BCH2203 Python - 6. Visualization

Ramses van Zon

16 February 2022

Review

Python language basics

- **Python command line**
- **scripts**, “shebangs”
- **variables:** numbers, strings
- **indentation** as syntax
- `print` and `input` functions
- `if`/`elif`/`else` statements
- `try`/`except` for errors

Further Python language basics

- `while` and `for` loops
- `range` generator
- **lists**
- **dicts**
- **file I/O:** `open`, `with`
- **slicing, finding, replacing** with lists and strings

Some more Python

- **list comprehension**
- **writing functions**
- **comments**
- **doc strings**
- **packages**: `import`, `import ... as ...`
- **command-line arguments**
- **regular expressions**

useful external Python packages

- `numpy`: arrays and operations
- `pandas`: dataframes i.e. tables with labels

Today:

- `matplotlib`: plotting

Installing Packages

- When a package does not come with Python, it has to be installed first.
- Some python distributions (e.g. Anaconda, the python module on Teach) come with a lot of packages already installed.
- How to install additional packages, depends on a few factors
  1. Using plain python or (ana)conda?
  2. Shared system or your own?
  3. System-wide Python installation or per user?
  4. Part of the PyPI repository or not?

Installing Packages

Using plain python or (Ana)conda?

- Plain python uses `pip install ...` to install packages.
- Anaconda uses `conda install ...` .
- On your own computer, Anaconda is more convenient, but on shared systems (e.g. Teach), plain python is better.

Shared system or your own?

- Cannot add package to system Python on a shared system like Teach.

Installing Packages

System-wide Python installation or per user?

If Python is installed system-wide (includes python modules on Teach), before installing packages, you need to create your own virtual environment base off of that Python:

```
$ ssh -Y USERNAME@teach.scinet.utoronto.ca
$ module load python/3
$ virtualenv --system-site-packages $HOME/myenv
```

and each time you want to use python, you have to say

```
$ source $HOME/myenv/bin/activate
```

You can now `pip install ...` packages, and they will be added to this virtual environment.

(conda environments are similar, with different syntax).

Part of the PyPI repository or not?

If it's not in the Python Package Index (PyPI, https://pypi.org), you'll have to install from source, instead of using `pip`.

Read the packages documentation, the installation mechanisms vary.

Matplotlib

Simple plot in matplotlib

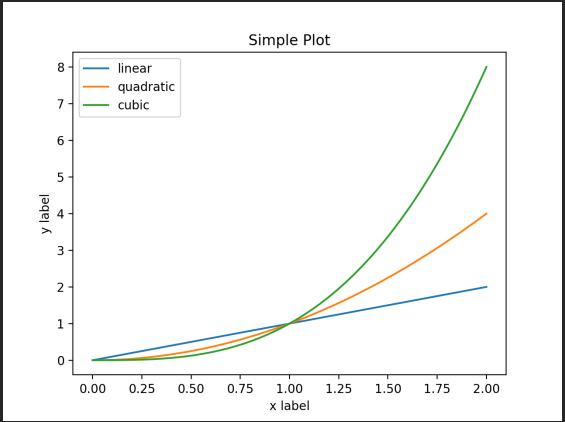
```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>>
>>> def show():
...     plt.pause(.1)
>>>
>>> x = np.linspace(0, 2, 100)
>>>
>>> linear, = plt.plot(x, x)
>>> show()
```

```
>>> quadratic, = plt.plot(x, x**2, label='quadratic')
>>> show()
```

