Visualization in Python

Numerical Computing with Python

Alexey Fedoseev

November 14, 2019

Getting started

Visualization is the product of almost all computations.

There are a number of high-quality visualization packages available in Python

- `matplotlib` focuses on generating publication-quality plots
- `seaborn` targets statistical data analysis
- `ggplot` is based on the famous `R` package
- `Plotly` and `Bokeh` focus on interactivity
- and others

Installing Matplotlib

To install `matplotlib` package run the following command in your terminal

```
$ pip install matplotlib
```

Anaconda base environment comes with pre-installed `matplotlib` package. If you need to install it in a new environment run

```
$ conda install matplotlib
```

`matplotlib` is imported using the following command

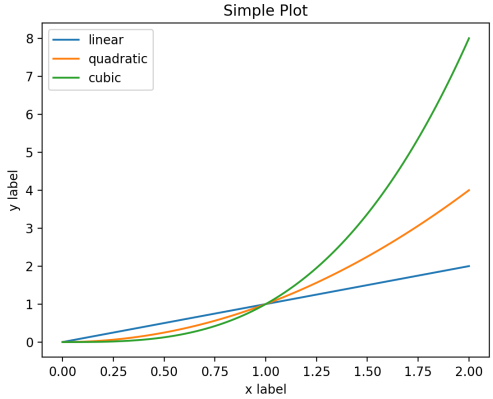
```
>>> import matplotlib.pyplot as plt
```

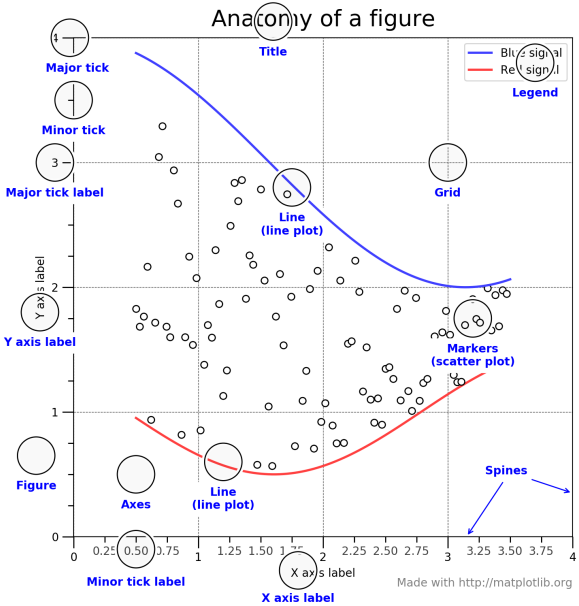
Also import `numpy` as it is frequently used together with `matplotlib`

```
>>> import numpy as np
```

Simple plot in matplotlib

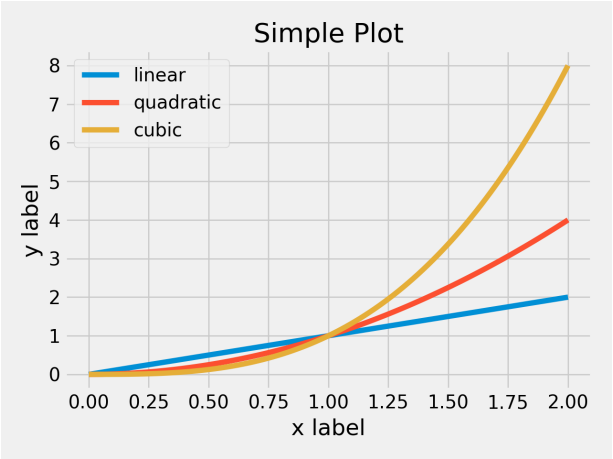
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)

plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()

