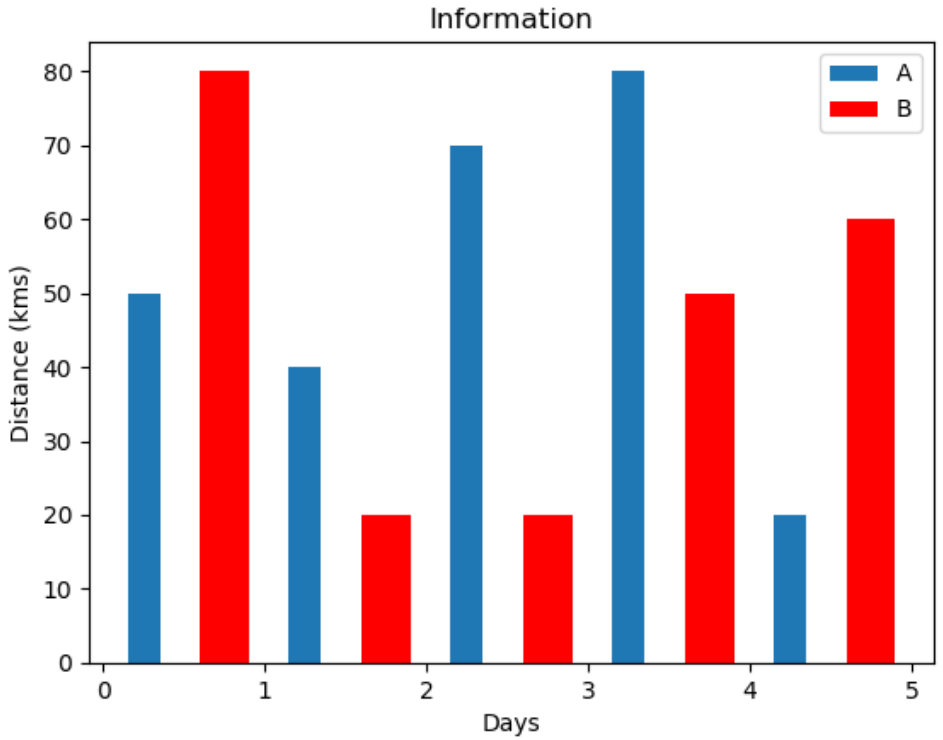
Other types of plots for 1D data

- Besides the simple line plots, there are various other types of plots for 1D data, which can be created using python matplotlib, e.g. bar graph, histogram, scatter plot, area plot, pie plot, etc.

Bar Graph

- A bar graph uses bars to compare data among different categories. It is well suited when you want to measure the changes over a period of time. It can be represented horizontally or vertically. Also, the important thing to keep in mind is that longer the bar, greater is the value.

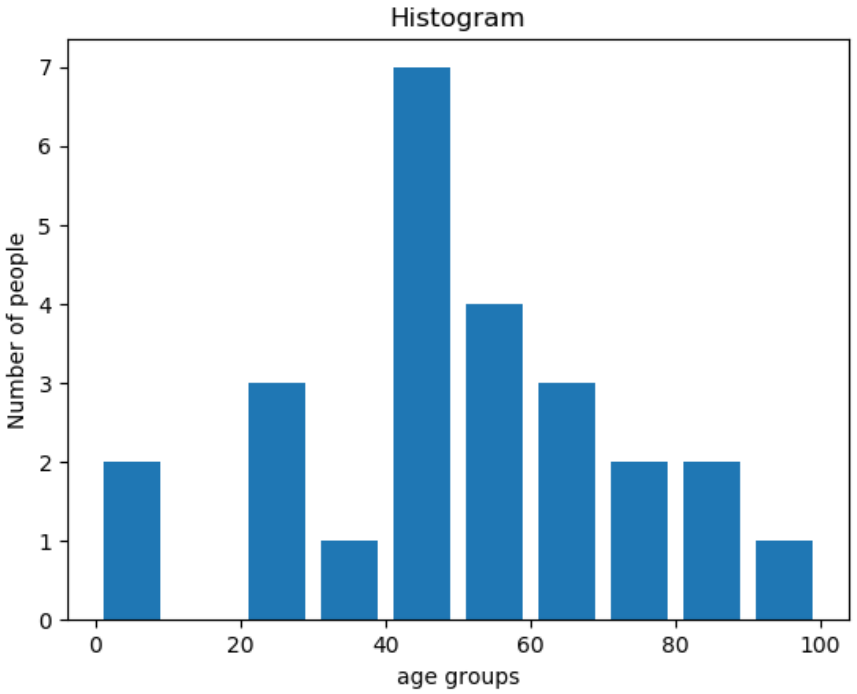
```
from matplotlib import pyplot as plt

plt.bar([0.25,1.25,2.25,3.25,4.25],[50,40,
70,80,20],
label="A",width=.5)
plt.bar([.75,1.75,2.75,3.75,4.75],[80,20,2
0,50,60],
label="B", color='r',width=.5)
plt.legend()
plt.xlabel('Days')
plt.ylabel('Distance (kms)')
plt.title('Information')
plt.show()
```



Histogram

- Histograms are used to show a distribution whereas a bar chart is used to compare different entities. Histograms are useful when you have arrays or a very long list. Let's consider an example where I have to plot the age of population with respect to bin. Now, bin refers to the range of values that are divided into series of intervals. Bins are usually created of the same size.

```
import matplotlib.pyplot as plt
population_age =
[22,55,62,45,21,22,34,42,42,4,2,102,95,85,
55,110,120,70,65,55,111,115,80,75,65,54,4
4,43,42,48]
bins = [0,10,20,30,40,50,60,70,80,90,100]
plt.hist(population_age, bins, histtype='bar',
rwidth=0.8)
plt.xlabel('age groups')
plt.ylabel('Number of people')
plt.title('Histogram')
plt.show()
```



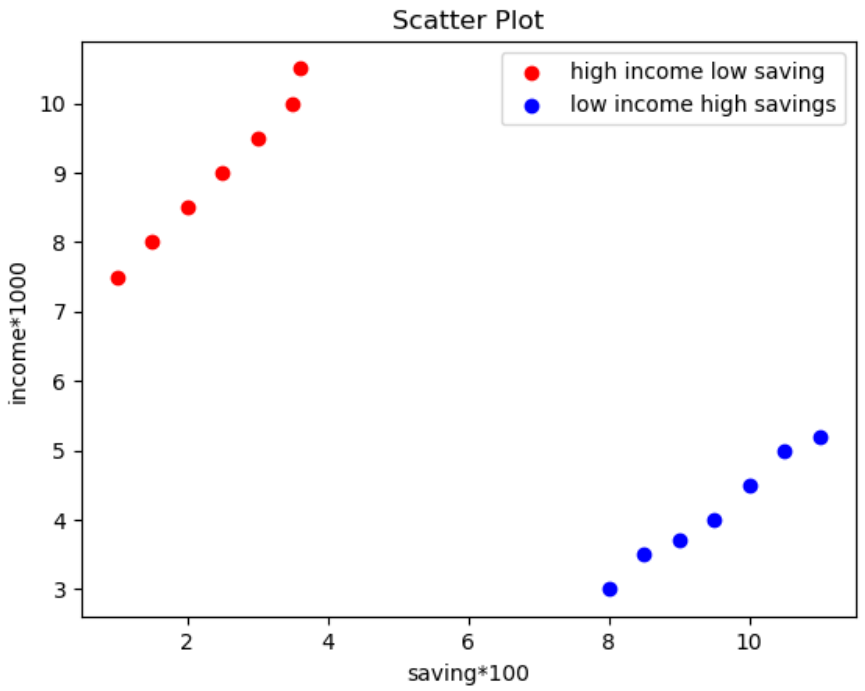
Plot histogram using bar()

```
import matplotlib.pyplot as plt
population_age = [0,10,10,22,55,62,45,21,22,34,
42,42,4,2,102,95,85,55,110,120,70,65,55,111,115,80,75,65,54,44,43,42,48]
bins = [0,10,20,30,40,50,60,70,80,90,100]

plt.figure()
plt.subplot(131)
bins = [0,18,60,100]
plt.hist(population_age, bins, histtype='bar', rwidth=0.5)
plt.xlabel('age groups'); plt.ylabel('Number of people')
plt.title('Histogram'); plt.show()

plt.subplot(132)
Nbin=10
A=plt.hist(population_age, Nbin, histtype='bar', rwidth=0.5)

plt.xlabel('age groups'); plt.ylabel('Number of people')
plt.title('Histogram'); plt.show()

##check the return of plt.hist()
print(type(A)); print(A); print(A[0]); print(A[1])

plt.subplot(133)
plt.bar((A[1][0:-1]+A[1][1:])/2,A[0],width=5)

plt.xlabel('age groups'); plt.ylabel('Number of people')
plt.title('Bar()'); plt.show()
```

