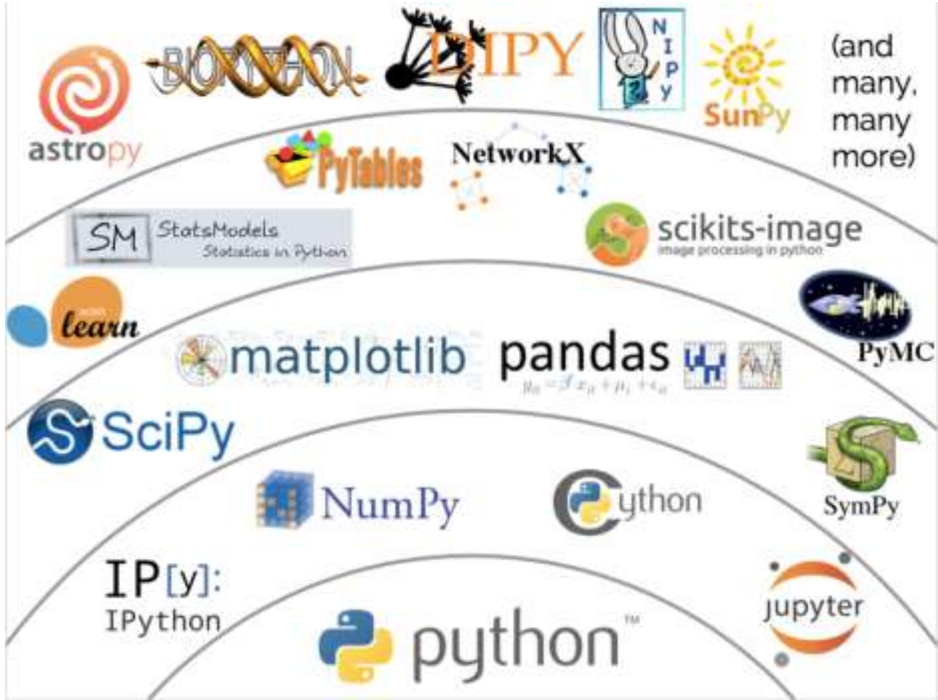
Python-Matplotlib

Dr. Sarwan Singh

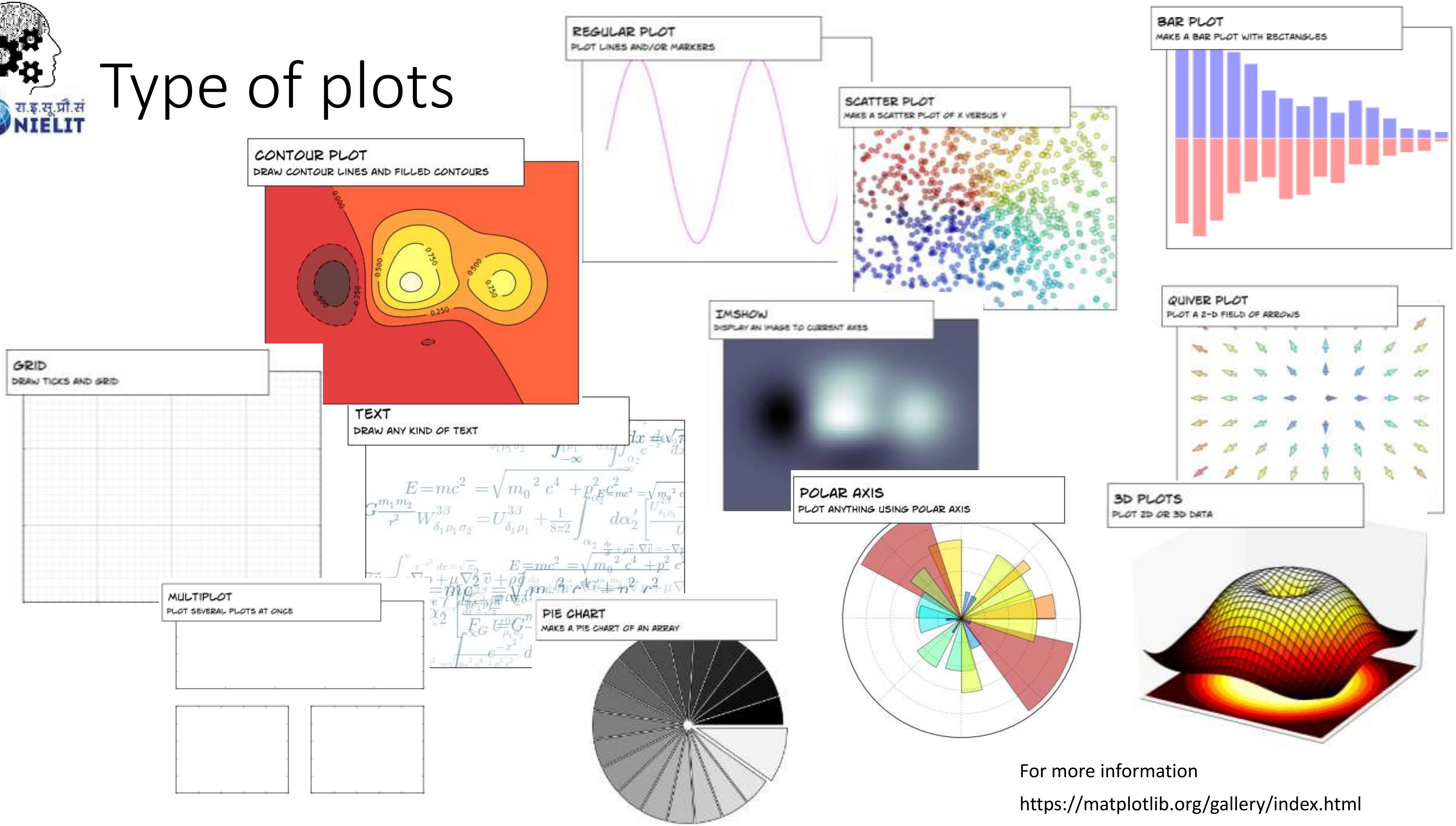
Agenda

- Introduction
- History, usage
- Adjusting the Plot: Line Colors, style, etc.
- Scatter plot
- Density and Contour Plots
- Visualizing 3D



One guiding principle of Python code is that
“explicit is better than implicit”

Type of plots

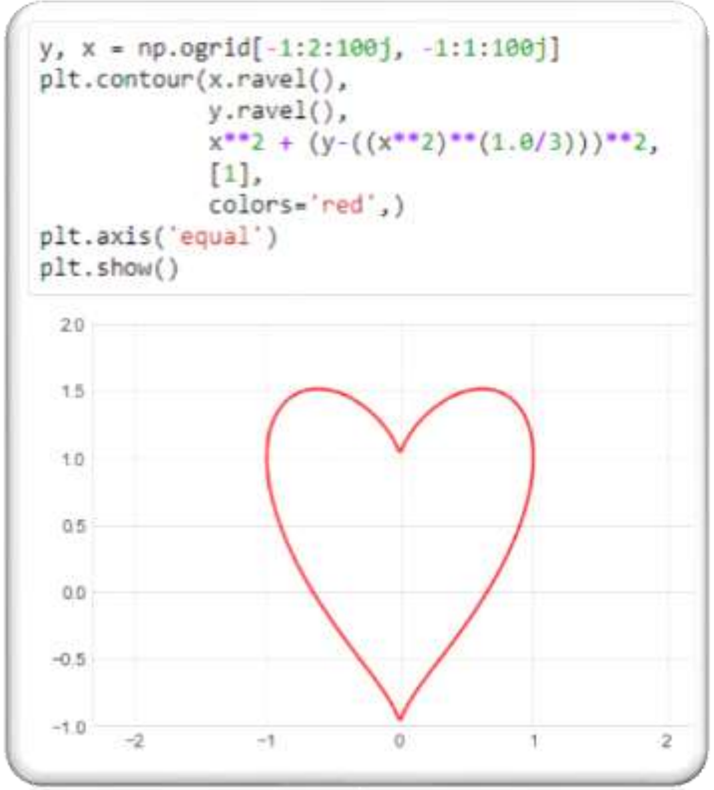


For more information

https://matplotlib.org/gallery/index.html

Introduction

- Matplotlib is a Python 2D plotting library which produces publication quality figures
- Matplotlib is a multiplatform data visualization library built on NumPy arrays, and designed to work with the broader SciPy stack.