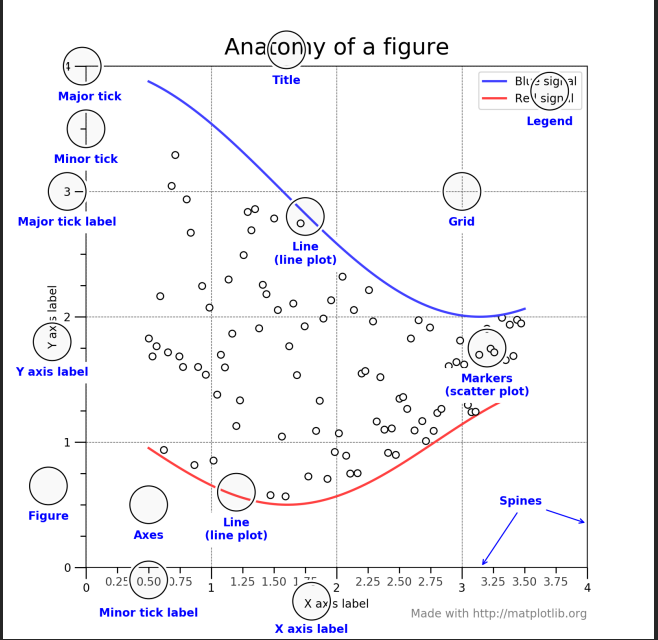
```
>>> mylegend = plt.legend()
>>> show()
```

```
>>> linear.set_label('linear')
>>> show()
```

```
>>> cubic, = plt.plot(x, x**3, label='cubic')
>>> myxlabel = plt.xlabel('x label')
>>> myylabel = plt.ylabel('y label')
>>> mytitle = plt.title("Simple Plot")
>>> show()
```



- matplotlib functions return objects.
- In jupyter, put all plot in the same cell!



d

Source: Matplotlib.org

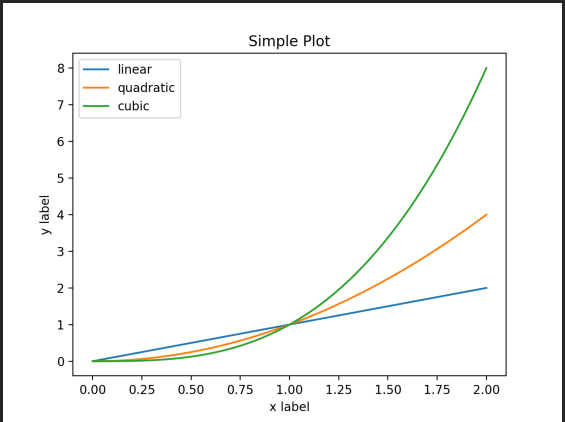
Save a plot in matplotlib

Use the 'savefig' command to save the plot into a file.

Best practice is to put the plot generation in a script.

```
#!/usr/bin/env python3
import matplotlib
matplotlib.use('agg') # no display

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.savefig("simpleplot.png")
```

```
$ python simpleplot.py
$ display simpleplot.png
```



Style

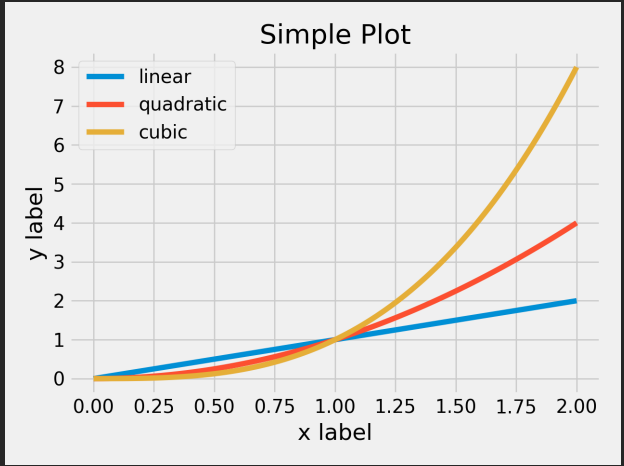
Styling

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

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

Must do this before plotting!

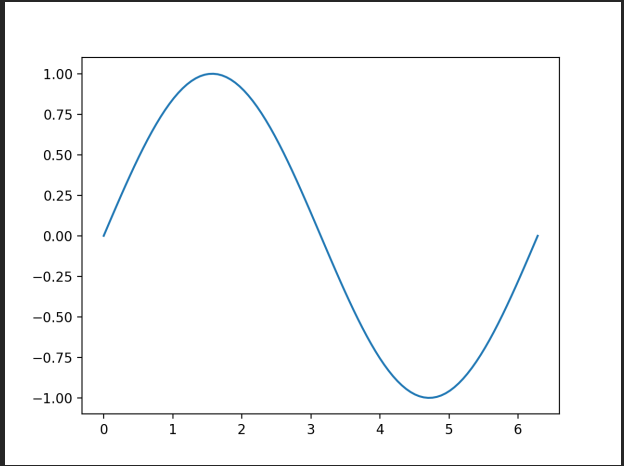
In interactive plotting, clear the current figure with:

```
>>> plt.clf()
```



