Python – Graphical outputs

Python can enable graphical outputs with the modules `numpy` and `matplotlib.pyplot`. For 3D plots, the function `Axes3d` of module `mpl_toolkits.mplot3d` can be used.

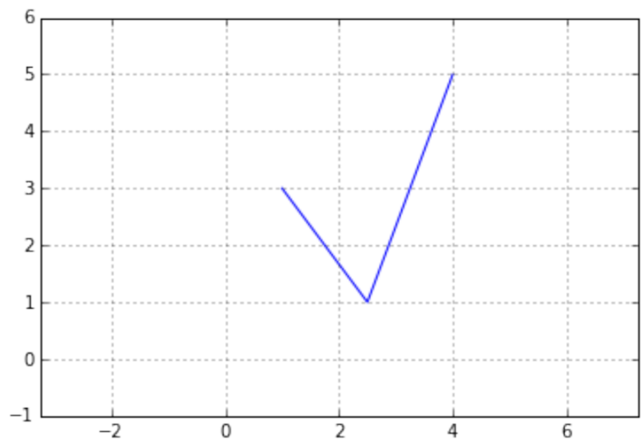
```
import math
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
```

The following examples illustrate the use of these modules.

1. Plot of piecewise linear functions.

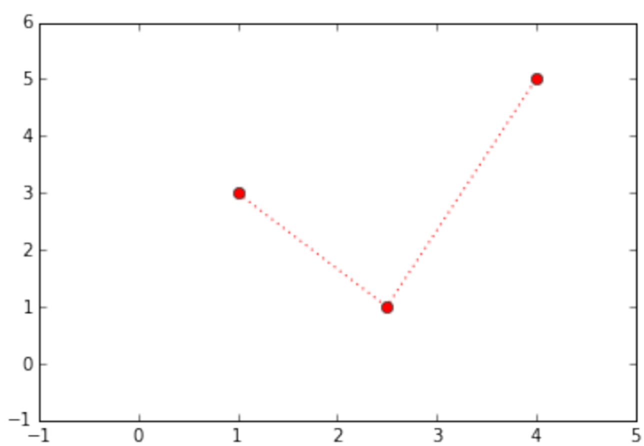
Given two lists of x and y coordinates, the function `axis` creates the window which contains the graphical output. The option `equal` is used to obtain the same scale on both axis. Multiple plots created with the `plot` function are superimposed in the same window. The function `plt.clf()` is used to remove all plots from the graphical window.

```
x = [1., 2.5, 4.]
y = [3., 1., 5.]
plt.axis('equal')
plt.plot(x, y)
plt.axis([-1., 5., -1., 6.])
plt.grid()
plt.show()
```



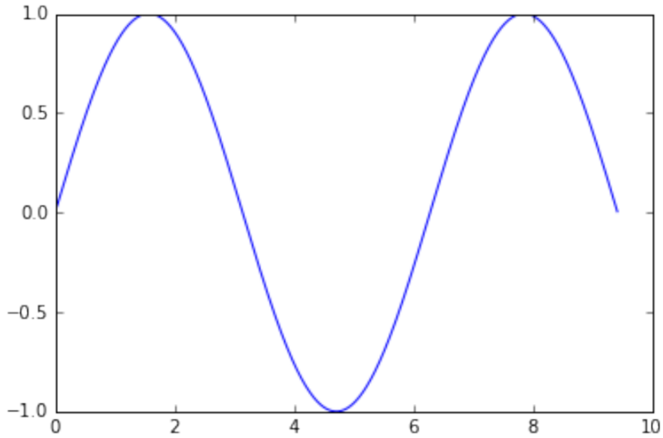
The `plot` function allows for many plotting options. The parameter `color` is used to choose the color of the line plot (`'g'` for green, `'r'` for red, `'b'` for blue). To define a line style, use the `linestyle` parameter (`'-'` for continuous line, `'- -'` for dashed line, `':'` for dotted line). To mark the end points of the line segments with a particular symbol, use the parameter `marker` (`'+'` , `'.'` , `'o'` , `'v'` are a few markers).

```
x = [1., 2.5, 4.]
y = [3., 1., 5.]
plt.axis([-1., 5., -1., 6.])
plt.plot(x, y, color='r', linestyle=':',
         marker='o')
plt.show()
```