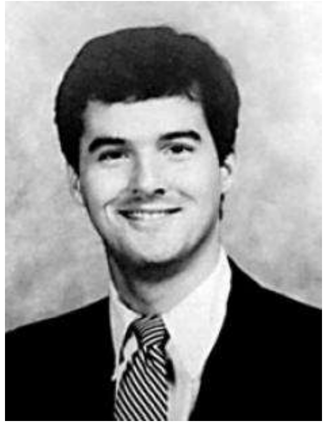


A Fiscally Sponsored Project of

NUMFOCUS

OPEN CODE = BETTER SCIENCE

History



- Matplotlib is the brainchild of John Hunter (1968-2012), who, along with its many contributors, have put an immeasurable amount of time and effort into producing a piece of software utilized by thousands of scientists worldwide.
- Almuns of Princeton University
- American neurobiologist
- one of the founding directors of NumFOCUS Foundation
- Matplotlib was originally conceived to visualize Electrocorticography (ECoG) data of epilepsy patients during post-doctoral research in Neurobiology

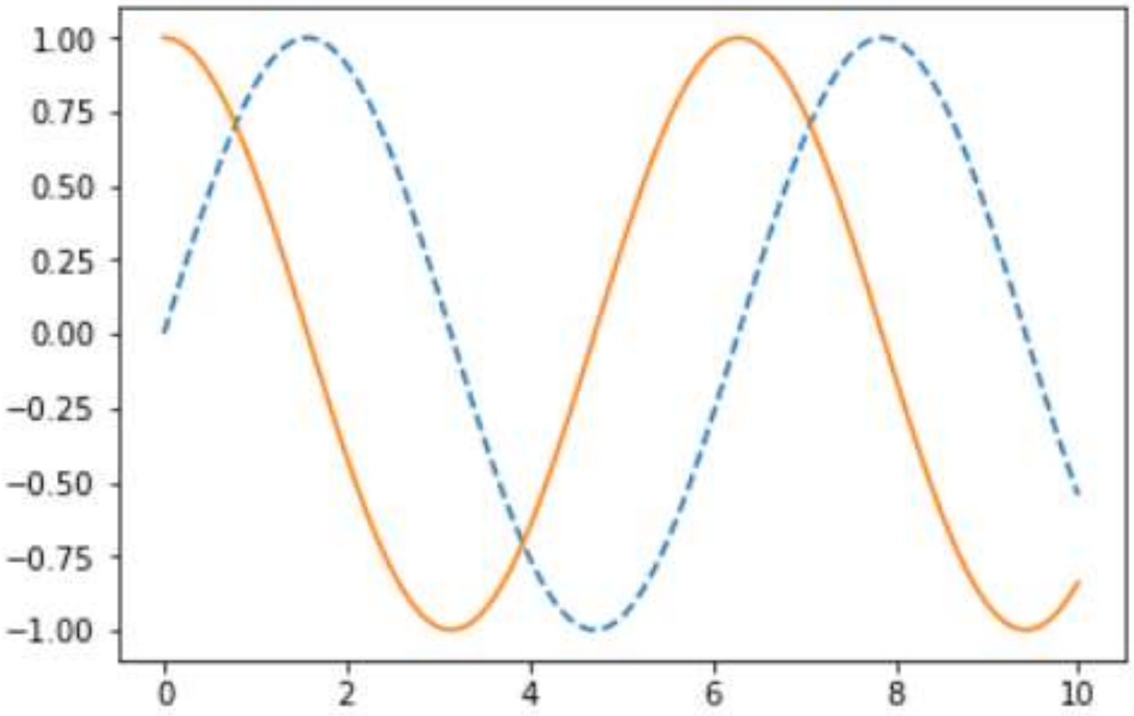
A Fiscally Sponsored Project of NUMFOCUS

OPEN CODE = BETTER SCIENCE

History

- John Hunter in 2002, originally conceived it as a patch to IPython for enabling interactive MATLAB-style plotting via gnuplot from the IPython command line
- IPython’s creator, Fernando Perez, was at the time scrambling to finish his PhD, and let John know he wouldn’t have time to review the patch for several months. John took this as a cue to set out on his own, and the Matplotlib package was born, with version 0.1 released in 2003.
- It received an early boost when it was adopted as the plotting package of choice of the Space Tele- scope Science Institute (the folks behind the Hubble Telescope), which financially supported Matplotlib’s development and greatly expanded its capabilities

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x),'--')
plt.plot(x, np.cos(x))
plt.show()
```



Two Interfaces for the Price of One

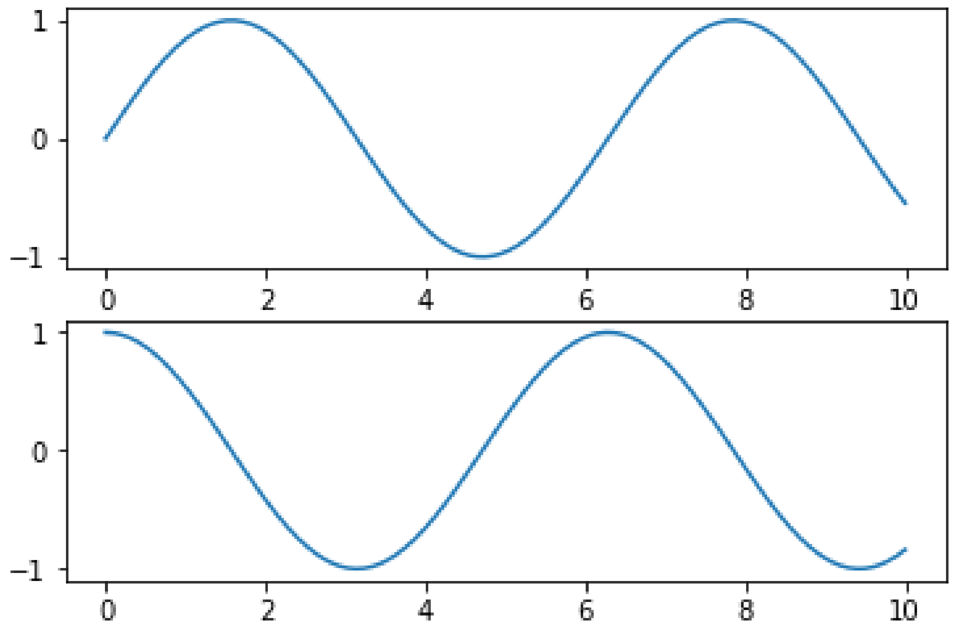
- A potentially confusing feature of Matplotlib is its dual interfaces: *a convenient MATLAB-style state-based interface, and a more powerful object-oriented interface.*

1. MATLAB-style interface : Matplotlib was originally written as a Python alternative for MATLAB users, and much of its syntax reflects that fact. The MATLAB-style tools are contained in the pyplot (plt) interface.
2. Object-oriented interface : The object-oriented interface is available for these more complicated situations, and with more control over figure.

MATLAB-style interface

- It’s important to note that this interface is stateful: it keeps track of the “current” figure and axes.

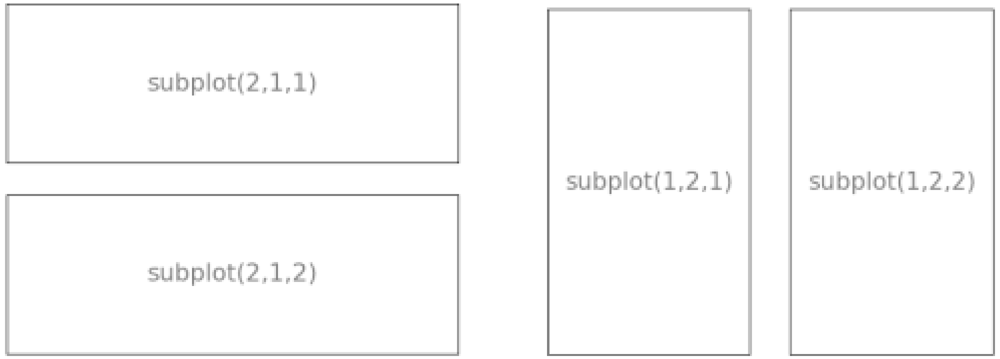
In [12]:
```
plt.figure() # create a plot figure
# create the first of two panels and set current axis
plt.subplot(2, 1, 1) # (rows, columns, panel number)
plt.plot(x, np.sin(x))
# create the second panel and set current axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
plt.show();
```

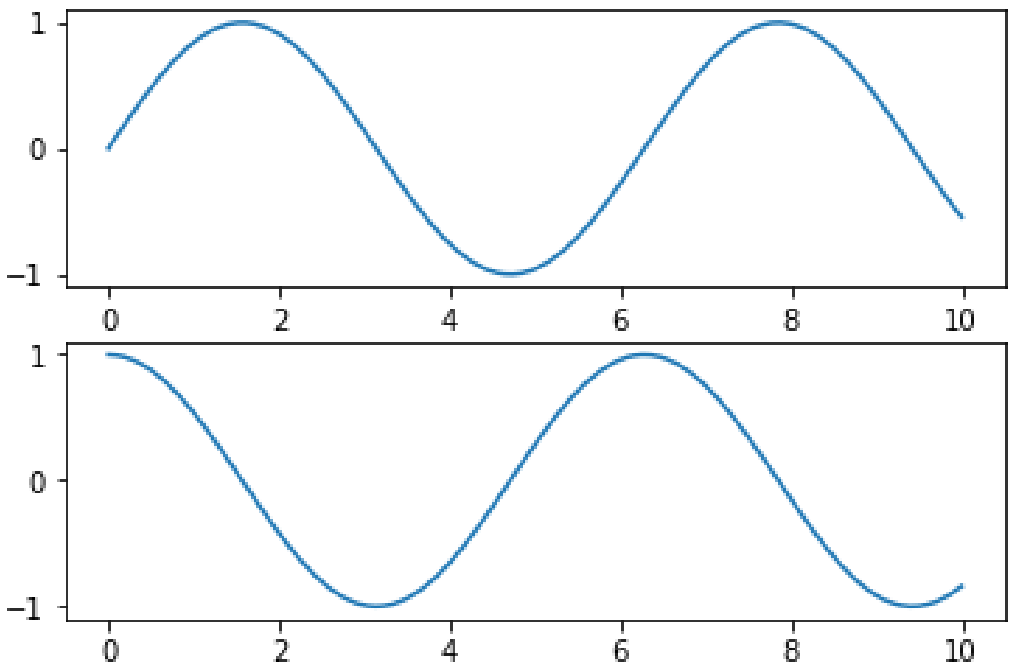


Object-oriented interface

- In the object-oriented interface the plotting functions are methods of explicit Figure and Axes objects.
- With subplot you can arrange plots in a regular grid.