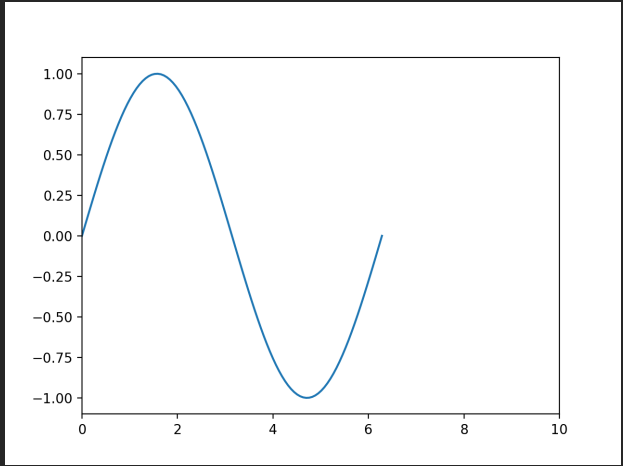
Simple start

```
import matplotlib.pyplot as plt
import numpy as np
def show():
   plt.pause(.1)

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

plt.plot(x, np.sin(x), label='Sine') ; show()
```



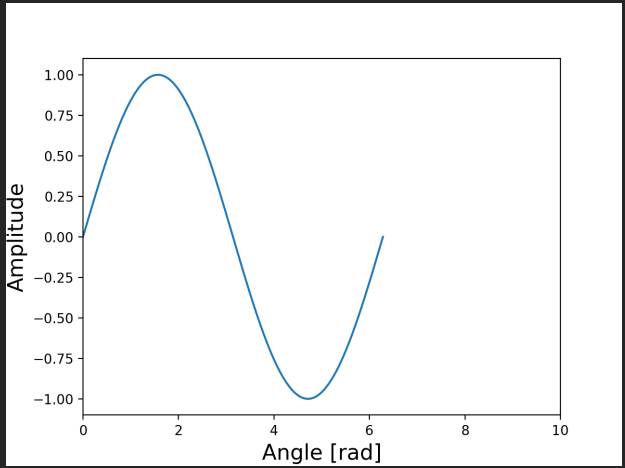
Adjust Limits

```
import matplotlib.pyplot as plt
import numpy as np
def show():
   plt.pause(.1)

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

plt.plot(x, np.sin(x), label='Sine')
plt.xlim(0, 10) ; show()
```



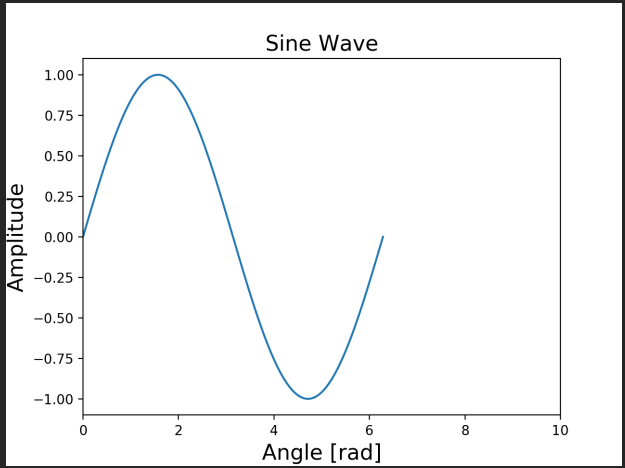
Add Labels

```
import matplotlib.pyplot as plt
import numpy as np
def show():
   plt.pause(.1)

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

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



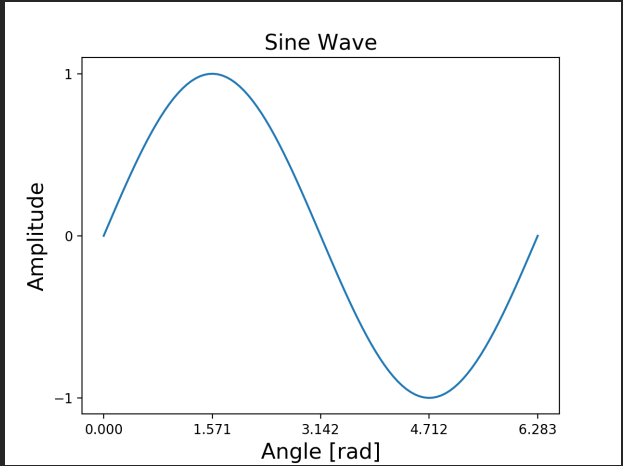
Add Title