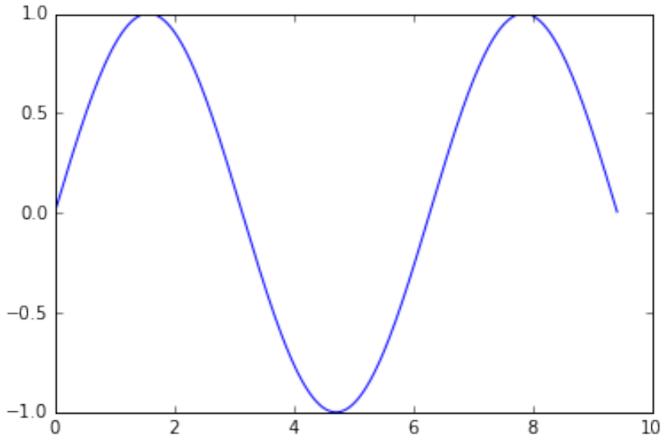
2. Plot of functions. First a list of x values is defined. The list of the $y = f(x)$ values is then created. The following example shows the function $f(x) = \sin(x)$ in the interval $[0, 3\pi]$.

```
def f(x):
    return math.sin(x)

X = np.arange(0, 3*np.pi, 0.01)
Y = [ f(x) for x in X ]
plt.plot(X, Y)
plt.show()
```



It is more interesting to use functions of the `numpy` module, rather than those of the `math` module. It is then possible to work equally well with scalars and arrays. The latter are referred as "vectorized functions" or "universal functions (`ufunc` in the language of Python).

```
def f(x):
    return np.sin(x)

X = np.arange(0, 3*np.pi, 0.01)
Y = f(X)
plt.plot(X, Y)
plt.show()
```



Another solution consists of using the `vectorize` function of the `numpy` module. It allows for a transformation of a scalar function into a function operating on arrays. However, it is a lot more efficient to use vectorized functions of the `numpy` module.

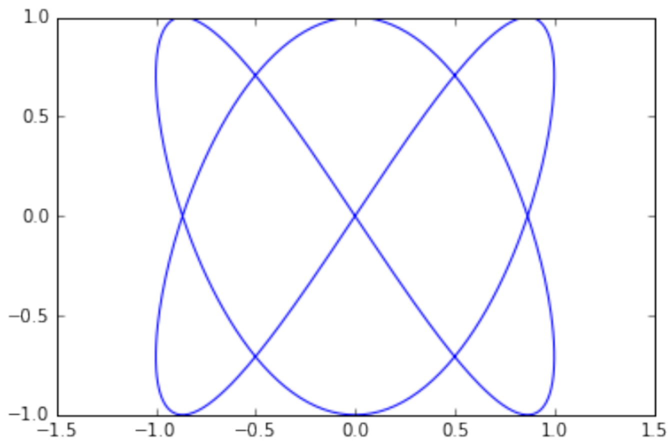
```
def f(x):
    return math.sin(x)

f = np.vectorize(f)
```

Note that Python operators (+,-,*, etc) can act on arrays, term by term. Hence, a function $f(x, y)$ can act on two scalars or on two arrays, or even on a scalar and an array.

```
def f(x, y):
    return np.sqrt(x**2 + y**2)

>>> f(3, 4)
5.0
>>> f(np.array([1, 2, 3]), np.array([4, 5, 6]))
array([ 4.12310563,  5.38516481,  6.70820393])
>>> f(np.array([1, 2, 3]), 4)
array([ 4.12310563,  4.47213595,  5.        ])
```