```
# First create a grid of plots
# ax will be an array of two Axes objects
fig, ax = plt.subplots(2)
# Call plot() method on the appropriate object
ax[0].plot(x, np.sin(x))
ax[1].plot(x, np.cos(x));
plt.show();
```

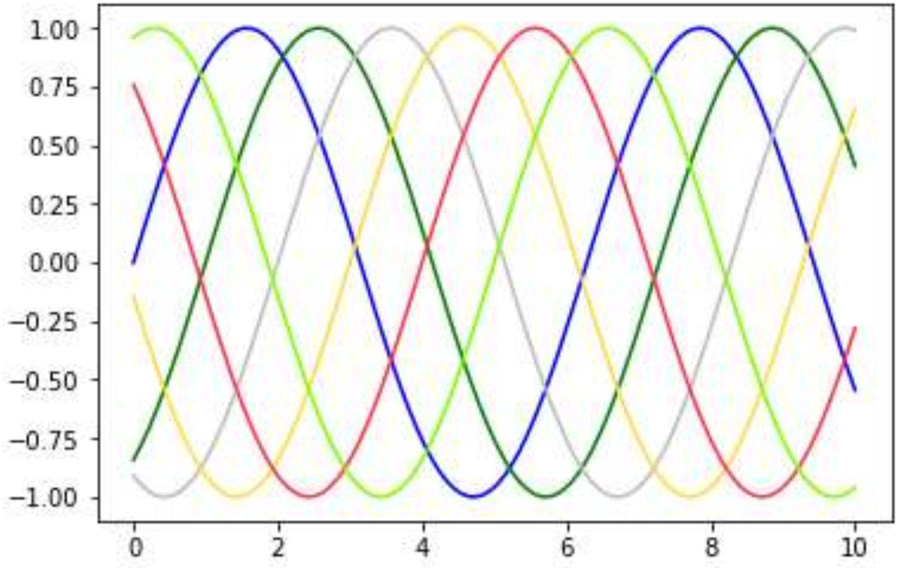




Adjusting the Plot: Line Colors

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

```
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 and 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
plt.show();
```

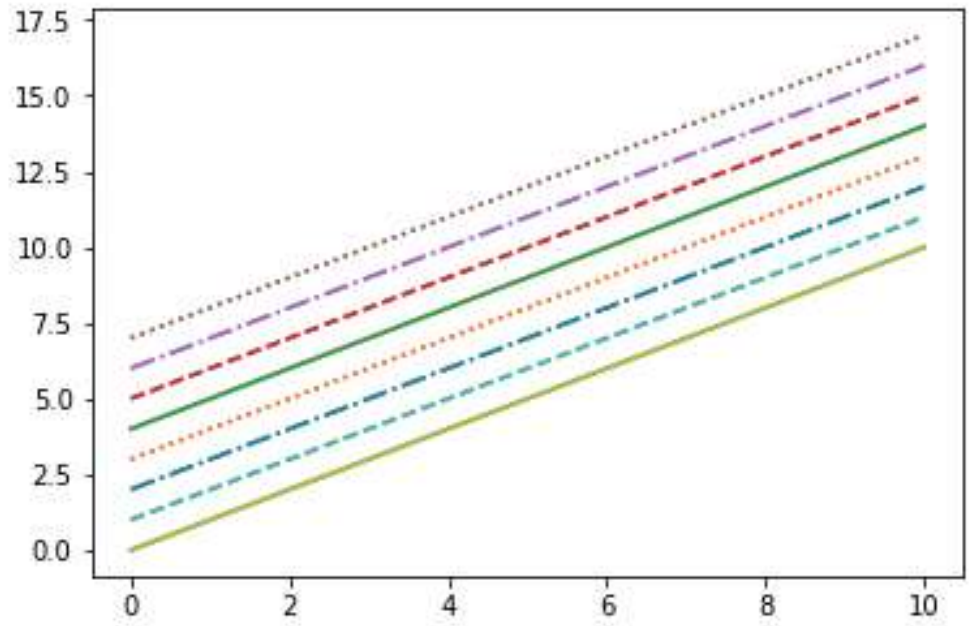


Adjusting the Plot: Styles

- The linestyle and color codes can be combined into a single non-keyword argument