```
import matplotlib.pyplot as plt
import numpy as np
def show():
   plt.pause(.1)

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

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



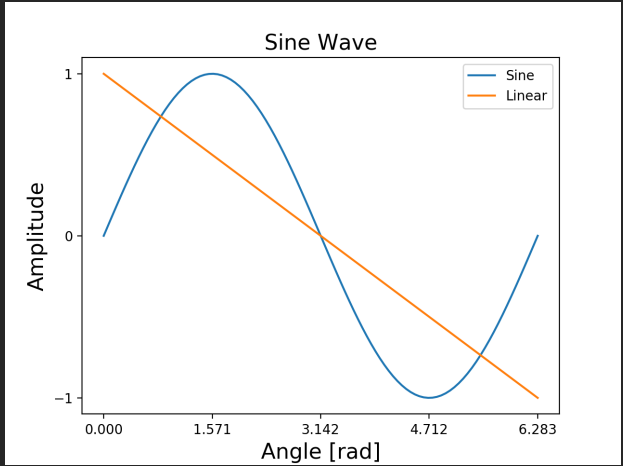
Adjust Ticks

```
import matplotlib.pyplot as plt
import numpy as np
def show():
   plt.pause(.1)

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

plt.plot(x, np.sin(x), label='Sine')
plt.xlim(0, 10)
plt.xlabel('Angle [rad]', fontsize = 16)
plt.ylabel('Amplitude', fontsize = 16)
plt.title('Sine Wave', fontsize = 16)
plt.xlim(0, 2*np.pi)
plt.xticks(np.linspace(0, 2*np.pi, num=5))
plt.yticks(np.linspace(-1, 1, num=3)) ; show()
```



Add Legend

```
import matplotlib.pyplot as plt
import numpy as np
def show():
   plt.pause(.1)

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

plt.plot(x, np.sin(x), label='Sine')
plt.xlim(0, 10)
plt.xlabel('Angle [rad]', fontsize = 16)
plt.ylabel('Amplitude', fontsize = 16)
plt.title('Sine Wave', fontsize = 16)
plt.xlim(0, 2*np.pi)
plt.xticks(np.linspace(0, 2*np.pi, num=5))
plt.yticks(np.linspace(-1, 1, num=3))
plt.plot(x, (-x+np.pi)/np.pi, label='Linear')
plt.legend()
show()
```

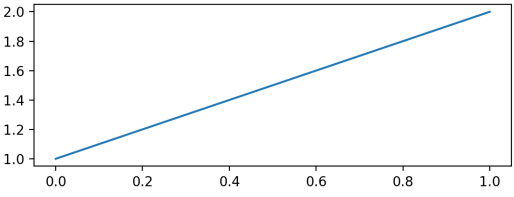


Subplots

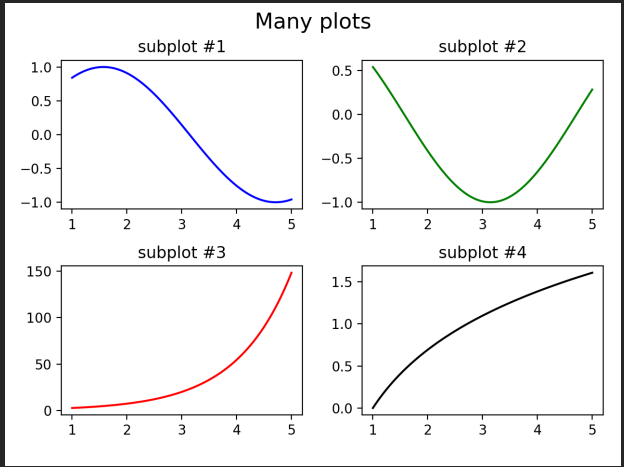
Subplots

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

def show():
   plt.pause(.1)

plt.subplot(2,1,1)
plt.plot([0,1],[1,2])

plt.subplot(2,1,2)
plt.plot([0,1],[2,1])

show()
```



```python
import matplotlib.pyplot as plt
import numpy as np
def show():
   plt.pause(.1)