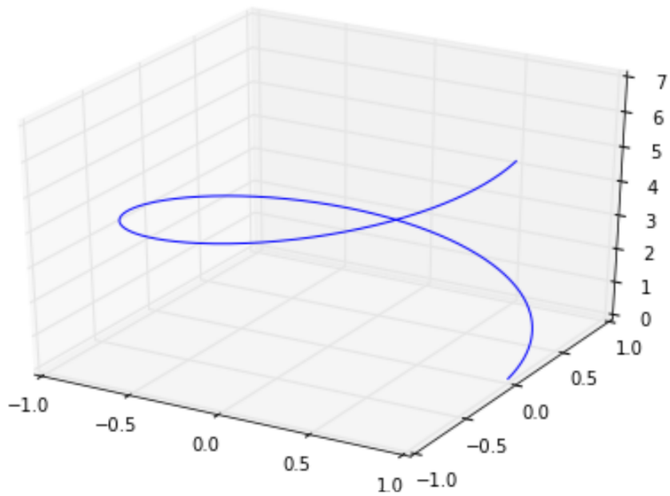
3. Plot of parametrized curves. First, a list of values of the parameter t is defined. Then a list of the $(x(t), y(t))$ coordinates is created. The plot is then created.

```
def x(t):
    return np.sin(2*t)

def y(t):
    return np.sin(3*t)

T = np.arange(0, 2*np.pi, 0.01)
X = x(T)
Y = y(T)
plt.axis('equal')
plt.plot(X, Y)
plt.show()
```



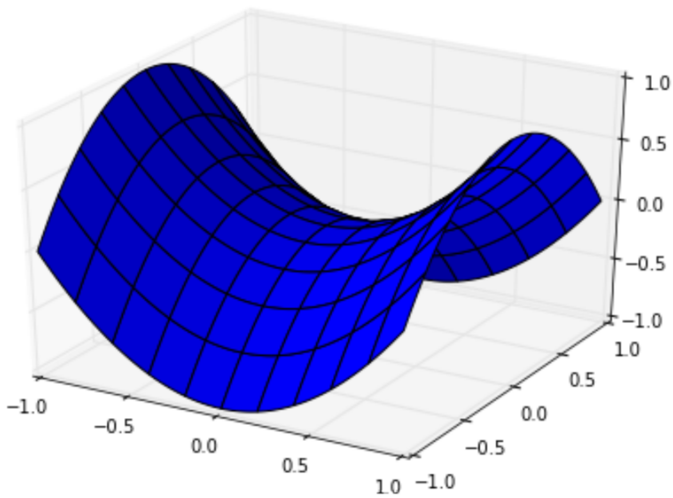
3D parametrized curves can also be plotted.

```
ax = Axes3D(plt.figure())
T = np.arange(0, 2*np.pi, 0.01)
X = np.cos(T)
Y = np.sin(T)
Z = T
ax.plot(X, Y, T)
plt.show()
```



4. Plots of surfaces. To plot a surface $z = f(x, y)$, we must first build an array of (x, y) values. Then values z are calculated for each entry of the array. The surface can then be plotted by using the function `plot_surface`.

```
ax = Axes3D(plt.figure())

def f(x,y):
    return x**2 - y**2

f=np.vectorize(f)
X = np.arange(-1, 1, 0.02)
Y = np.arange(-1, 1, 0.02)
X, Y = np.meshgrid(X, Y)
Z = f(X, Y)
ax.plot_surface(X, Y, Z)
plt.show()
```



5. Plots of contour lines. To plot the contour lines of functions $f(x, y)$, we must first build an array of (x, y) values. Then values f are calculated for each entry of the array. The function `contour` is then used by indicating a list of contour values $k = f(x, y)$ to be used. The contour lines $f(x, y) = k$ can then be plotted.

```
def f(x,y):
    return x**2 + y**2 + x*y

f=np.vectorize(f)
X = np.arange(-1, 1, 0.01)
Y = np.arange(-1, 1, 0.01)
X, Y = np.meshgrid(X, Y)
Z = f(X, Y)
plt.axis('equal')
plt.contour(X, Y, Z, [0.1,0.4,0.5])
plt.show()
```