plt.show()
```




Anatomy of a figure
Major tick
Minor tick
Major tick label
Title
Legend
Grid
Line
(line plot)
Y axis label
Markers
(scatter plot)
Figure
Axes
Line
(line plot)
Spines
Minor tick label
X axis label
Made with http://matplotlib.org

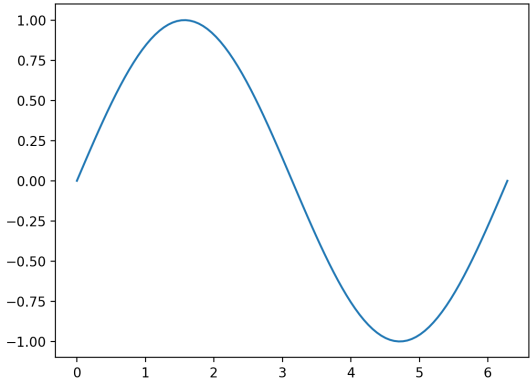
Styling

```
>>> plt.style.available
['seaborn-dark',
'seaborn-darkgrid',
'seaborn-ticks',
'fivethirtyeight',
...
'seaborn-poster',
'seaborn-deep']

>>> plt.style.use('fivethirtyeight')
>>> plt.style.use('default')
```



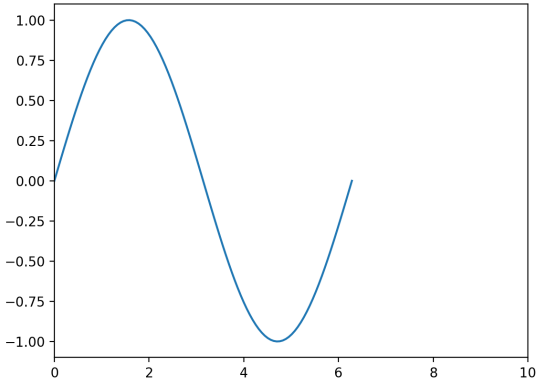
Simple start

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2*np.pi, 100)

plt.plot(x, np.sin(x))

plt.show()
```



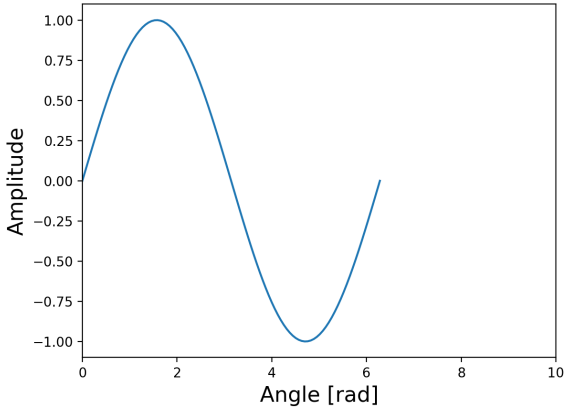
Adjust Limits

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2*np.pi, 100)

plt.plot(x, np.sin(x))
plt.xlim(0, 10)

plt.show()
```



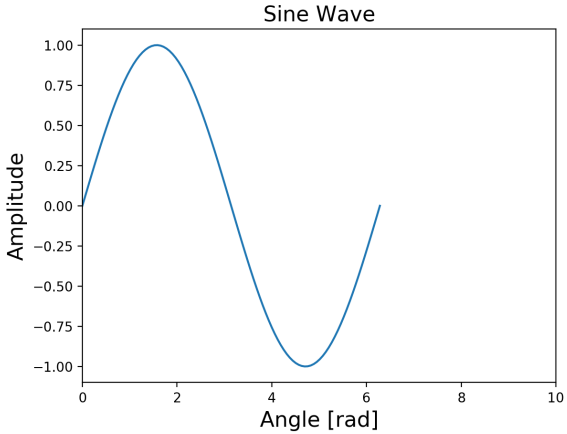
Add Labels

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2*np.pi, 100)

plt.plot(x, np.sin(x))
plt.xlim(0, 10)
plt.xlabel('Angle [rad]',
    fontsize = 16)
plt.ylabel('Amplitude',
    fontsize = 16)

plt.show()
```