```
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
```

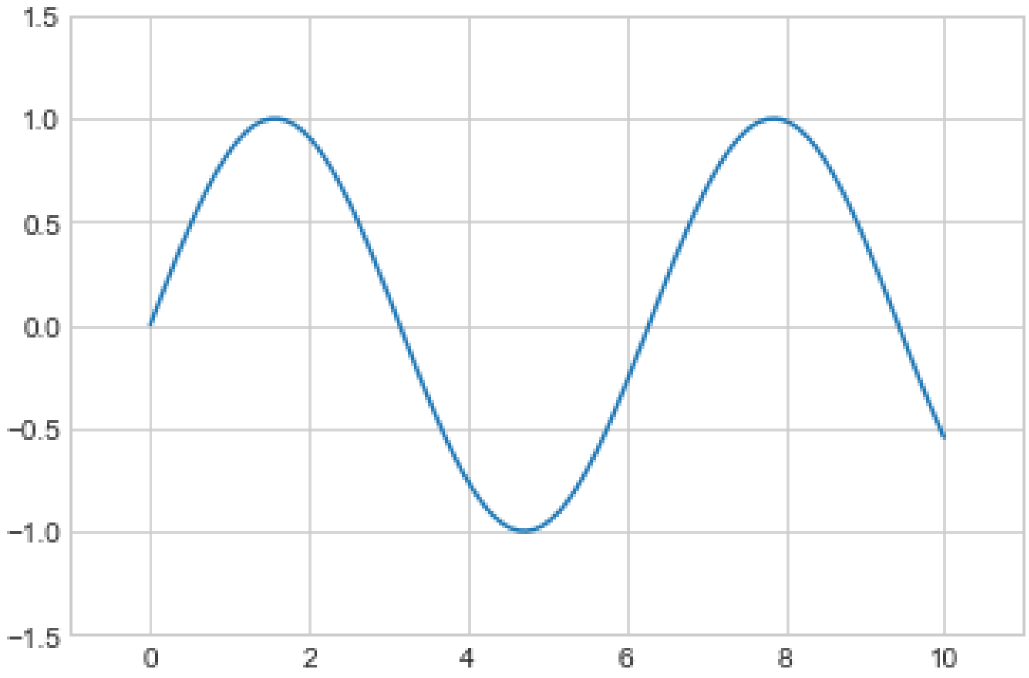
```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');
# For short, you can use the following codes:
plt.plot(x, x + 4, linestyle='-') # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':'); # dotted
plt.show();
```



Adjusting the Plot: Axes Limits

- basic way to adjust axis limits is to use the plt.xlim() and plt.ylim() methods
- The plt.axis() method allows you to set the x and y limits with a single call, by passing a list that specifies [xmin, xmax, ymin, ymax]
- plt.axis('tight'); #automatically tighten the bounds around the current plot
- plt.axis('equal'); #equal aspect ratio

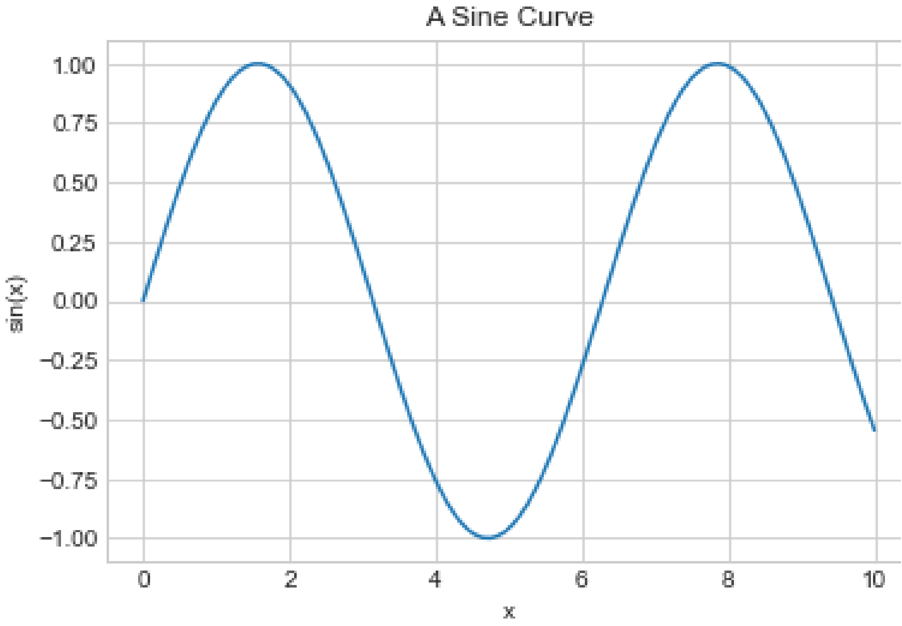
```
plt.style.use('seaborn-whitegrid')
plt.plot(x, np.sin(x))
plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5)
plt.show();
```



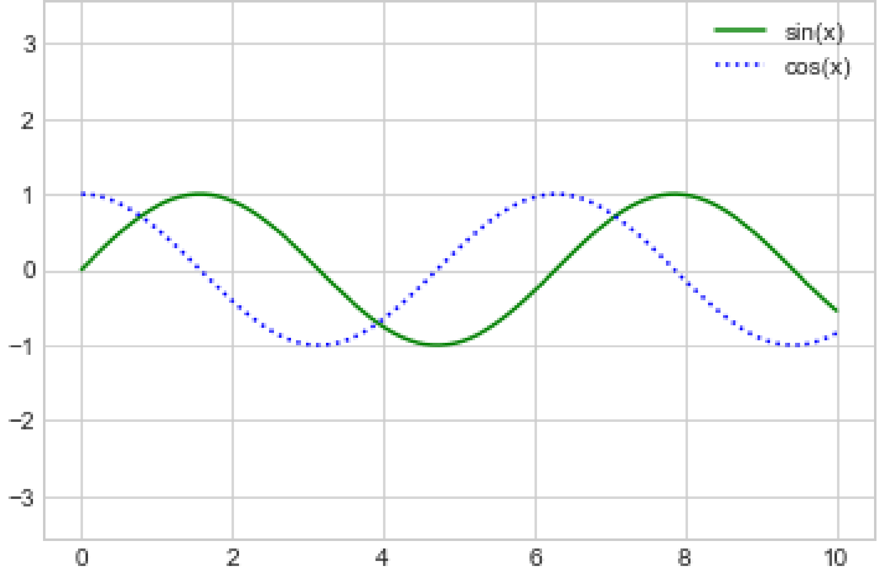
axes is different from axis

Labeling Plots

```
19]: plt.style.use('seaborn-whitegrid')
     plt.plot(x, np.sin(x))
     plt.title("A Sine Curve")
     plt.xlabel("x")
     plt.ylabel("sin(x)");
     plt.show();
```