Scatter plot

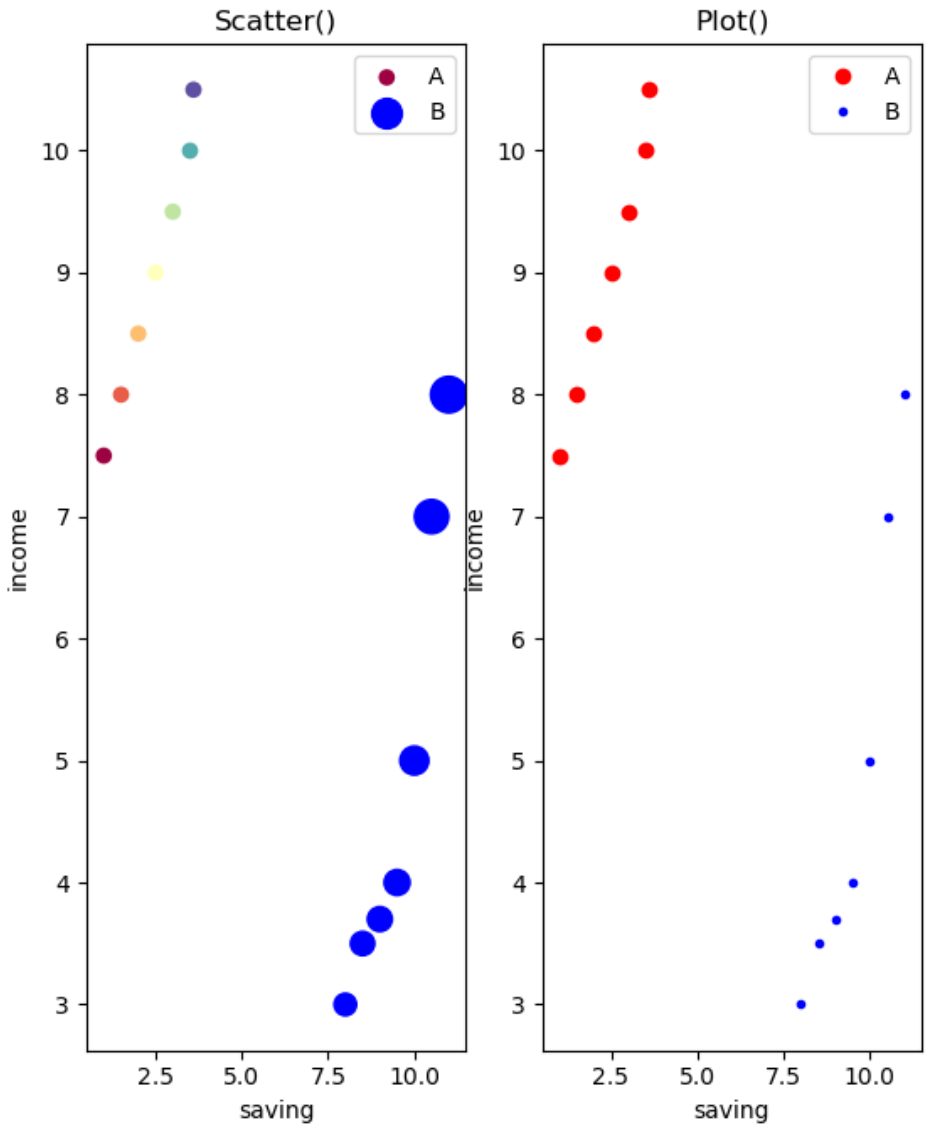
- Usually we need scatter plots in order to compare variables, for example, how much one variable is affected by another variable to build a relation out of it. The data is displayed as a collection of points, each having the value of one variable which determines the position on the horizontal axis and the value of other variable determines the position on the vertical axis.

```
import matplotlib.pyplot as plt
x = [1,1.5,2,2.5,3,3.5,3.6]
y = [7.5,8,8.5,9,9.5,10,10.5]
x1=[8,8.5,9,9.5,10,10.5,11]
y1=[3,3.5,3.7,4,4.5,5,5.2]
plt.scatter(x,y, label='high income low
saving',color='r')
plt.scatter(x1,y1,label='low income high
savings',color='b')
plt.xlabel('saving*100')
plt.ylabel('income*1000')
plt.title('Scatter Plot')
plt.legend(); plt.show();
```



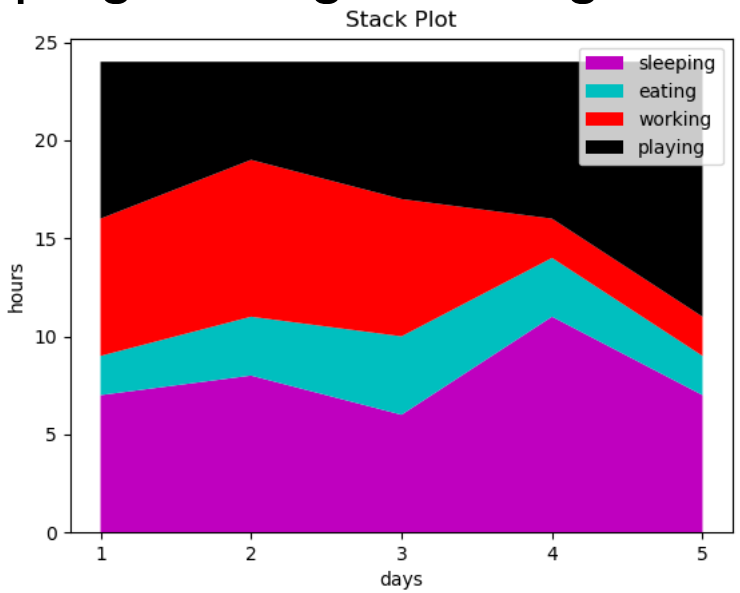
Difference between scatter() and plot()

- The difference between the two functions is: with pyplot.plot() any property you apply (color, shape, size of points) will be applied across all points whereas in pyplot.scatter() you have more control in each point's appearance.
- In plt.scatter() you can have the color, shape and size of each dot (datapoint) to vary based on another variable. Or even the same variable (y). Whereas, with pyplot.plot(), the properties you set will be applied to all the points in the chart.



Area plot

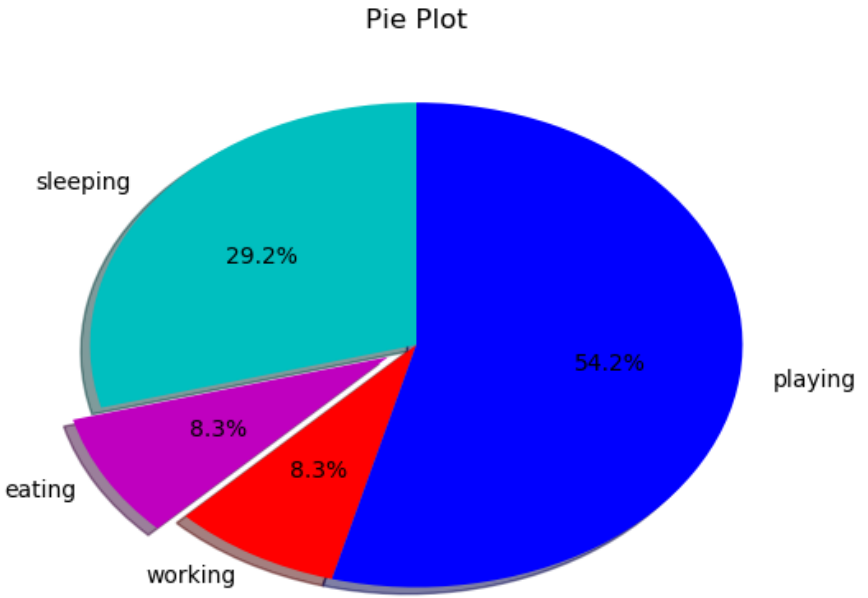
- Area plots are pretty much similar to the line plot. They are also known as stack plots. These plots can be used to track changes over time for two or more related groups that make up one whole category. For example, let's compile the work done during a day into categories, say sleeping, eating, working and playing.




```
import matplotlib.pyplot as plt
days = [1,2,3,4,5]
sleeping =[7,8,6,11,7]; eating = [2,3,4,3,2]
working =[7,8,7,2,2]; playing = [8,5,7,8,13]

# plt.plot([],[],color='m', label='Sleeping', linewidth=5)
# plt.plot([],[],color='c', label='Eating', linewidth=5)
# plt.plot([],[],color='r', label='Working', linewidth=5)
# plt.plot([],[],color='k', label='Playing', linewidth=5)
# plt.stackplot(days, sleeping,eating,working,playing, colors=['m','c','r','k'])
activities = ['sleeping','eating','working','playing']
plt.stackplot(days, sleeping,eating,working,playing,colors=['m','c','r','k'],labels=activities)

plt.xlabel('x'); plt.xticks(days); plt.ylabel('y'); plt.title('Stack Plot'); plt.legend()
```

Pie Chart

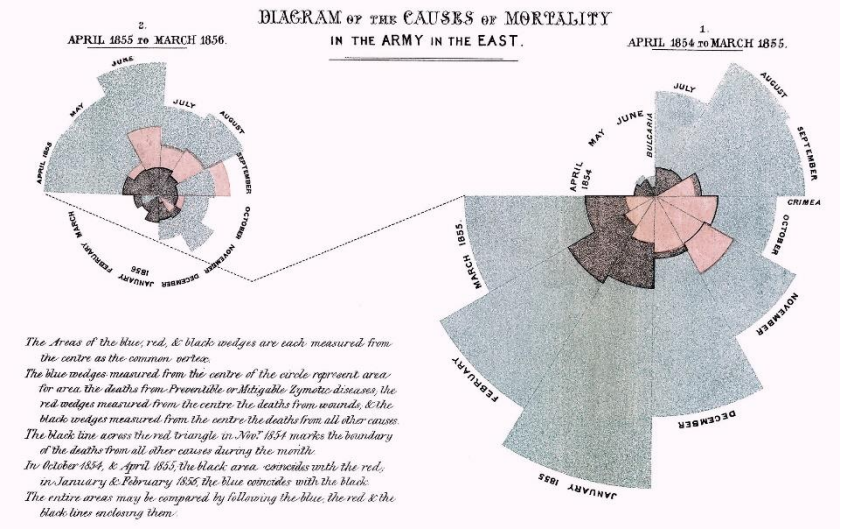
- A pie chart refers to a circular graph which is broken down into segments i.e. slices of pie. It is basically used to show the percentage or proportional data where each slice of pie represents a category.




```
import matplotlib.pyplot as plt

slices = [7,2,2,13]
activities = ['sleeping','eating','working','playing']
cols = ['c','m','r','b']

plt.pie(slices, labels=activities, colors=cols, startangle=90, shadow= True,
explode=(0,0.1,0,0), autopct='%1.1f%%')

plt.title('Pie Plot')
plt.show()
```

Nightingale rose chart

- Nightingale Rose Charts are drawn on a polar coordinate grid. Also known as a Coxcomb Chart, Polar Area Diagram.