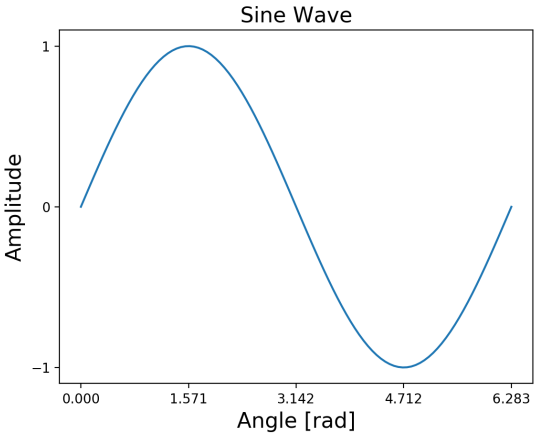
Add Title

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2*np.pi, 100)

plt.plot(x, np.sin(x))
plt.xlim(0, 10)
plt.xlabel('Angle [rad]',
    fontsize = 16)
plt.ylabel('Amplitude',
    fontsize = 16)
plt.title('Sine Wave',
    fontsize = 16)

plt.show()
```



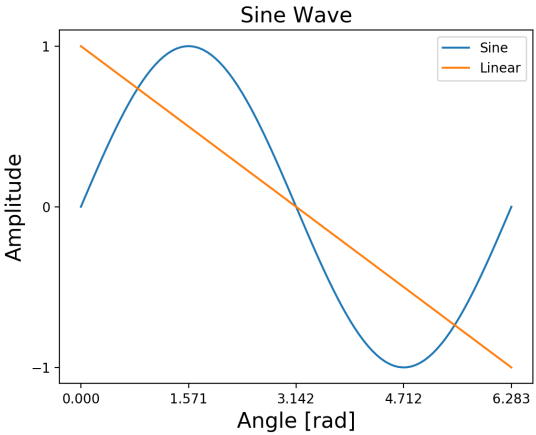
Adjust Ticks

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2*np.pi, 100)

plt.plot(x, np.sin(x))
plt.xlabel('Angle [rad]',
    fontsize = 16)
plt.ylabel('Amplitude',
    fontsize = 16)
plt.title('Sine Wave',
    fontsize = 16)
plt.xticks(np.linspace(0,
    2*np.pi, num=5))
plt.yticks(np.linspace(-1, 1, num=3))
plt.show()
```