```
[20]: plt.plot(x, np.sin(x), '-g', label='sin(x)')
      plt.plot(x, np.cos(x), ':b', label='cos(x)')
      plt.axis('equal')
      plt.legend(); plt.show();
```

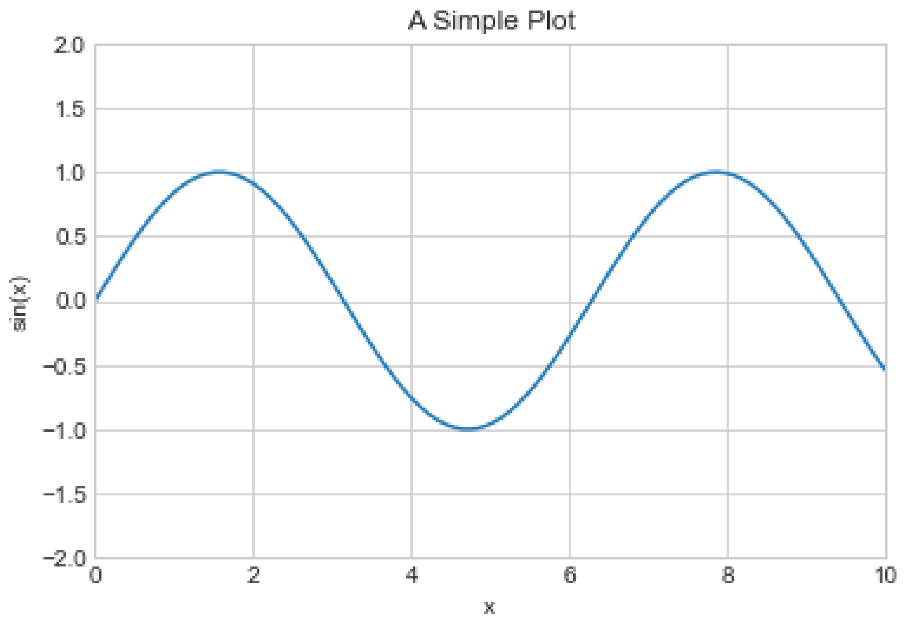


Matplotlib Gotchas

For transitioning between MATLAB-style functions and object-oriented methods, make the following changes:

- plt.xlabel() → ax.set_xlabel()
- plt.ylabel() → ax.set_ylabel()
- plt.xlim() → ax.set_xlim()
- plt.ylim() → ax.set_ylim()
- plt.title() → ax.set_title()

In [26]:
```
ax = plt.axes()
ax.plot(x, np.sin(x))
ax.set(xlim=(0, 10), ylim=(-2, 2), xlabel='x',
       ylabel='sin(x)', title='A Simple Plot');
plt.show();
```

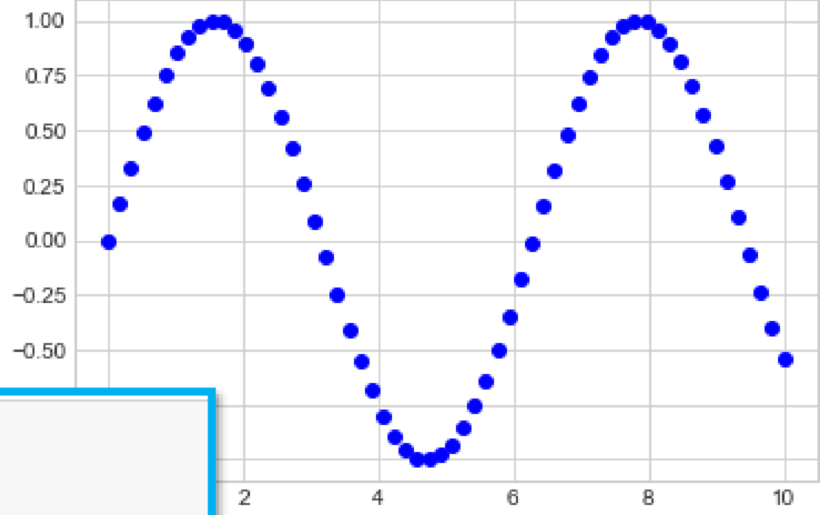


Scatter Plots

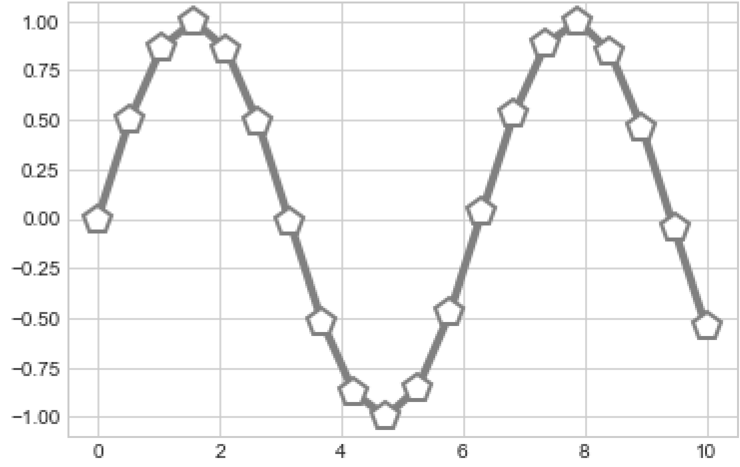
- scatter plot, a close cousin of the line plot
- plt.plot(x, y, '-ok');

  # line (-), circle marker (o), black (k)

41]:
```
x = np.linspace(0, 10, 60)
y = np.sin(x)
plt.plot(x, y, 'o',color='blue');
plt.show();
```



43]:
```
x = np.linspace(0, 10, 20)
y = np.sin(x)
plt.plot(x, y, '-p', color='gray',
 markersize=15, linewidth=4,
 markerfacecolor='white',
 markeredgecolor='gray',
 markeredgewidth=2)
plt.show();
```

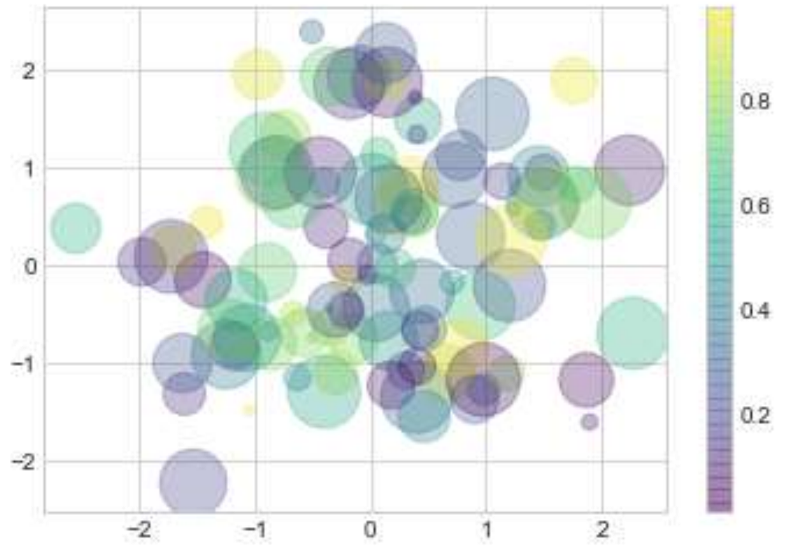


Scatter Plots-efficiency

- as datasets get larger than a few thousand points, plt.plot is noticeably more efficient than plt.scatter.
- plt.scatter has the capability to render a different size and/or color for each point
- plt.plot, the points are always essentially clones of each other
- plt.plot should be preferred over plt.scatter for large datasets.

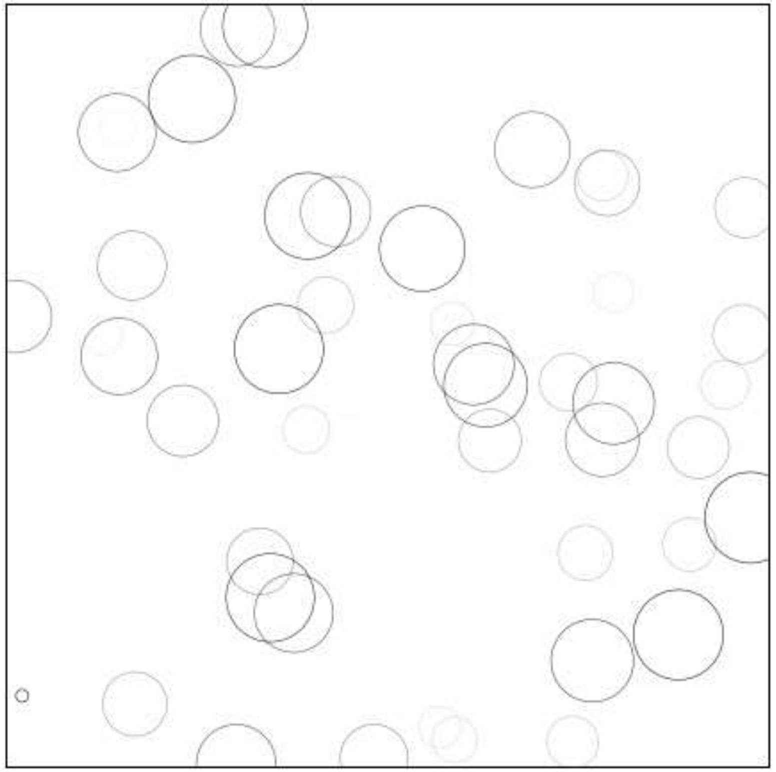
n [46]:

```
rng = np.random.RandomState(0)
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
sizes = 1000 * rng.rand(100)
plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,cmap='viridis')
plt.colorbar(); # show color scale
plt.show();
```



Drip Drop

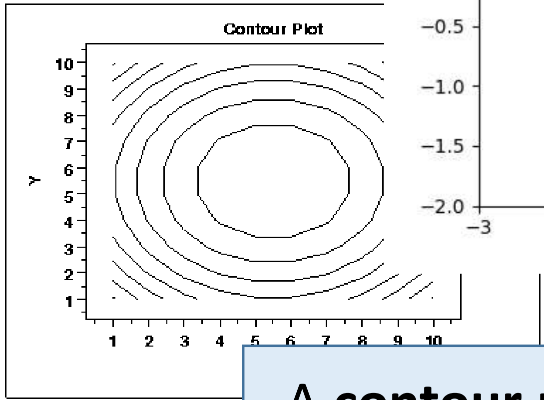
- A very simple rain effect can be obtained by having small growing rings randomly positioned over a figure. Of course, they won't grow forever since the wave is supposed to damp with time.
- To simulate that, we can use a more and more transparent color as the ring is growing, up to the point where it is no more visible. At this point, we remove the ring and create a new one.
- Don’t remove the largest ring but re-use it to set a new ring at a new random position, with nominal size and color.
- Keeping count of rings constant

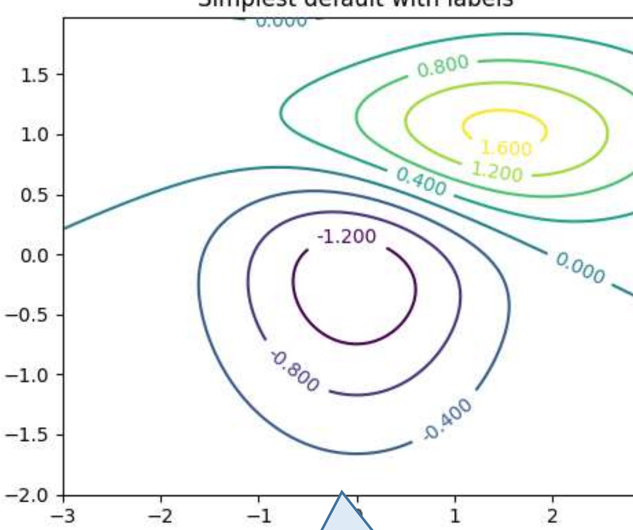


Source:labri.fr

Density and Contour Plots

- It is useful to display three-dimensional data in two dimensions using contours or color-coded regions.
- There are three Matplotlib functions that can be helpful for this task:
  - plt.contour for contour plots,
  - plt.contourf for filled contour plots,
  - plt.imshow for showing images.





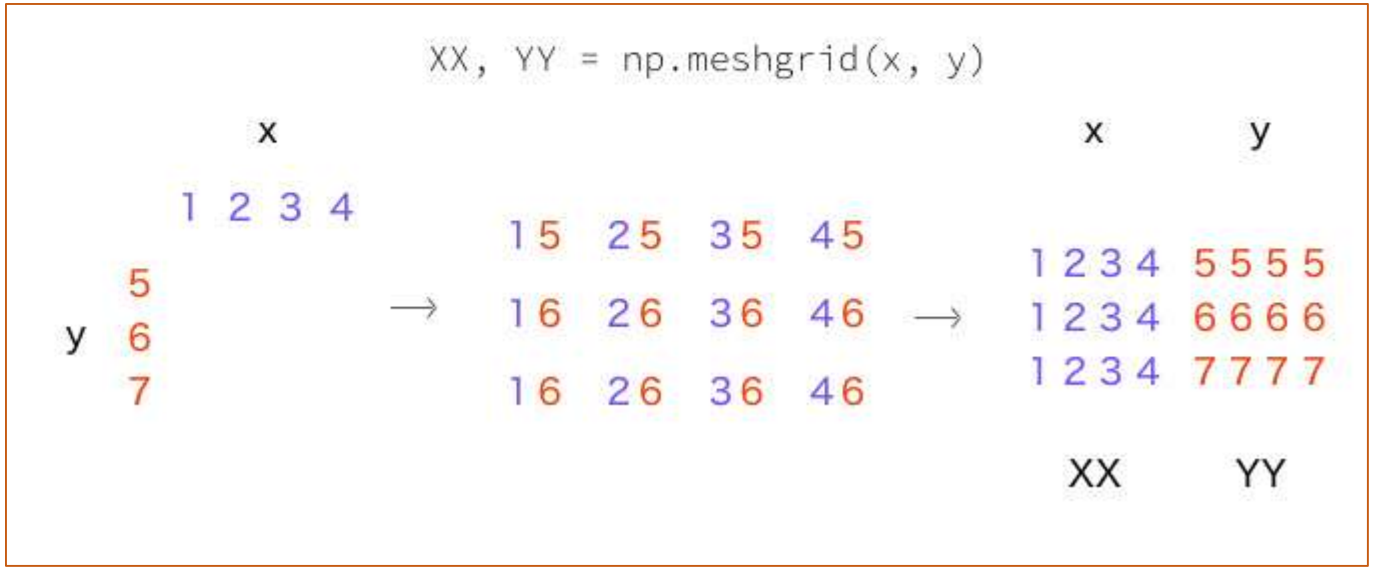
A **contour plot** is a graphical technique for representing a 3-dimensional surface by plotting constant z slices, called **contours**, on a 2-dimensional format. That is, given a value for z, lines are drawn for connecting the (x,y) coordinates where that z value occurs.

Density and Contour Plots

- A contour plot can be created with the plt.contour function.
- It takes three arguments:
  - a grid of x values, a grid of y values, and a grid of z values.
  - The x and y values represent positions on the plot, and the z values will be represented by the contour levels.
- np.meshgrid function is used, which builds two-dimensional grids from one-dimensional arrays

meshgrid working

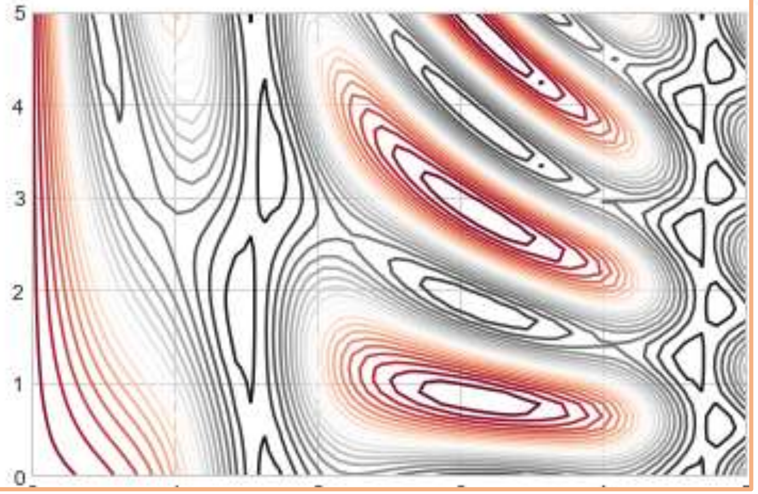
```python
xlist = np.linspace(-3.0, 3.0, 3)
ylist = np.linspace(-3.0, 3.0, 4)
X, Y = np.meshgrid(xlist, ylist)
Z = np.sqrt(X**2 + Y**2)
print("\nxlist value");print(xlist);
print("\nylist value");print(ylist);
print("\nX value");print(X);
print("\nY value");print(Y);
print("\nZ value");print(Z);
```



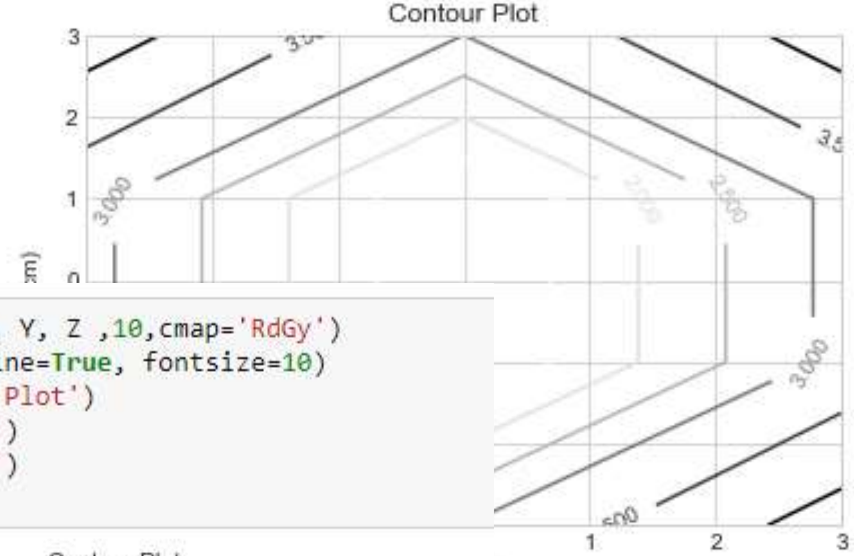

```
xlist value
[-3.  0.  3.]

ylist value
[-3. -1.  1.  3.]

X value
[[-3.  0.  3.]
 [-3.  0.  3.]
 [-3.  0.  3.]
 [-3.  0.  3.]]

Y value
[[-3. -3. -3.]
 [-1. -1. -1.]
 [ 1.  1.  1.]
 [ 3.  3.  3.]]