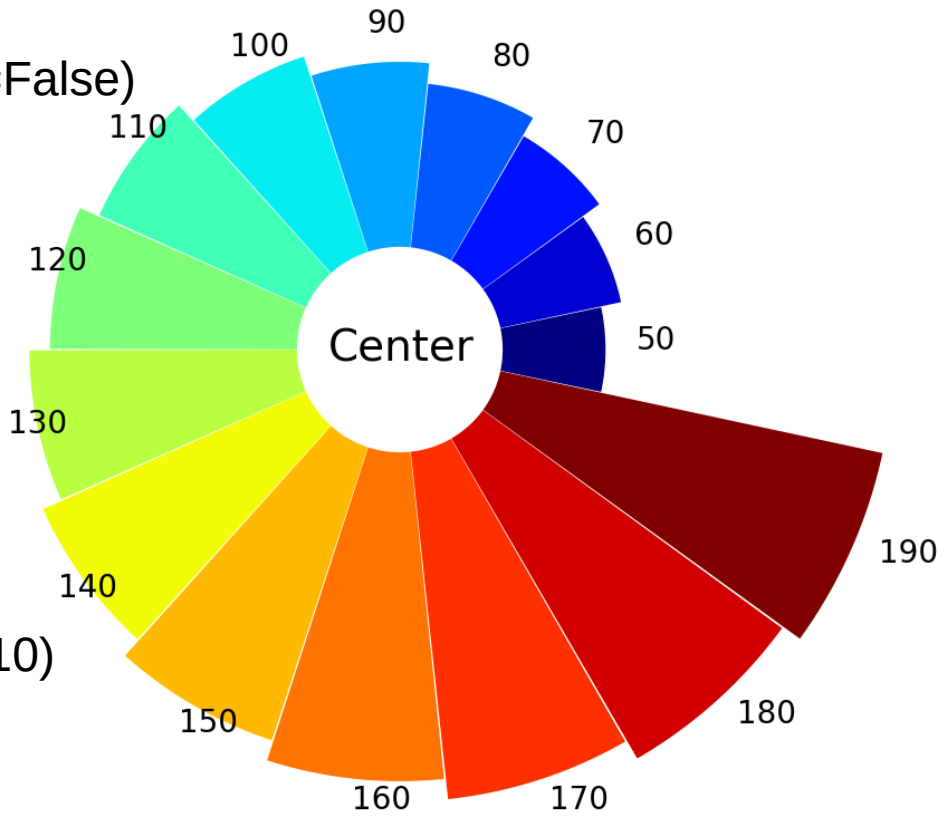
- This chart was famously used by statistician and medical reformer, Florence Nightingale to communicate the avoidable deaths of soldiers during the Crimean war.

- Because the relationship between radius and area is a square relationship, it can exaggerate the proportion of the data, which is especially suitable for comparing values with similar sizes.

- It is not suitable if there are too few categories.

Use matplotlib to plot

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

plt.figure(figsize=(6,6),dpi=200)
plt.subplot(111,projection='polar')
plt.axis('off')

r = np.arange(50,200,10)
theta = np.linspace(0,2*np.pi,len(r),endpoint=False)
angle_scaler=theta[1]*0.99
plt.bar(theta,r,
        width=theta[1]*scaler,
        color=plt.cm.jet(np.linspace(0,1,len(r))),
        bottom=50)

plt.text(-np.pi,35,'Center',fontsize=14)

