + 0.00, data[0], color = 'b', width = 0.25)
ax.bar(X + 0.25, data[1], color = 'g', width = 0.25)
ax.bar(X + 0.50, data[2], color = 'r', width = 0.25)
plt.show()
```



67

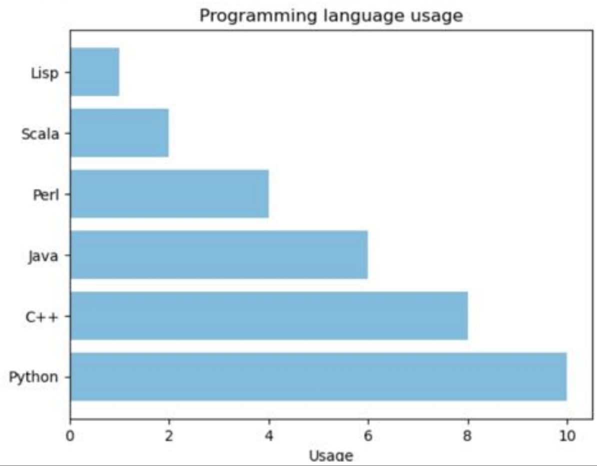
```
#stacked bar plots
import numpy as np
import matplotlib.pyplot as plt
N = 5
menMeans = (20, 35, 30, 35, 27)
womenMeans = (25, 32, 34, 20, 25)
ind = np.arange(N) # the x locations for the groups
width = 0.35
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(ind, menMeans, width, color='r')
ax.bar(ind, womenMeans, width,bottom=menMeans, color='b')
ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.set_xticks([0,1,2,3,4])
ax.set_xticklabels(['G1', 'G2', 'G3', 'G4', 'G5'])
ax.set_yticks(np.arange(0, 81, 10))
ax.legend(labels=['Men', 'Women'])
plt.show()
```



68

Horizontal bar plot

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

objects = ('Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp')
y_pos = np.arange(len(objects))
performance = [10,8,6,4,2,1]

plt.barh(y_pos, performance, align='center', alpha=0.5)
plt.yticks(y_pos, objects)
plt.xlabel('Usage')
plt.title('Programming language usage')

plt.show()
```



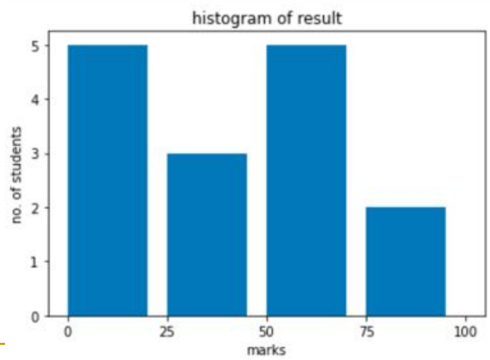
plot2.ipynb

69

HISTOGRAM

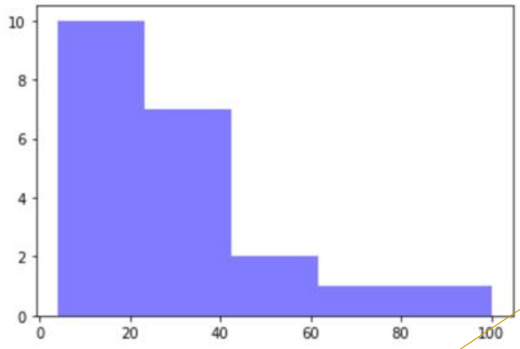
A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable. It is a kind of bar graph.
To construct a histogram, follow these steps –

```
#histogram
from matplotlib import pyplot as plt
import numpy as np
fig,ax = plt.subplots(1,1)
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
ax.hist(a, bins = [0,25,50,75,100], width = 20)
ax.set_title("histogram of result")
ax.set_xticks([0,25,50,75,100])
ax.set_xlabel('marks')
ax.set_ylabel('no. of students')
plt.show()
```



70

```python
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

x = [21,22,23,4,5,6,77,8,9,10,31,32,33,34,35,36,37,18,49,50,100]
num_bins = 5
n, bins, patches = plt.hist(x, num_bins, facecolor='blue', alpha=0.5)
plt.show()
print (n)
print (bins)
print (patches)
```



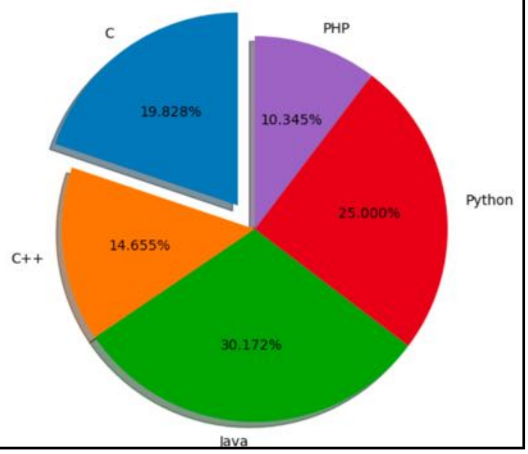
Data on the histogram plot

```
[10.  7.  2.  1.  1.]
[  4.   23.2  42.4  61.6  80.8 100. ]
<BarContainer object of 5 artists>
```

71

A **Pie Chart** can only display one series of data. Pie charts show the size of items (called wedge) in one data series, proportional to the sum of the items. The data points in a pie chart are shown as a percentage of the whole pie.