Add Legend

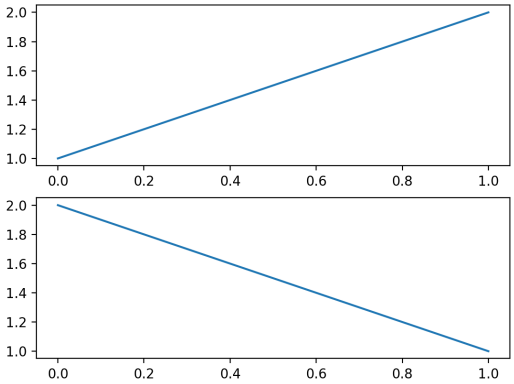
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2*np.pi, 100)
plt.plot(x, np.sin(x), label='Sine')
plt.plot(x, (-x+np.pi)/np.pi,
    label='Linear')
plt.xlabel('Angle [rad]',
    fontsize = 16)
plt.ylabel('Amplitude',fontsize=16)
plt.title('Sine Wave',fontsize=16)
plt.xticks(np.linspace(0,
    2*np.pi, num=5))
plt.yticks(np.linspace(-1, 1, num=3))
plt.legend()
plt.show()
```



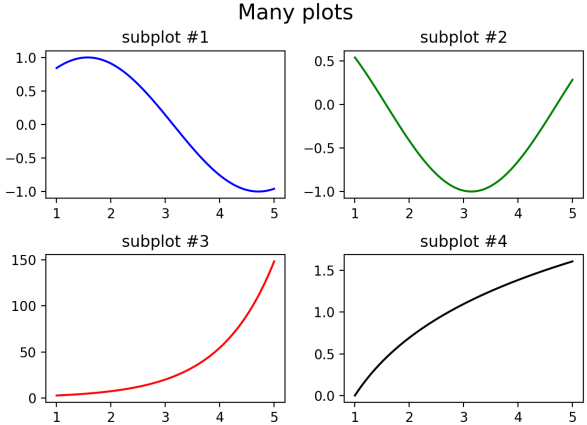
Subplots

Sometimes it is helpful to compare different views of data side by side. Matplotlib has the concept of subplots: groups of smaller axes that can exist together within a single figure.

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

plt.subplot(2,1,1)
plt.plot([1,2])

plt.subplot(2,1,2)
plt.plot([2,1])

plt.show()
```



```
x = np.linspace(1, 5, 100)
plt.suptitle('Many plots',
    fontsize=16)
plt.subplot(2,2,1)
plt.plot(x, np.sin(x), color="blue")
plt.title('subplot #1')
plt.subplot(2,2,2)
plt.plot(x, np.cos(x), color="green")
plt.subplot(2,2,3)
plt.plot(x, np.exp(x), color="red")
plt.subplot(2,2,4)
plt.plot(x, np.log(x), color="black")
# adjust spacing between subplots;
# rect parameter specifies the bounding box
# that the subplots will be fit inside.
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

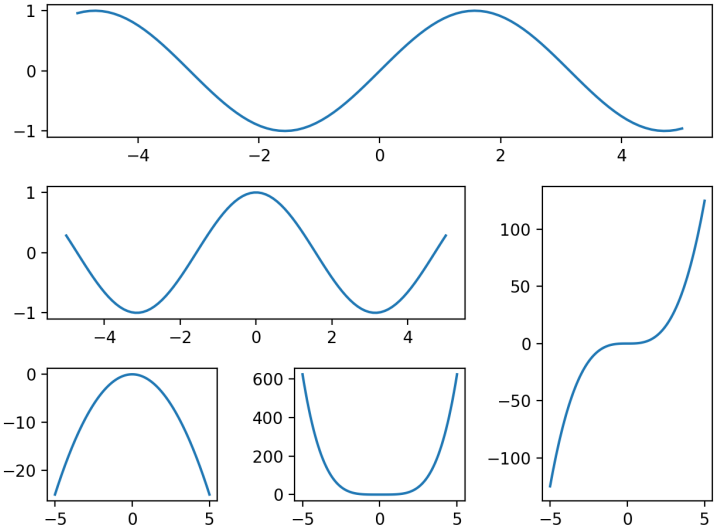


Grid of Subplots

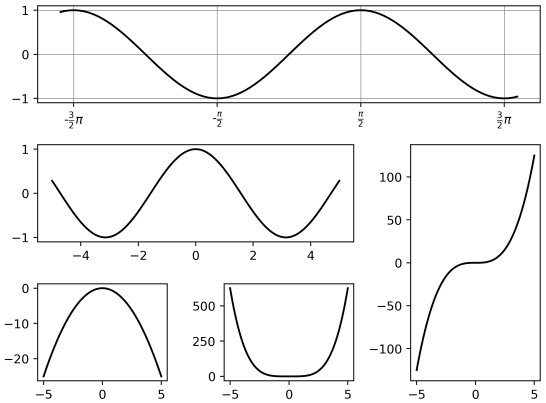
If you require a complex grid of subplots use `subplot2grid`