for ni in range(len(r)):
    plt.text(theta[ni],r[ni]+65,str(r[ni]),fontsize=10)
```



Colormap

- The idea behind choosing a good colormap is to find a good representation in 3D colorspace for your data set. The best colormap for any given data set depends on many things including:

✓ Whether representing form or metric data

✓ Your knowledge of the data set (e.g., is there a critical value from which the other values deviate?)

✓ If there is an intuitive color scheme for the parameter you are plotting

✓ If there is a standard in the field the audience may be expecting

- For many applications, a perceptually uniform colormap is the best choice --- one in which equal steps in data are perceived as equal steps in the color space.
- Color can be represented in 3D space in various ways. One way is using CIELAB. In CIELAB, color space is represented by lightness, L*; red-green, a*; and yellow-blue, b*.

Classes of colormaps

- Colormaps are often split into several categories based on their function:

✓ Sequential: change in lightness and often saturation(饱和度) of color incrementally, often using a single hue(色调); should be used for representing information that has **ordering**.

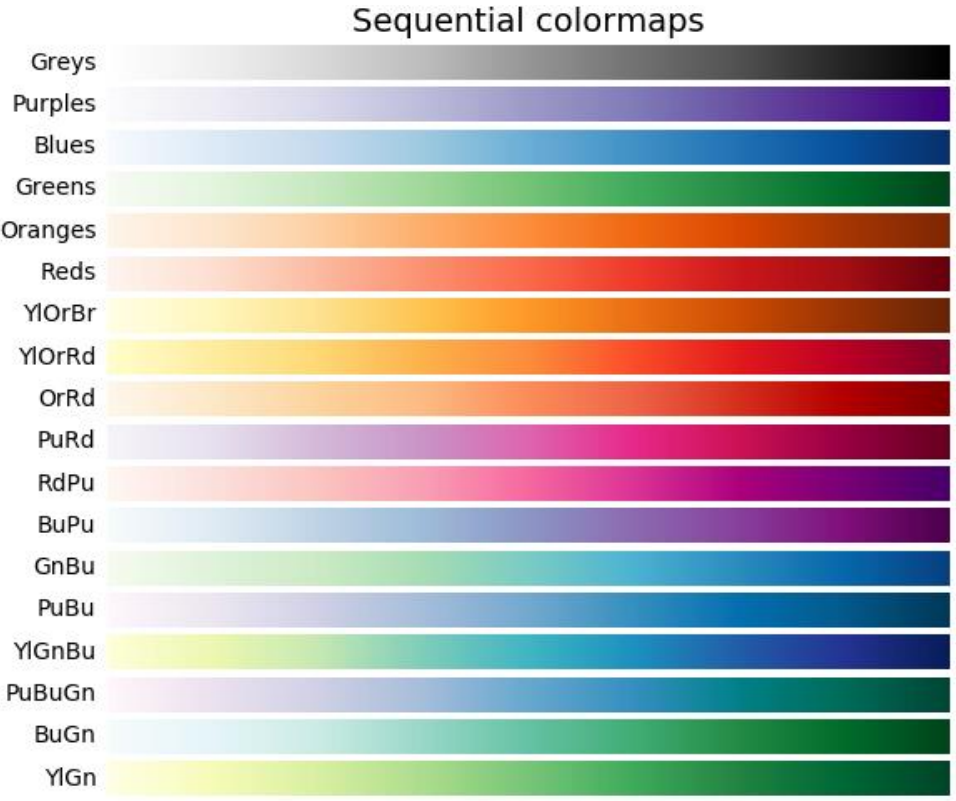
✓ Diverging: change in lightness and possibly saturation of two different colors that meet in the middle at an unsaturated color; should be used when the information being plotted has a **critical middle value**, such as topography or when the data deviates around zero.

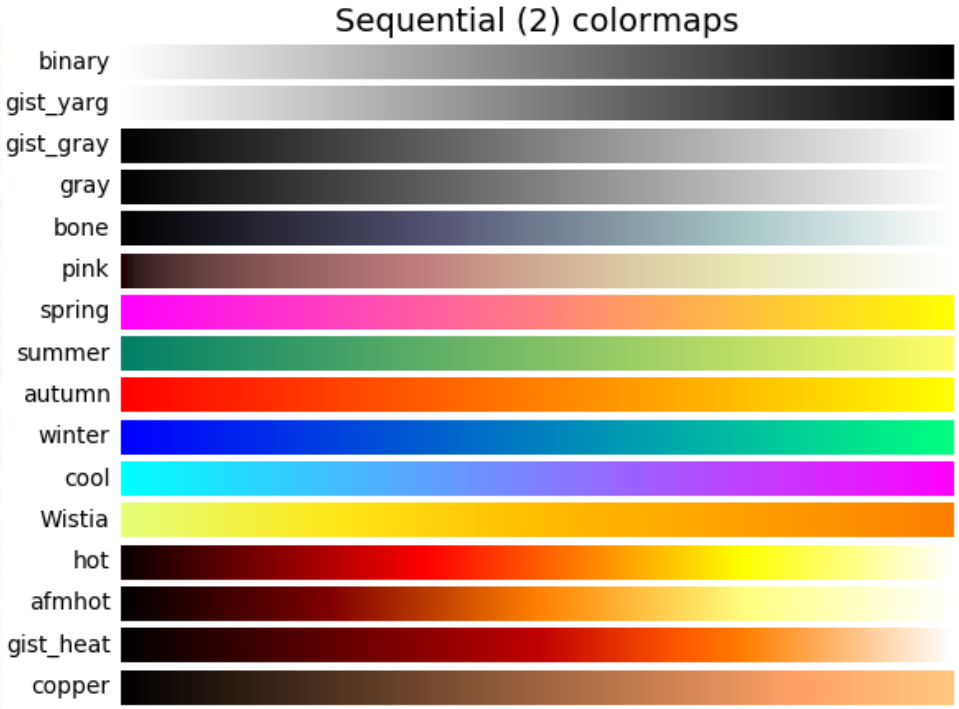
✓ Cyclic: change in lightness of two different colors that meet in the middle and beginning/end at an unsaturated color; should be used for values that wrap around at the endpoints, such as phase angle, wind direction, or time of day.

✓ Qualitative: often are miscellaneous colors; should be used to represent information which does not have ordering or relationships.

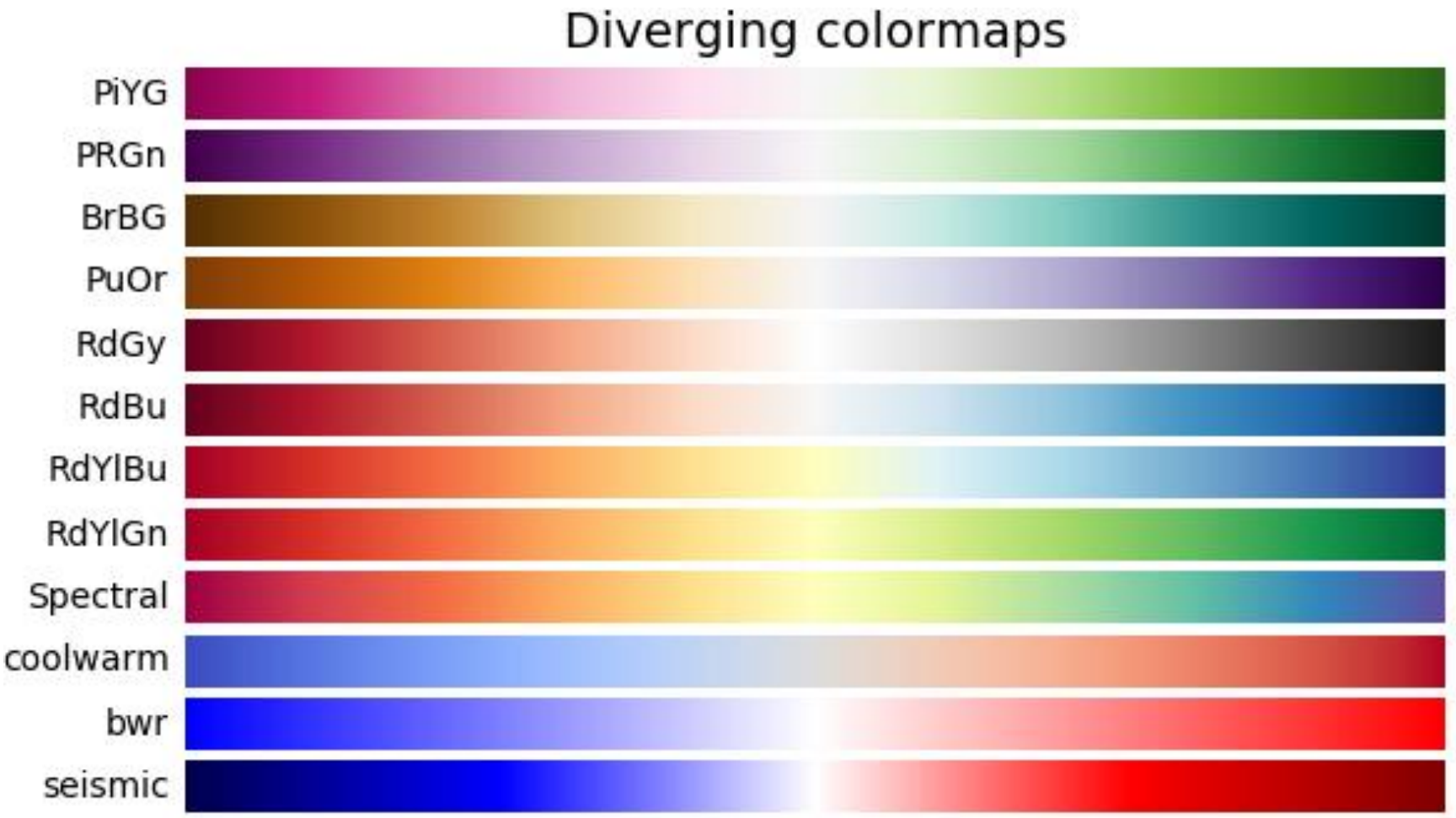
Sequential colormaps

- Different sequential colormaps.

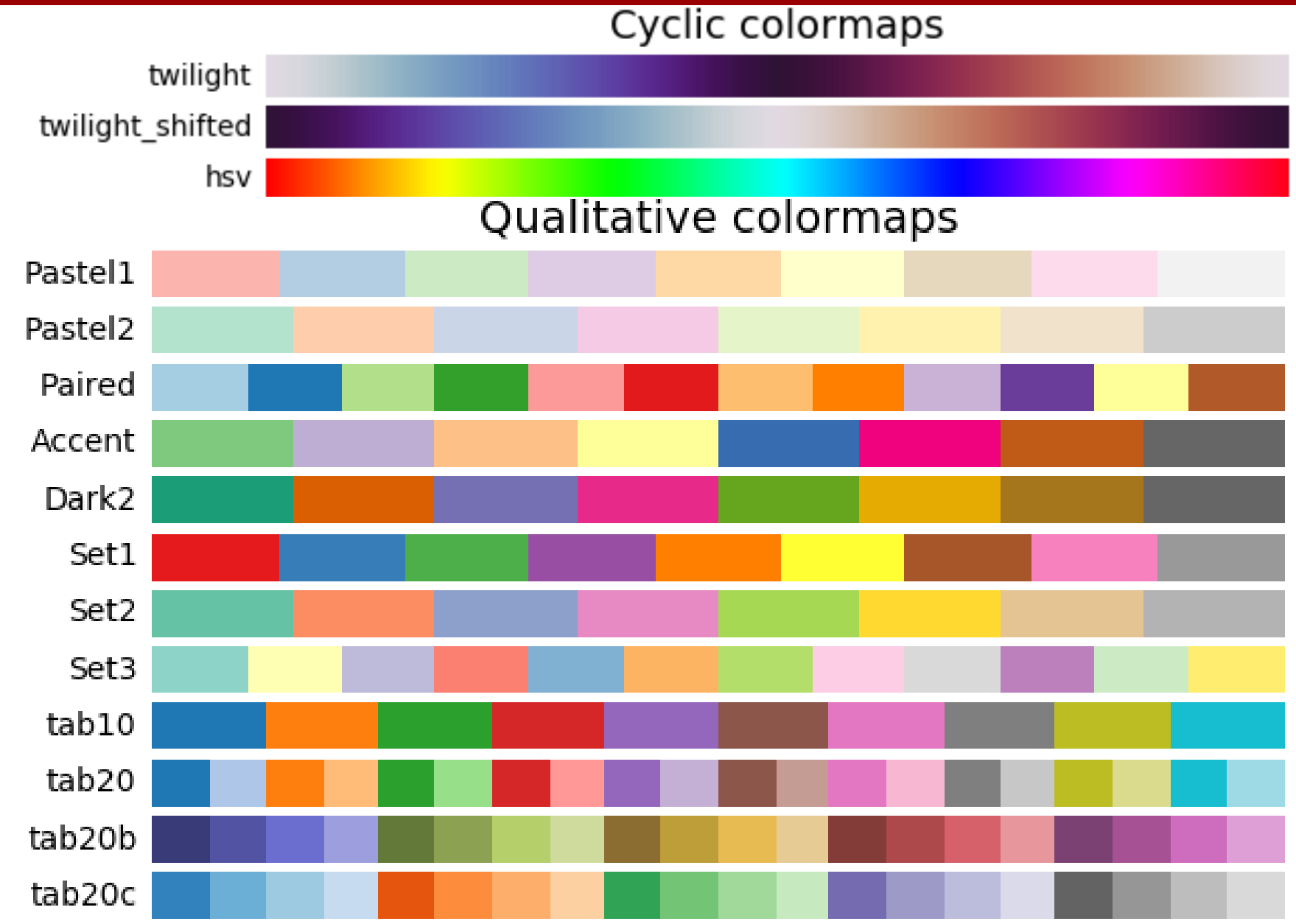




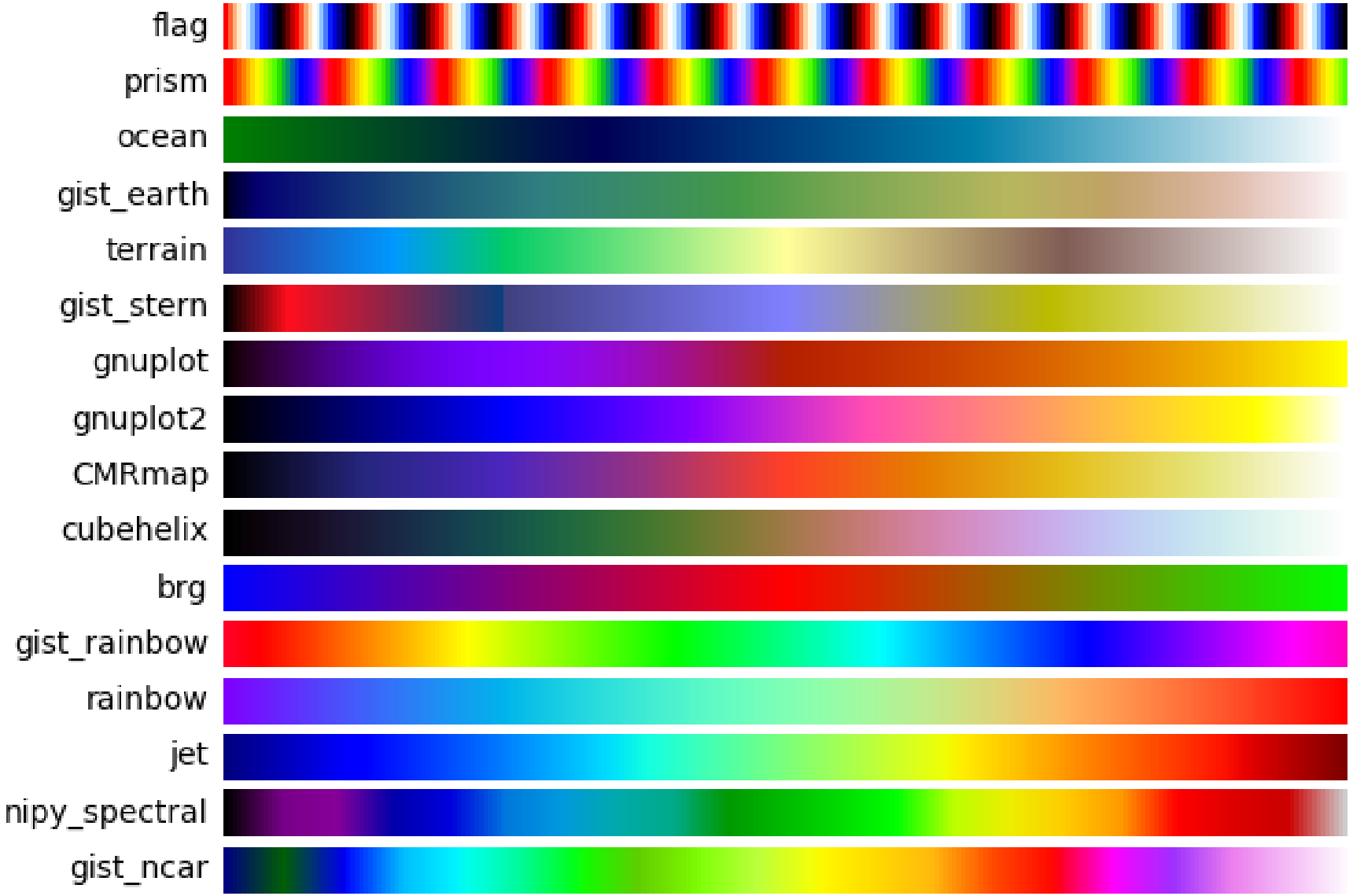
Diverging colormaps



Cyclic and qualitative colormaps



Miscellaneous colormaps

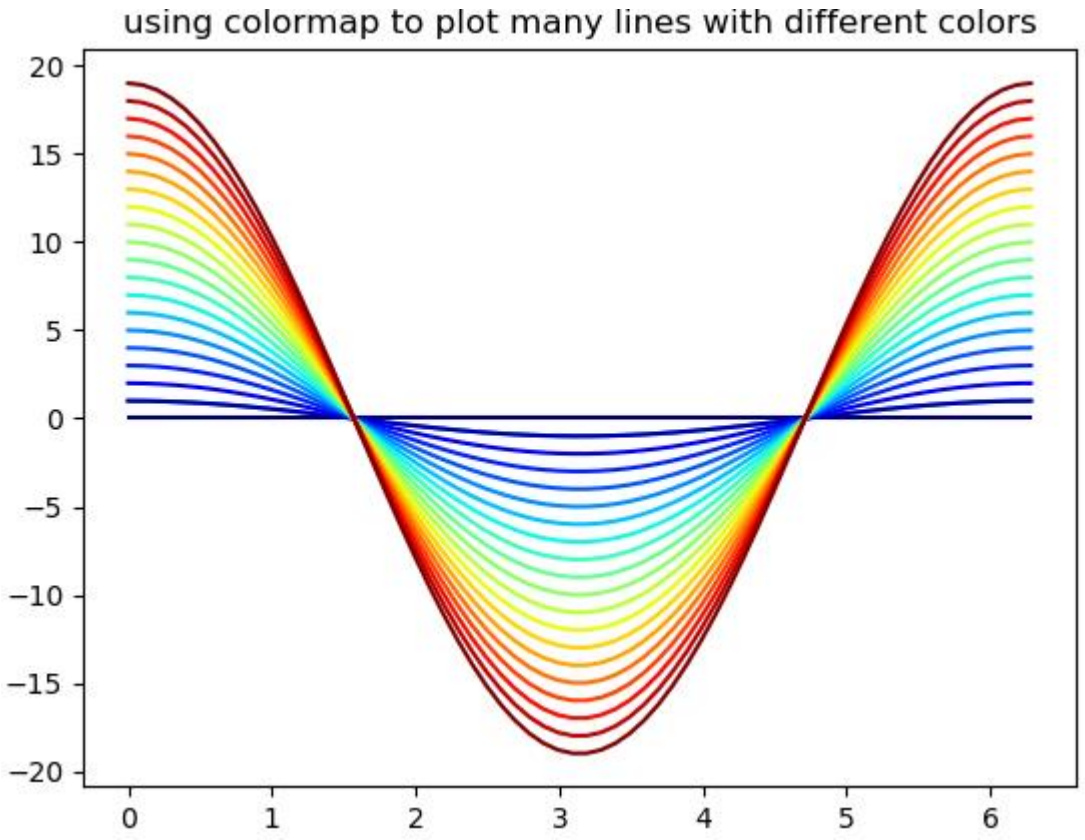


Find the RGB value for a given color

- https://htmlcolorcodes.com/color-chart/
- https://www.w3schools.com/colors/default.asp

Using colormap