Z value
[[ 4.24264069  3.          4.24264069]
 [ 3.16227766  1.          3.16227766]
 [ 3.16227766  1.          3.16227766]
 [ 4.24264069  3.          4.24264069]]
```

3]:
```python
def f(x, y):
    return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
plt.contour(X, Y, Z, colors='black');
plt.show();
```

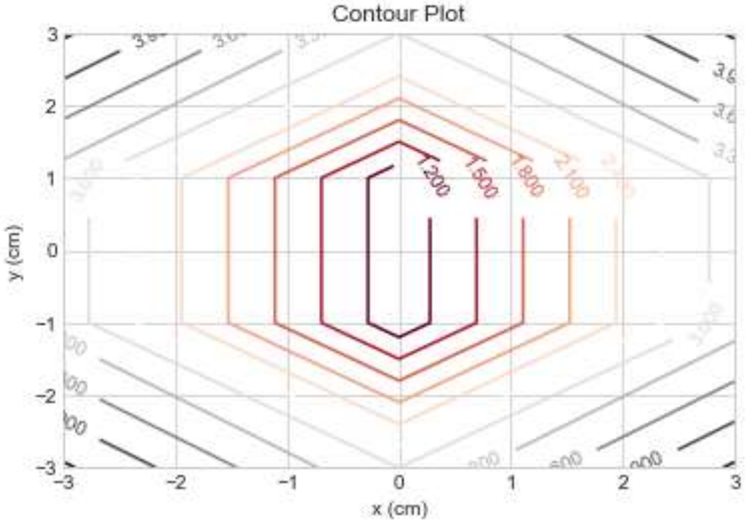
]:
```python
plt.contour(X, Y, Z, 20, cmap='RdGy');
plt.show();
```



[65]:
```python
cp = plt.contour(X, Y, Z )
plt.clabel(cp, inline=True,
          fontsize=10)
plt.title('Contour Plot')
plt.xlabel('x (cm)')
plt.ylabel('y (cm)')
plt.show();
```



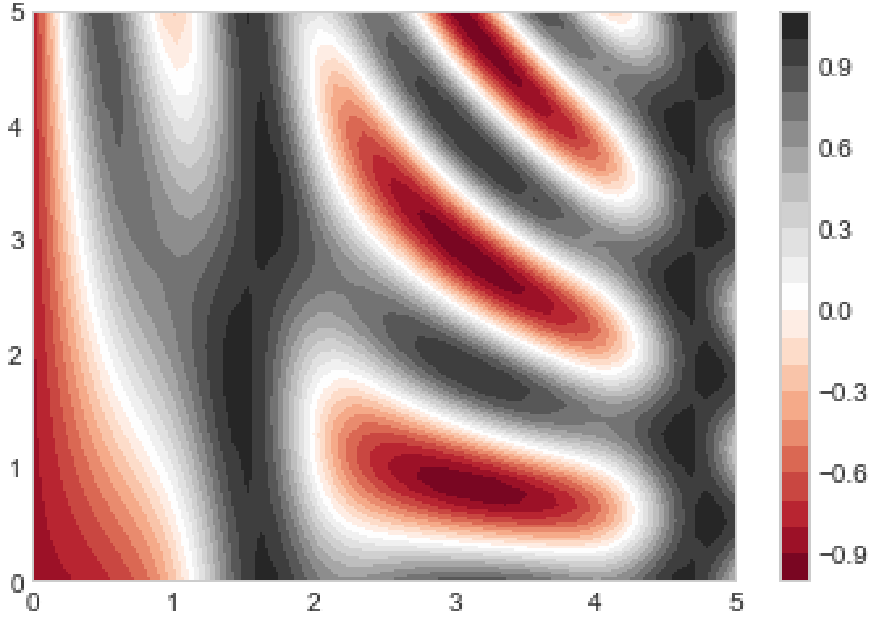
[67]:
```python
cp = plt.contour(X, Y, Z ,10,cmap='RdGy')
plt.clabel(cp, inline=True, fontsize=10)
plt.title('Contour Plot')
plt.xlabel('x (cm)')
plt.ylabel('y (cm)')
plt.show();
```



Filled contours

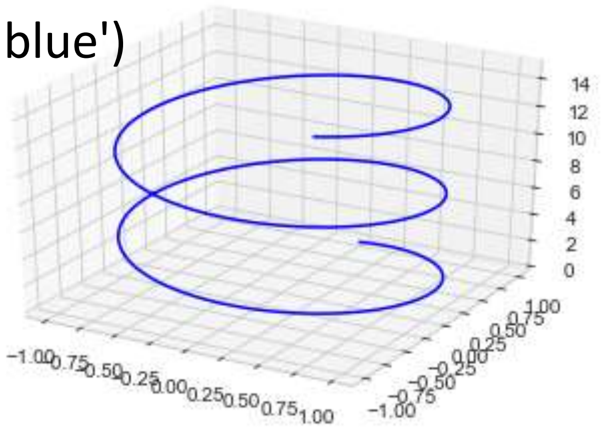
- The colorbar makes it clear that the black regions are “peaks,” while the red regions are “valleys.”

[88]:
```
def f(x, y):
    return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
plt.contourf(X, Y, Z ,20, cmap='RdGy');
plt.colorbar();
plt.show();
```

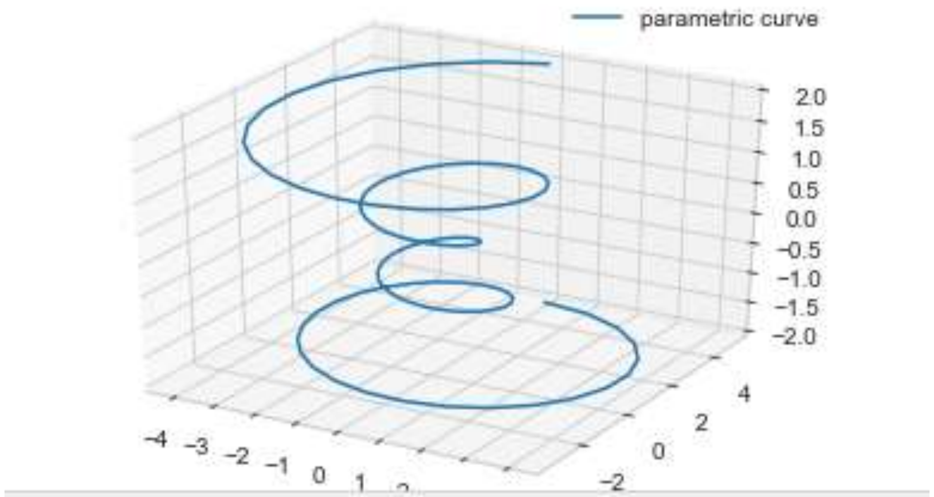


Three-Dimensional Plotting in Matplotlib

```
%matplotlib inline
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection='3d')
# Data for a three-dimensional line
zline = np.linspace(0, 15, 1000)
xline = np.sin(zline)
yline = np.cos(zline)
ax.plot3D(xline, yline, zline, 'blue')
```

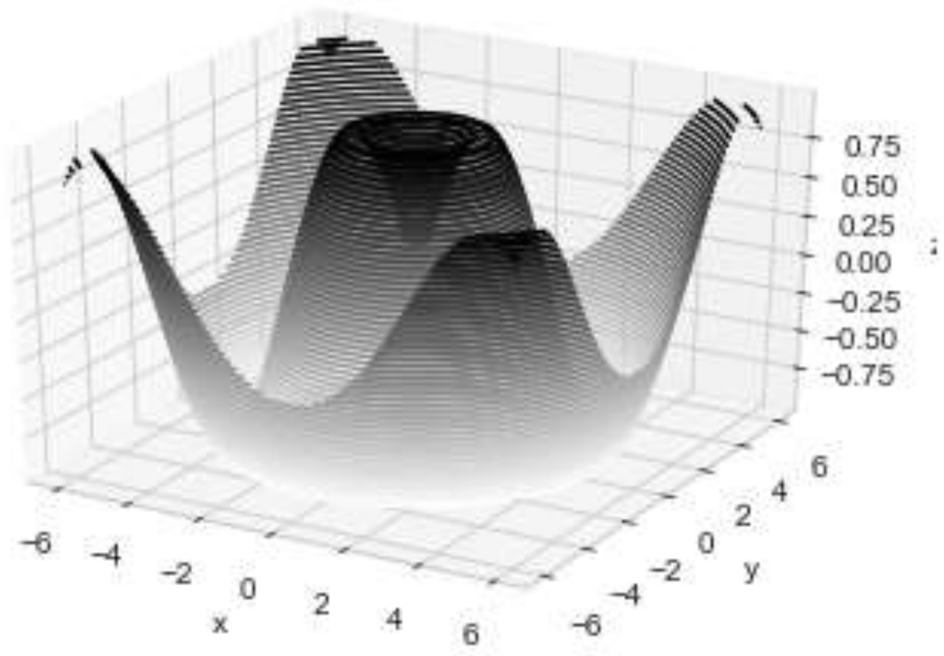



```
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
mpl.rcParams['legend.fontsize'] = 10
fig = plt.figure()
ax = fig.gca(projection='3d')
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-2, 2, 100)
r = z**2 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)
ax.plot(x, y, z, label='parametric curve')
ax.legend()
plt.show()
```



3D contours