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.title('subplot #2')
plt.subplot(2,2,3)
plt.plot(x, np.exp(x), color="red")
plt.title('subplot #3')
plt.subplot(2,2,4)
plt.plot(x, np.log(x), color="black")
plt.title('subplot #4')

# 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])

show()
```

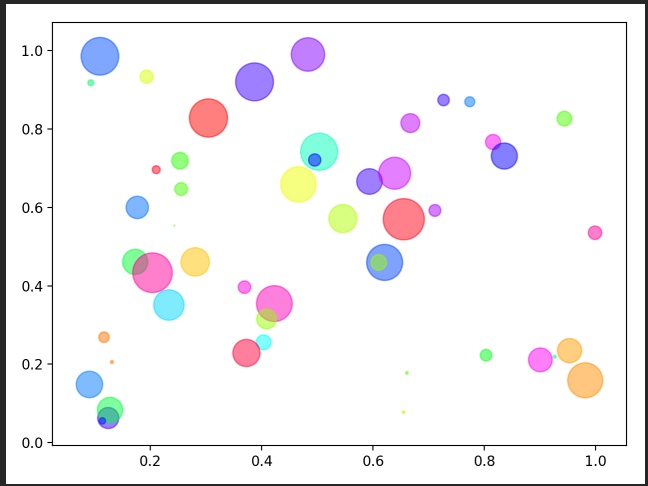


Scatter plots

Scatter plot

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

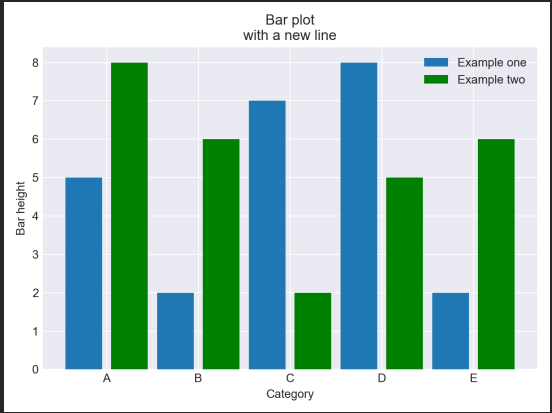
```
import matplotlib.pyplot as plt
import numpy as np
def show():
   plt.pause(.1)

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)

show();
```



Bar plot

Bar plot

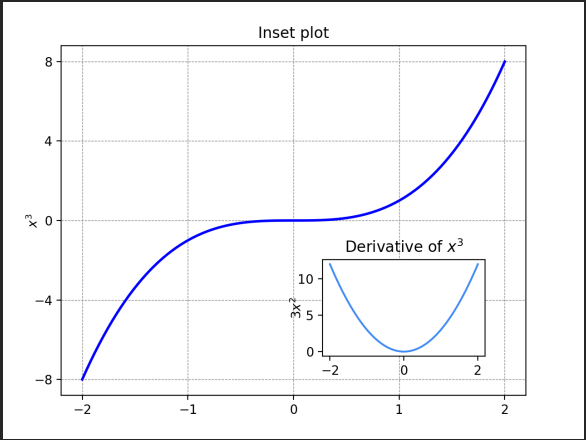
```
import matplotlib.pyplot as plt
import numpy as np
def show():
   plt.pause(.1)