```
import numpy as np
#import matplotlib.pylab as pl
import matplotlib.pyplot as plt


x = np.linspace(0, 2*np.pi, 64)
y = np.cos(x)

plt.figure()
plt.plot(x,y)

n = 20
colors = plt.cm.jet(np.linspace(0,1,n))

for i in range(n):
    plt.plot(x, i*y, color=colors[i])

plt.title('using colormap to plot many lines with different colors')
```



Plot for 2D array

- In Matplotlib, we can use imshow() or matshow(), pcolor() or pcolormesh() and contour() or contour() to generate 2-D image-style plots for a 2D array, even if the horizontal dimensions are unevenly spaced.

- ***matplotlib.pyplot.imshow(X, cmap=None, norm=None, aspect=None, interpolation=None, alpha=None, vmin=None, vmax=None, origin=None, extent=None, shape=None, filternorm=1, filterrad=4.0, imlim=None, resample=None, url=None, *, data=None, **kwargs)***

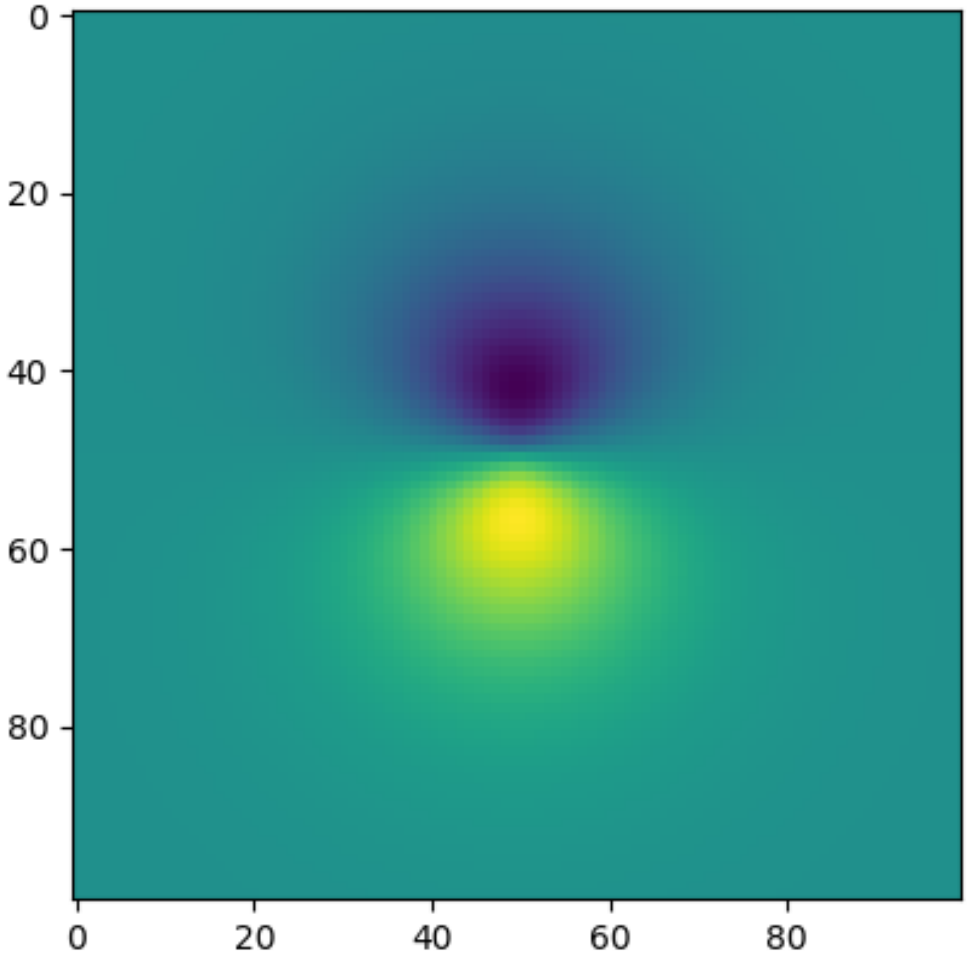
- ***matplotlib.pyplot.pcolormesh(*args, alpha=None, norm=None, cmap=None, vmin=None, vmax=None, shading='flat', antialiased=False, data=None, **kwargs)***

- ***matplotlib.pyplot.contour(*args, data=None, **kwargs)***

- ***matplotlib.pyplot.contourf(*args, data=None, **kwargs)***

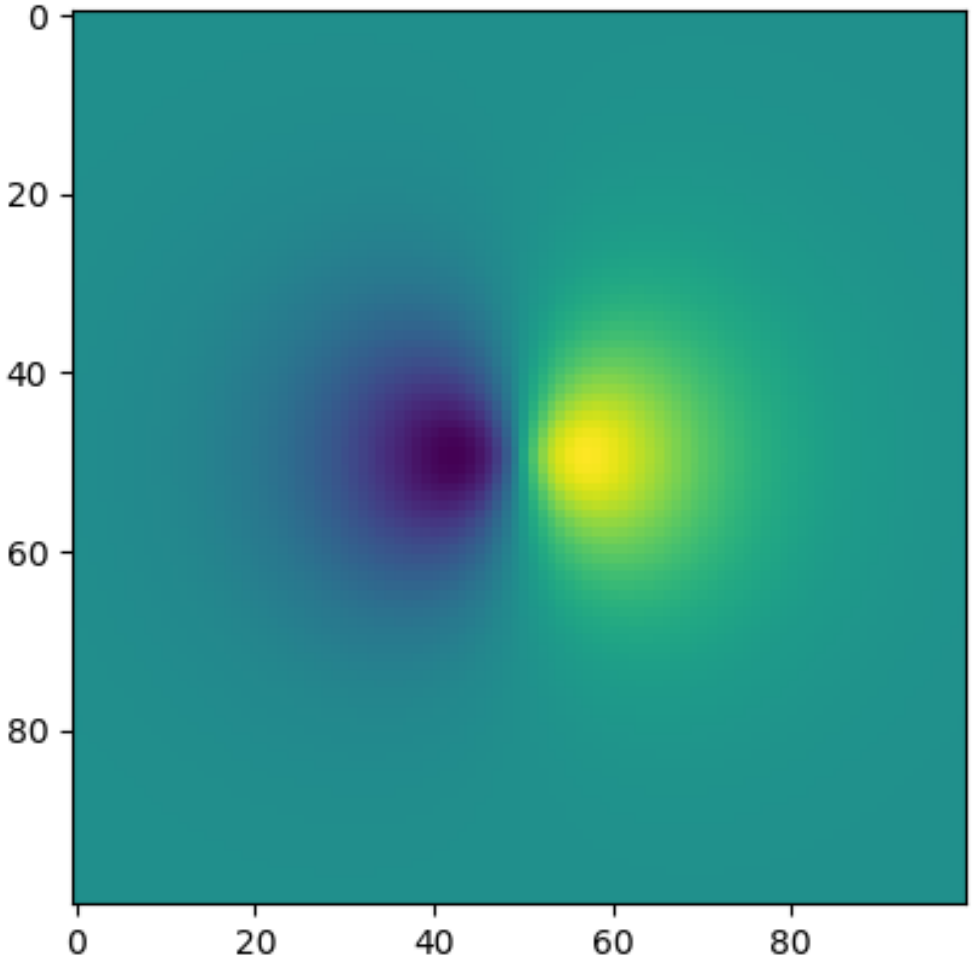
Example: p orbital

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

epsilon=10E-10; Z=1; N=3
def func_px(x,y,z):
    r=np.sqrt(x**2+y**2+z**2)
    if r<epsilon:
        return 0
    rho=2*Z*r/N
    R2p=(1/2/np.sqrt(6))*rho*Z**3/2*np.exp(-rho/2)
    Y2px=np.sqrt(3)*x/r/np.sqrt(4*np.pi)
    return R2p*Y2px

x_min=-10; x_max=10; y_min=-10; y_max=10;z=0

num_x=40; num_y=40
px=np.zeros([num_x,num_y],float)
xx=np.zeros([num_x,num_y],float);
yy=np.zeros([num_x,num_y],float)
for nx in range(num_x):
    for ny in range(num_y):
        px[nx,ny]=func_px(nx/(num_x-1)*(x_max-x_min)+x_min,ny/(num_y-1)*(y_max-y_min)+y_min,z)

plt.imshow(px)
```