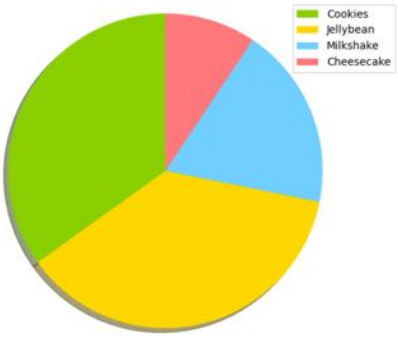
```python
#pie chart
from matplotlib import pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]

explode = (0.15, 0, 0, 0,0)  # explode 1st slice
ax.pie(students, explode, labels = langs,autopct='%1.3f%%',
       shadow=True, startangle=90
      )
plt.show()
```



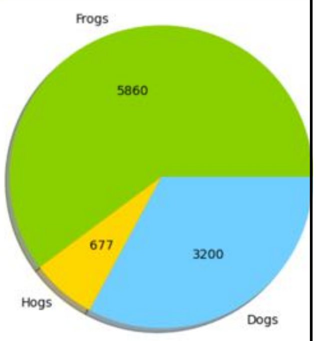
72

Examples

```python
import matplotlib.pyplot as plt

labels = ['Cookies', 'Jellybean', 'Milkshake', 'Cheesecake']
sizes = [38.4, 40.6, 20.7, 10.3]
colors = ['yellowgreen', 'gold', 'lightskyblue',
          'lightcoral']
patches, texts = plt.pie(sizes, colors=colors,
                         shadow=True, startangle=90)
plt.legend(patches, labels, loc="best")
plt.axis('equal')
plt.tight_layout()
plt.show()
```

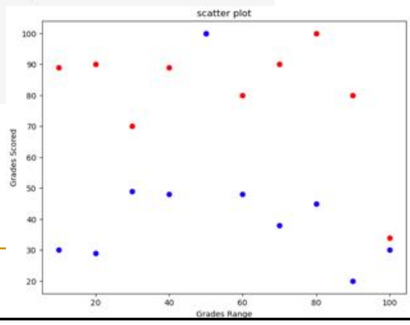



```python
import matplotlib.pyplot as plt
import numpy

labels = ['Frogs', 'Hogs', 'Dogs']
sizes = numpy.array([5860, 677, 3200])
colors = ['yellowgreen', 'gold', 'lightskyblue']

def absolute_value(val):
    a  = numpy.round(val/100.*sizes.sum(), 0)
    return int(a)

plt.pie(sizes, labels=labels, colors=colors,
        autopct=absolute_value, shadow=True)

plt.axis('equal')
plt.show()
```



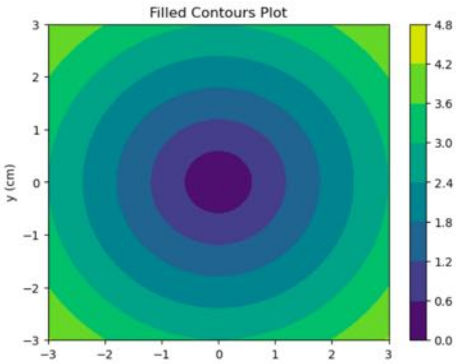
Scatter plots are used to plot data points on horizontal and vertical axis in the attempt to show how much one variable is affected by another. Each row in the data table is represented by a marker the position depends on its values in the columns set on the X and Y axes. A third variable can be set to correspond to the color or size of the markers, thus adding yet another dimension to the plot.

```python
import matplotlib.pyplot as plt
girls_grades = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
boys_grades = [30, 29, 49, 48, 100, 48, 38, 45, 20, 30]
grades_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.scatter(grades_range, girls_grades, color='r')
ax.scatter(grades_range, boys_grades, color='b')
ax.set_xlabel('Grades Range')
ax.set_ylabel('Grades Scored')
ax.set_title('scatter plot')
plt.show()
```



Contour plots (sometimes called Level Plots) are a way to show a three-dimensional surface on a two-dimensional plane. It graphs two predictor variables X Y on the y-axis and a response variable Z as contours.
A contour plot is appropriate if you want to see how value Z changes as a function of two inputs X and Y, such that **Z = f(X,Y).** A contour line or isoline of a function of two variables is a curve along which the function has a constant value.
The independent variables x and y are usually restricted to a regular grid called meshgrid. The **numpy.meshgrid** creates a rectangular grid out of an array of x values and an array of y values.

```
#contour plot example
import numpy as np
import matplotlib.pyplot as plt
xlist = np.linspace(-3.0, 3.0, 100)
ylist = np.linspace(-3.0, 3.0, 100)
X, Y = np.meshgrid(xlist, ylist)
Z = np.sqrt(X**2 + Y**2)
fig,ax=plt.subplots(1,1)
cp = ax.contourf(X, Y, Z)
fig.colorbar(cp) # Add a colorbar to a plot
ax.set_title('Filled Contours Plot')
#ax.set_xlabel('x (cm)')
ax.set_ylabel('y (cm)')
plt.show()
```



75

SEE YOU NEXT WEEK!!!

DR. ORHAN GÖKÇÖL
gokcol@gmail.com
orhan.gokcol@ozyegin.edu.tr

76