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')

show()
```



Inset plot

Inset plot

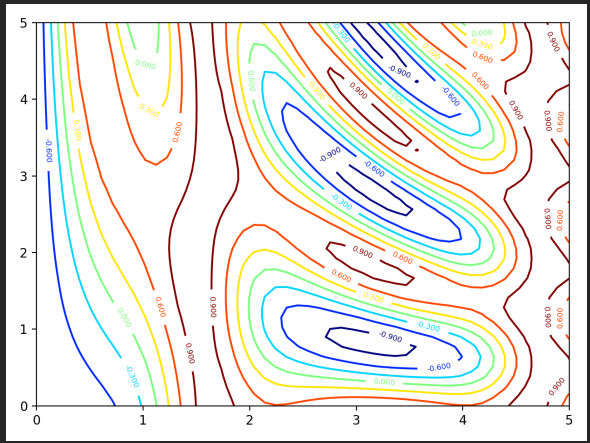
```
import matplotlib.pyplot as plt
import numpy as np
def show():
   plt.pause(.1)

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 $x^3$")

show()
```



Contour plots

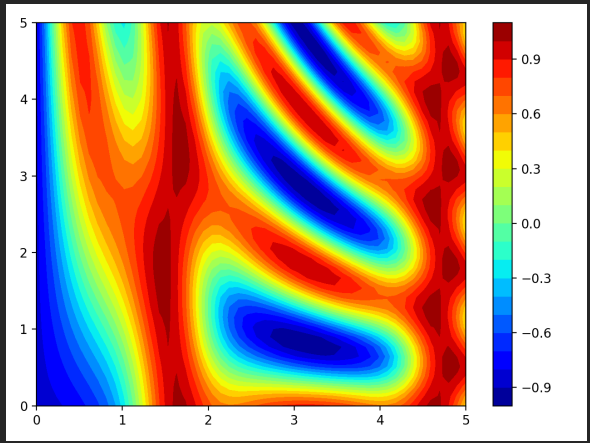
Contour plots

```
import matplotlib.pyplot as plt
import numpy as np
def show():
   plt.pause(.1)

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)

show()
```



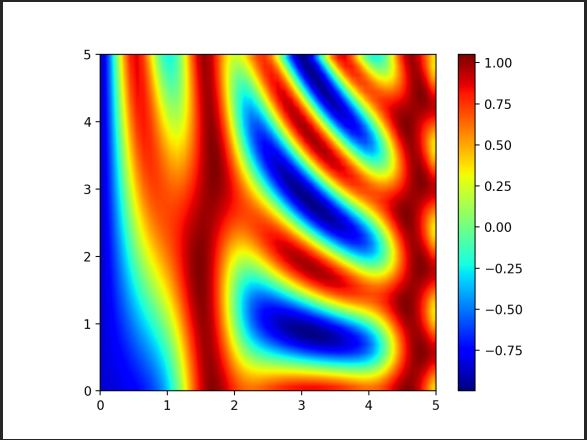
Filled contour plots

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

show()
```



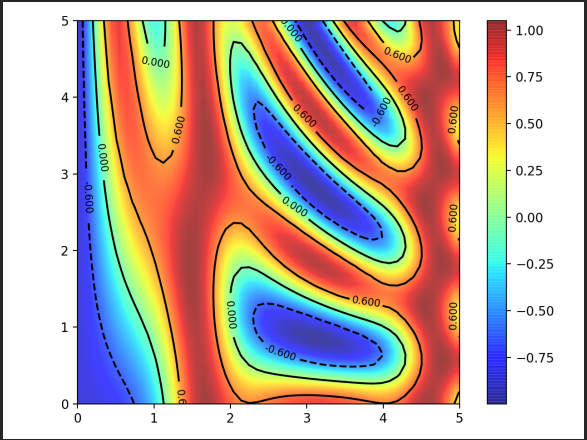
Plot data as an image

```
im=plt.imshow(Z,extent=[0,5,0,5],origin='lower',cmap='jet')
# where extent specifies the limits and
# origin='lower' changes the image origin to the lower left
#  (default: upper left)

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

# make it show
show()
```



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()

show()
```



Other visualization packages

`matplotlib` focuses on generating publication-quality plots

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

- `seaborn` targets data analysis and integrates well with pandas
- `plotnine` is based on the layered "Grammar of Graphics" also used in the `R` package ggplot2.
- `plotly` and `bokeh` focus on interactivity