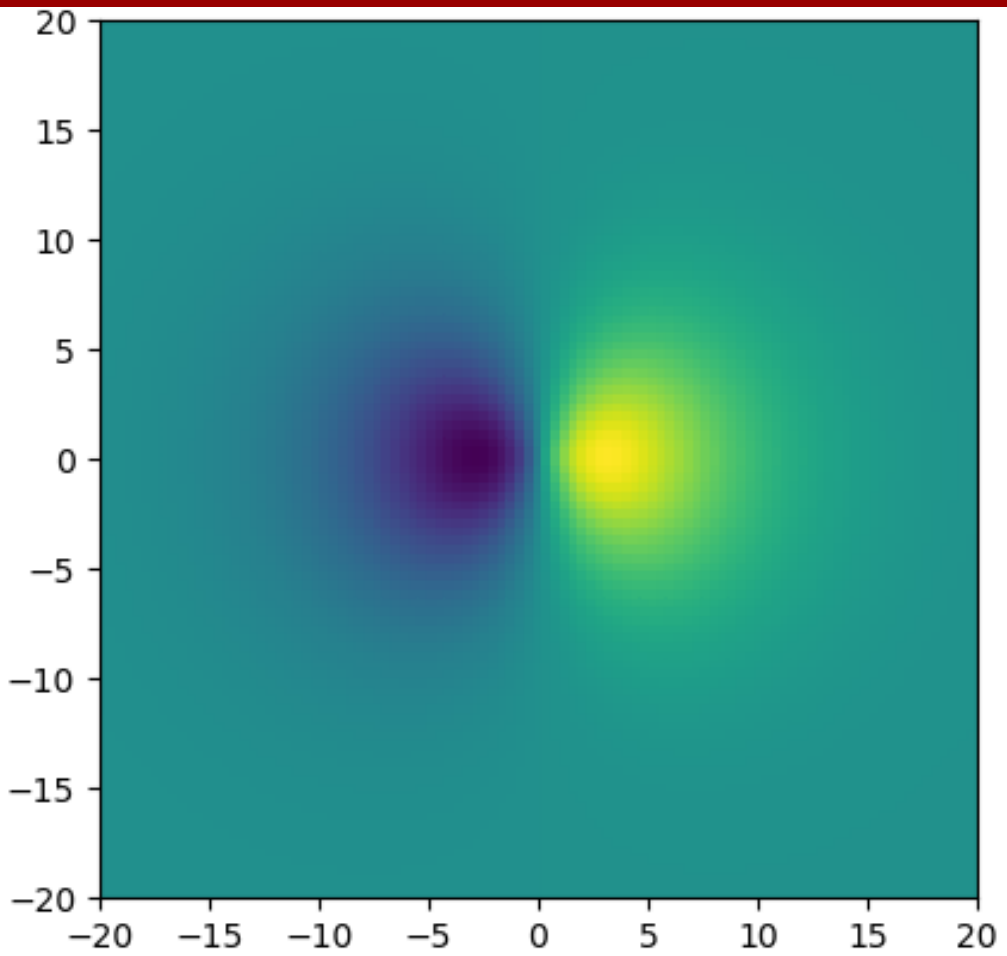
Make the order of X and Y correct

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

epsilon=10E-10; Z=1; N=3
def func_px(x,y,z):
    r=np.sqrt(x**2+y**2+z**2)
    if r<epsilon:
        return 0
    rho=2*Z*r/N
    R2p=(1/2/np.sqrt(6))*rho*Z**3/2*np.exp(-rho/2)
    Y2px=np.sqrt(3)*x/r/np.sqrt(4*np.pi)
    return R2p*Y2px

x_min=-10; x_max=10; y_min=-10; y_max=10;z=0

num_x=40; num_y=40
px=np.zeros([num_x,num_y],float)
xx=np.zeros([num_x,num_y],float);
yy=np.zeros([num_x,num_y],float)
for nx in range(num_x):
    for ny in range(num_y):
        px[ny,nx]=fun_px(nx/(num_x-1)*(x_max-x_min)+x_min,ny/(num_y-1)*(y_max-y_min)+y_min,z)

plt.imshow(px)
```



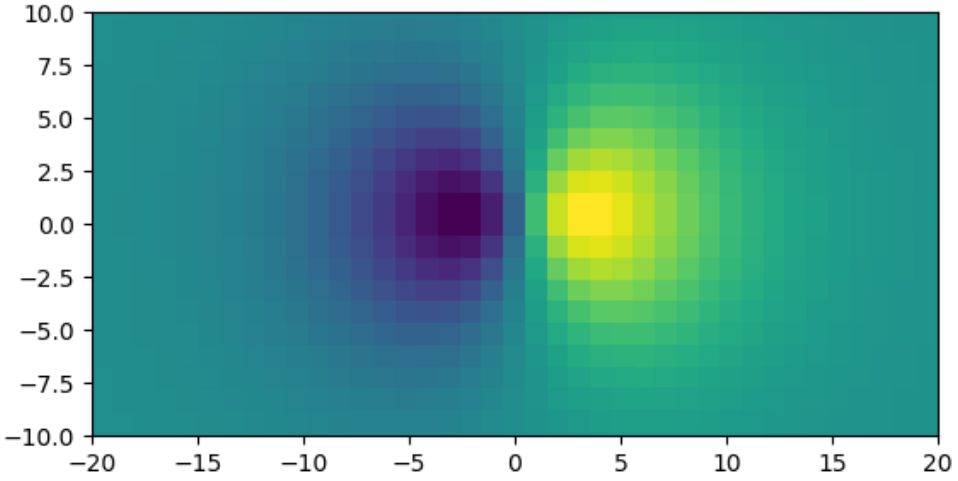
Make the value of X and Y correct

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

epsilon=10E-10; Z=1; N=3
def func_px(x,y,z):
    r=np.sqrt(x**2+y**2+z**2)
    if r<epsilon:
        return 0
    rho=2*Z*r/N
    R2p=(1/2/np.sqrt(6))*rho*Z**3/2*np.exp(-rho/2)
    Y2px=np.sqrt(3)*x/r/np.sqrt(4*np.pi)
    return R2p*Y2px

x_min=-10; x_max=10; y_min=-10; y_max=10;z=0

num_x=40; num_y=40
px=np.zeros([num_x,num_y],float)
xx=np.zeros([num_x,num_y],float);
yy=np.zeros([num_x,num_y],float)
for nx in range(num_x):
    for ny in range(num_y):
        xx[nx,ny]=nx/(num_x-1)*(x_max-x_min)+x_min
        yy[nx,ny]=ny/(num_y-1)*(y_max-y_min)+y_min
        px[nx,ny]=fun_px(nx/(num_x-1)*(x_max-x_min)+x_min,ny/(num_y-1)*(y_max-y_min)+y_min,z)

plt.pcolor(xx,yy,px); plt.axis('square')
```