```
fig = plt.figure()
# index starts with 0
ax1 = plt.subplot2grid((3, 3), (0, 0), colspan=3)
ax2 = plt.subplot2grid((3, 3), (1, 0), colspan=2)
ax3 = plt.subplot2grid((3, 3), (1, 2), rowspan=2)
ax4 = plt.subplot2grid((3, 3), (2, 0))
ax5 = plt.subplot2grid((3, 3), (2, 1))
```

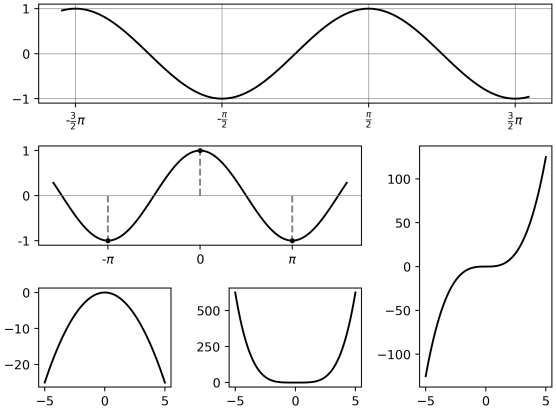
```
ax1.plot(x, np.sin(x))
ax2.plot(x, np.cos(x))
ax3.plot(x, x**3)
ax4.plot(x, -x**2)
ax5.plot(x, x**4)

plt.tight_layout()
plt.show()
```



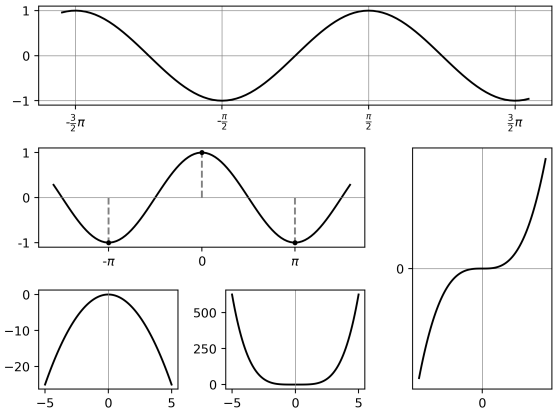
```
ax1.plot(x, np.sin(x), c='black')
ax1.grid(color='gray',linewidth=0.5)
ax1.set_xticks(
    np.linspace(-1.5*np.pi,1.5*np.pi,4))
ax1.set_xticklabels(
    [r"-$\frac{3}{2}\pi$",
     r"-$\frac{\pi}{2}$",
     r"$\frac{\pi}{2}$",
     r"$\frac{3}{2}\pi$"])

ax2.plot(x, np.cos(x), c='black')
ax3.plot(x, x**3, c='black')
ax4.plot(x, -x**2, c='black')
ax5.plot(x, x**4, c='black')
# Save plot instead of showing it
plt.savefig('gridplot_black.png', dpi=fig.dpi)
```



```
ax2.plot(x, np.cos(x), c='black')
ax2.set_yticks([-1,0,1])
ax2.set_yticklabels([-1,0,1])
ax2.set_xticks([-np.pi, 0, np.pi])
ax2.set_xticklabels([r"-$\pi$",
    r"$0$", r"$\pi$"])
ax2.set_ylim(-1.1, 1.1)
ax2.axhline(y=0, c='gray', lw=0.5)

ax2.plot([-np.pi,-np.pi], [0,-1],
    '--', c='gray')
ax2.plot([0,0], [0,1], '--', c='gray')
ax2.plot([np.pi, np.pi], [0, -1], '--',
    c='gray')
ax2.plot(-np.pi, -1, 'ko', ms=3)
ax2.plot(0, 1, 'ko', markersize=3) # same as color='black', marker='o'
ax2.plot(np.pi, -1, 'ko', ms=3)
```



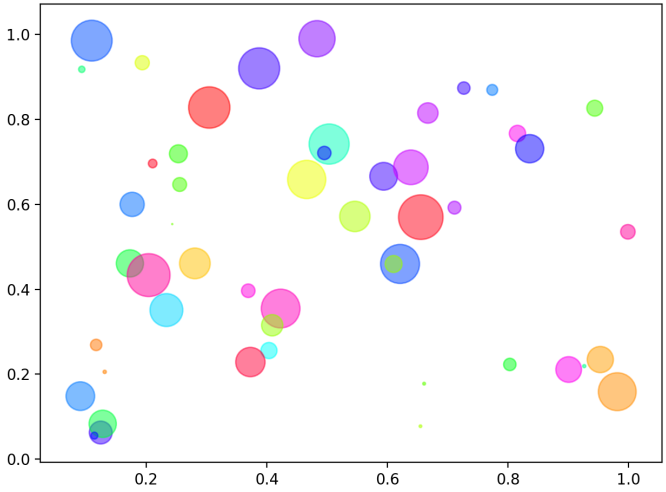
```
ax3.plot(x, x**3, c='black')
ax3.set_xticks([0])
ax3.set_yticks([0])
ax3.grid(color='gray', lw=0.5)