```
def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))
x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
plt.show();
```

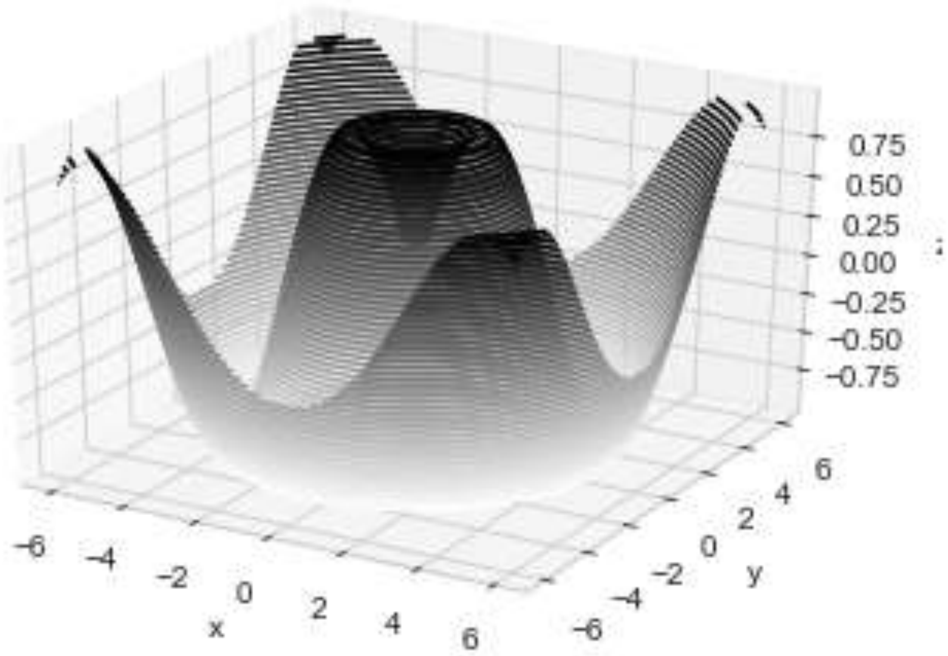


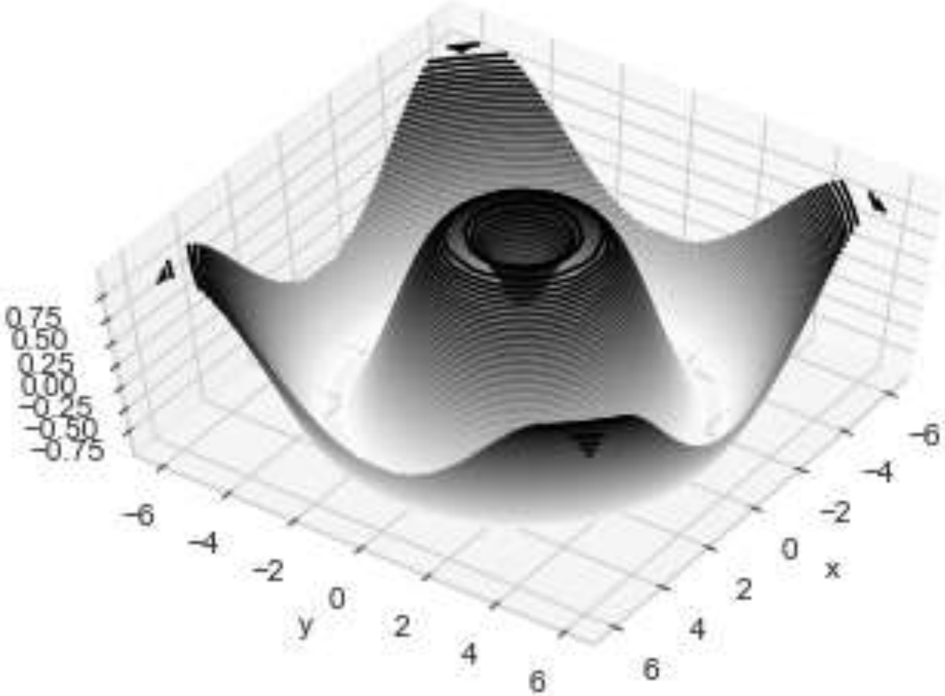
3D contours

view_init method to set the elevation and azimuthal angles

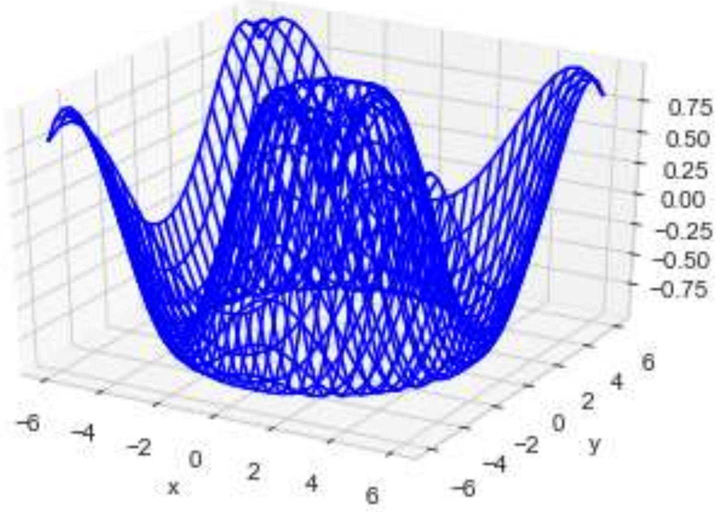
ax.view_init(60, 35)

elevation of 60 degrees (that is, 60 degrees above the x-y plane) and an azimuth of 35 degrees (that is, rotated 35 degrees counter-clockwise about the z-axis)

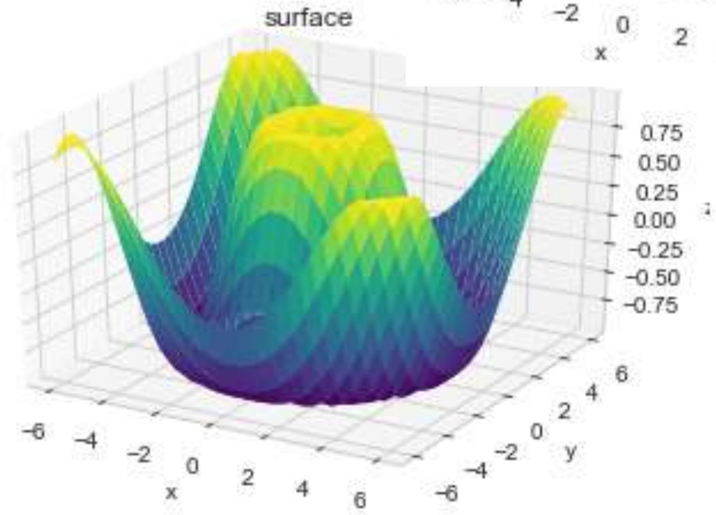




- ax.plot_wireframe(X, Y, Z, color='black')



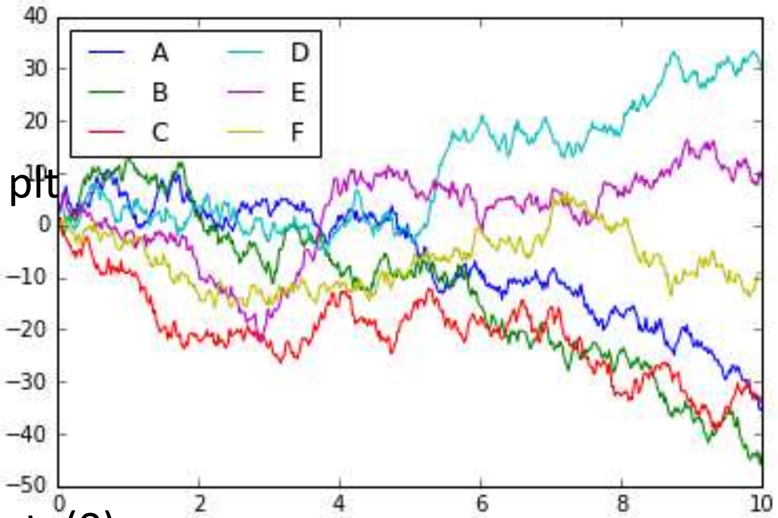
- ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='viridis', edgecolor='none')
- ax.set_title('surface')



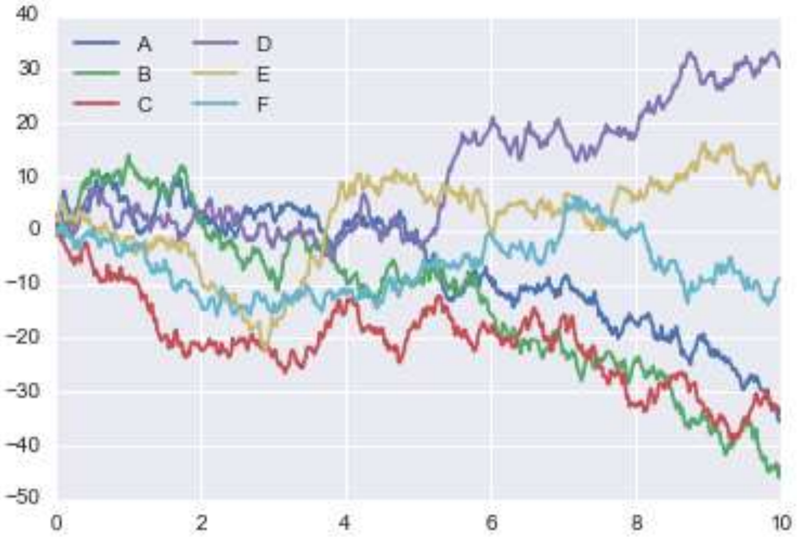
https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html

Visualization with Seaborn

- Seaborn provides an API on top of Matplotlib that offers sane choices for plot style and color defaults




```
import matplotlib.pyplot as plt
plt.style.use('classic')
%matplotlib inline
import numpy as np
import pandas as pd
# Create some random data
rng = np.random.RandomState(0)
x = np.linspace(0, 10, 500)
y = np.cumsum(rng.randn(500, 6), 0)
# Plot the data with Matplotlib defaults
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left');
plt.show();
```




```
import seaborn as sns
sns.set()
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left');
plt.show()
```