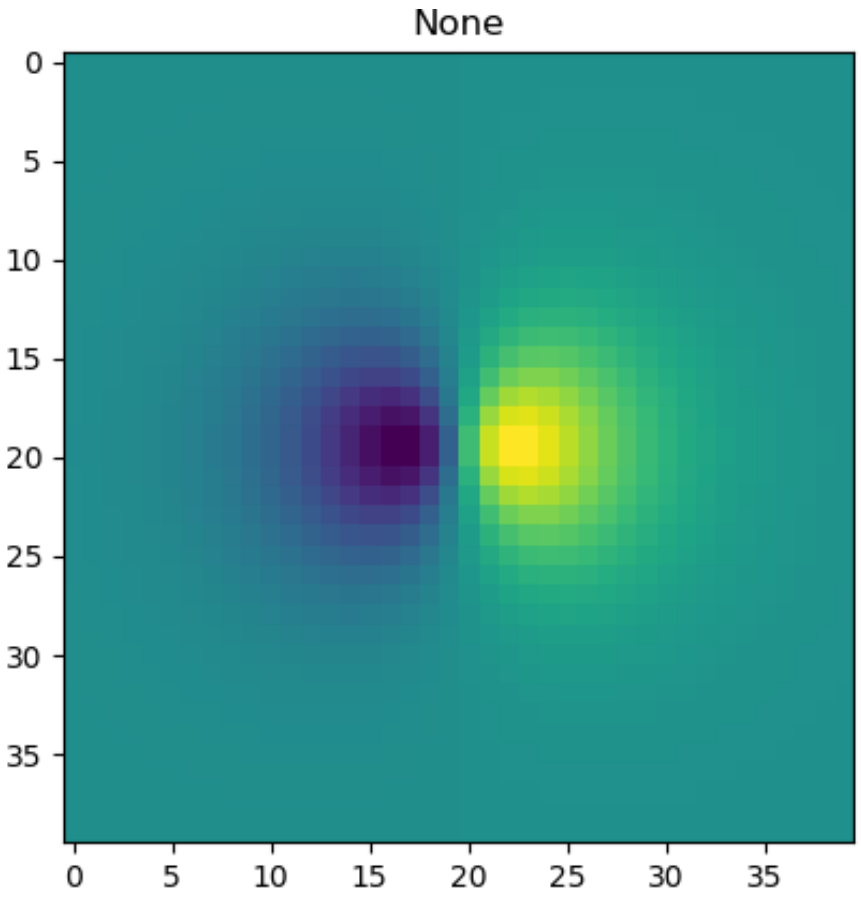
Set the limits of the current axes

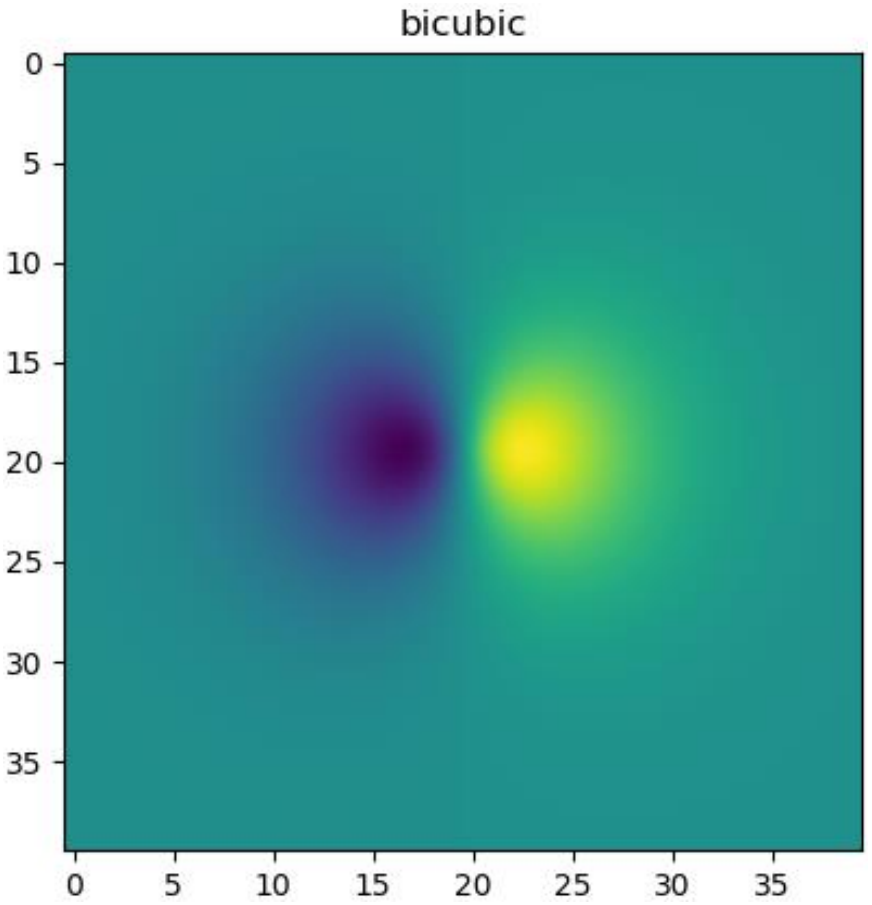
```
import numpy as np
import matplotlib.pyplot as plt

epsilon=10E-10; Z=1; N=3
def func_px(x,y,z):
    r=np.sqrt(x**2+y**2+z**2)
    if r<epsilon:
        return 0
    rho=2*Z*r/N
    R2p=(1/2/np.sqrt(6))*rho*Z**3/2*np.exp(-rho/2)
    Y2px=np.sqrt(3)*x/r/np.sqrt(4*np.pi)
    return R2p*Y2px

x_min=-10; x_max=10; y_min=-10; y_max=10;z=0

num_x=40; num_y=40
px=np.zeros([num_x,num_y],float); xx=np.zeros([num_x,num_y],float);
yy=np.zeros([num_x,num_y],float)
for nx in range(num_x):
    for ny in range(num_y):
        xx[nx,ny]=nx/(num_x-1)*(x_max-x_min)+x_min
        yy[nx,ny]=ny/(num_y-1)*(y_max-y_min)+y_min
        px[nx,ny]=fun_px(nx/(num_x-1)*(x_max-x_min)+x_min,ny/(num_y-1)*(y_max-y_min)+y_min,z)

plt.pcolor(xx,yy,px); plt.axis('square'); plt.ylim([-10,10]);
```



Interpolations for imshow/matshow

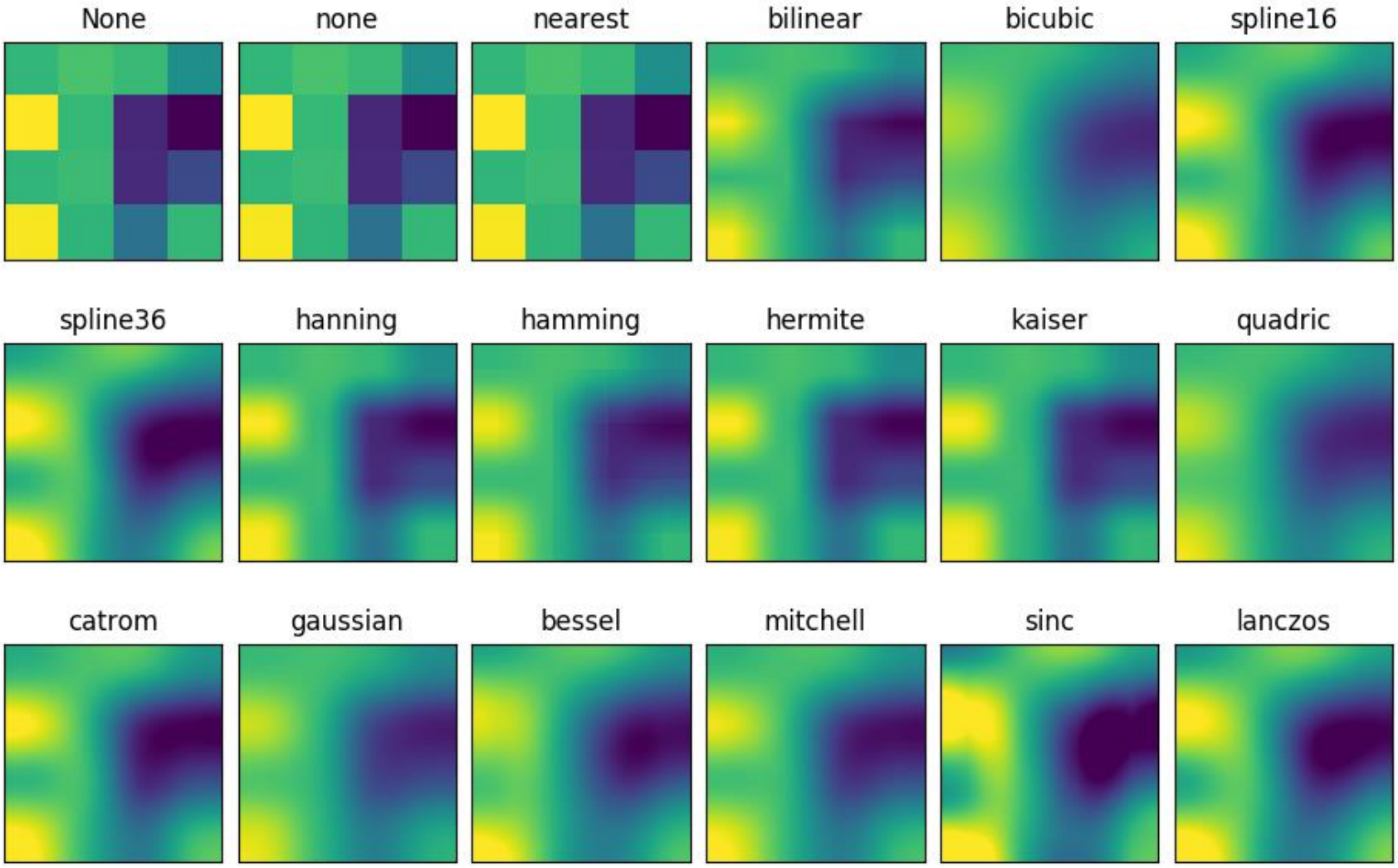
- We can use keyword **interpolation** to change the type of interpolation for imshow() or matshow()