ax4.plot(x, -x**2, c='black')
ax4.axvline(x=0, c='gray', lw=0.5)
ax5.plot(x, x**4, c='black')
ax5.axvline(x=0, c='gray', lw=0.5)
```



Scatter plot

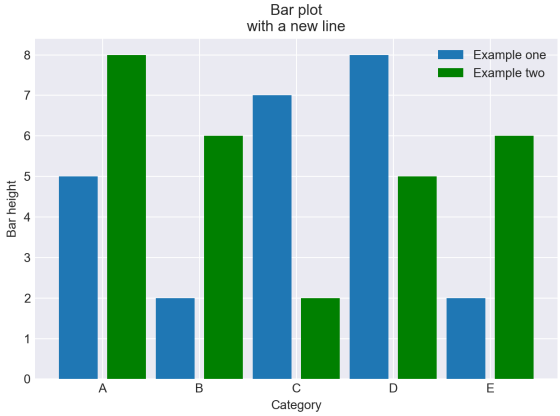
A scatter plot of y vs x with varying marker size and/or color.

```
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area=(30*np.random.rand(N))**2

plt.scatter(x, y, s=area,
    cmap='hsv', c=colors,
    alpha=0.5)
plt.show()
```



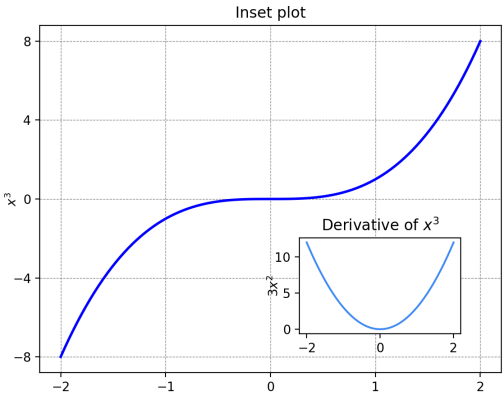
Bar plot

```
plt.style.use('seaborn-darkgrid')

plt.bar([1,3,5,7,9],[5,2,7,8,2],
    label="Example one")
plt.bar([2,4,6,8,10],[8,6,2,5,6],
    label="Example two", color='g')

plt.legend()
plt.xlabel('Category')
plt.ylabel('Bar height')
plt.xticks(np.linspace(1.5,9.5,num=5),
    ['A','B','C','D','E'])
plt.title('Bar plot\nwith a new line')

plt.show()
```



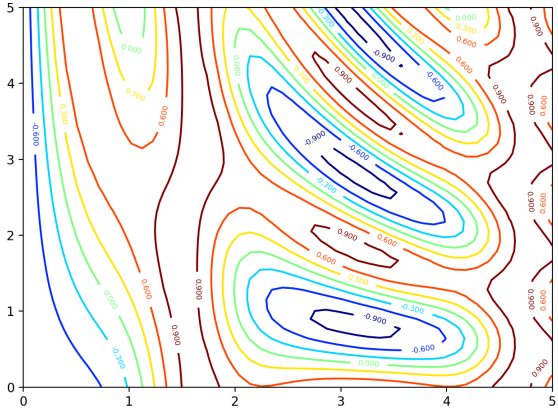
Inset plot

```
fig = plt.figure()
x = np.linspace(-2,2,100)
y, dy = x**3, 3*x**2
# axes' [left, bottom, width, height]
fig.add_axes([0.1, 0.1, 0.8, 0.8])
plt.plot(x, y, 'b-', linewidth=2)
plt.ylabel("$x^3$", labelpad=-5)
plt.title("Inset plots")
plt.xticks(np.linspace(-2,2,5))
plt.yticks(np.linspace(-8,8,5))
plt.grid(color='gray',
    linestyle='--', linewidth=0.5)
fig.add_axes([0.55, 0.19, 0.28, 0.22])
plt.plot(x, dy,c="#3f8af4")
plt.ylabel("$3x^2$", labelpad=-5)
plt.title("Derivative of $x^3$")
```



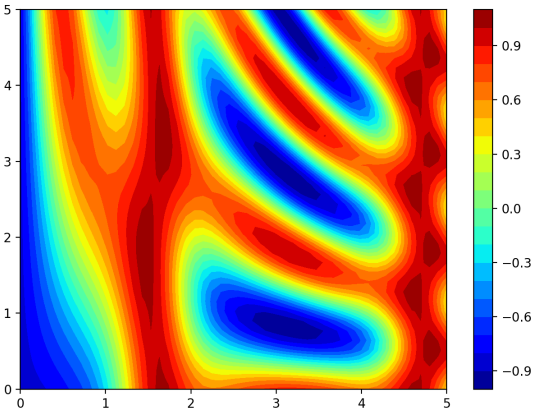
Contour plots

```
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)

# contour line is a curve along which
# the function has a constant value
contours = plt.contour(X, Y, Z, 6,
    cmap='jet')
plt.clabel(contours, inline=True, fontsize=6)
plt.show()
```



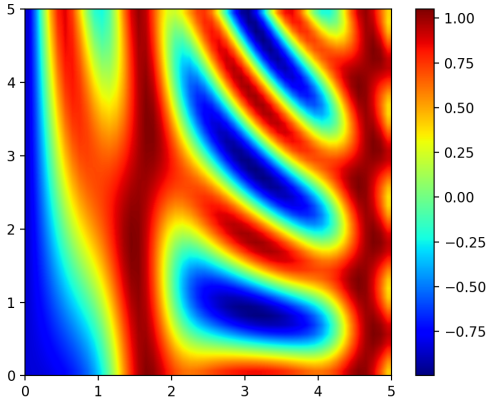
Filled contour plots

```
plt.contourf(X, Y, Z, 20, cmap='jet')
plt.colorbar()
```



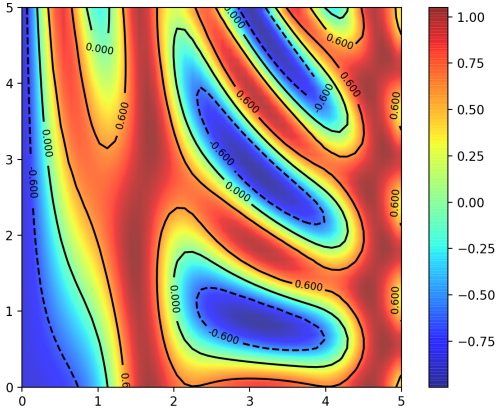
Plot data as an image

```
im = plt.imshow(Z,
     # specify the limits
     extent=[0, 5, 0, 5],
     # by default image origin
     # is in the upper left;
     # change it to the lower left
     origin='lower',
     cmap='jet')

# make it smooth
im.set_interpolation('bilinear')
plt.colorbar()
```



Filled contour plot with labels

```
contours = plt.contour(X, Y, Z, 3,
    colors='black')
plt.clabel(contours,
    inline=True, fontsize=8)
im = plt.imshow(Z, extent=[0, 5, 0, 5],
    origin='lower', cmap='jet',
    # set transparency
    alpha=0.75)
im.set_interpolation('bilinear')
plt.colorbar()
```



3d plotting

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure() # Open a figure.
ax = fig.add_subplot(111, 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()
```