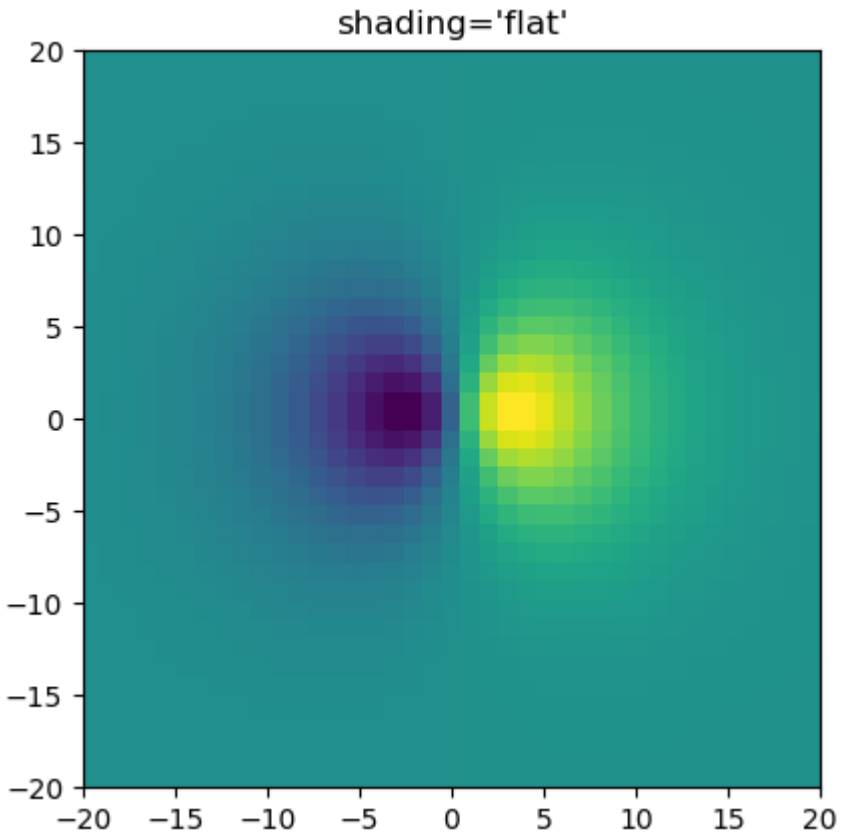
- The keyword **interpolation** cannot be used in pcolor().

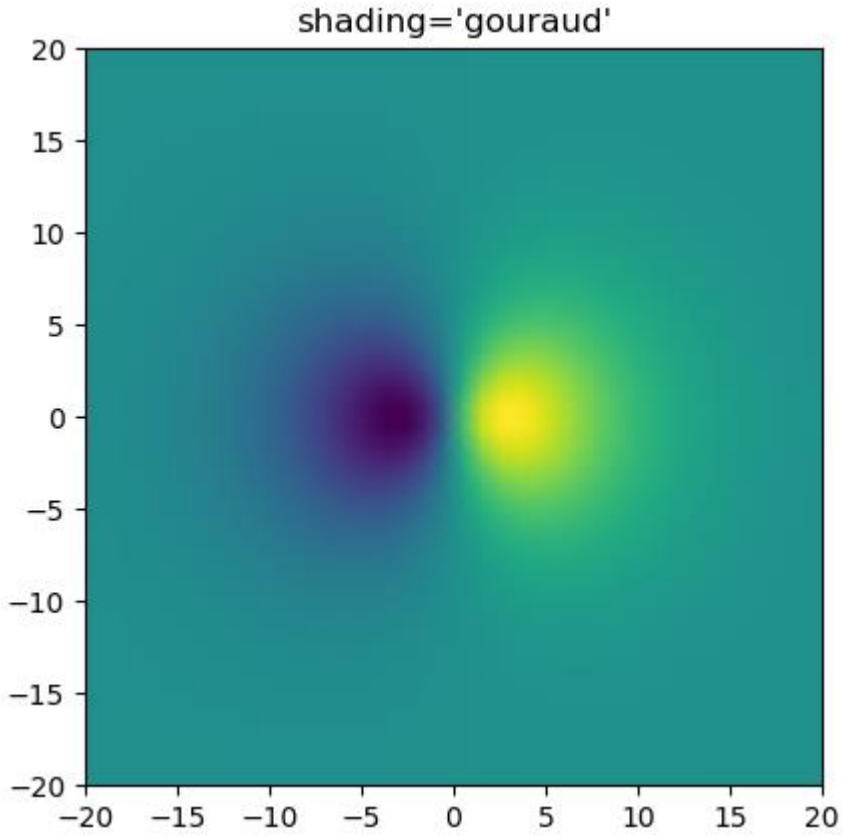
Different choices of interpolation



Shading for pcolormesh()

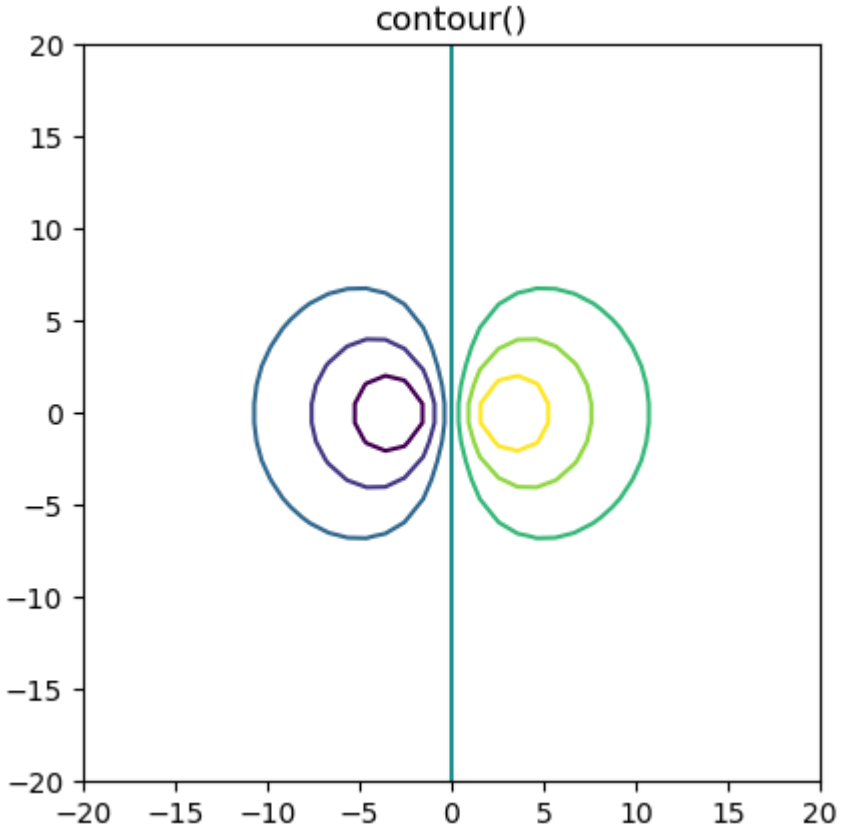
- **shading:** {'flat', 'gouraud'}, optional



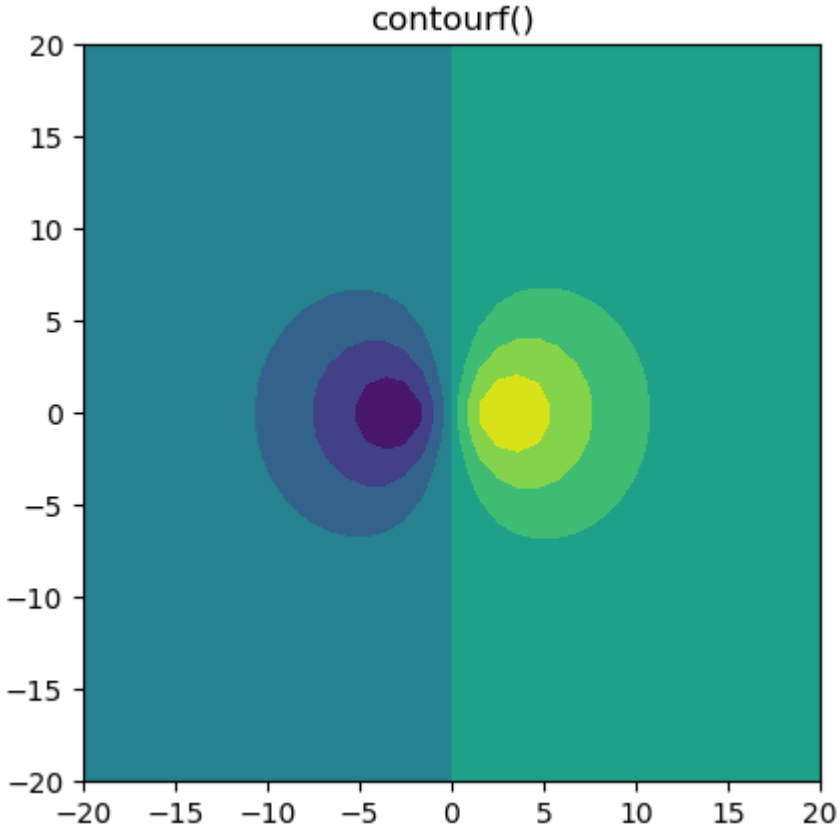


Contour plot

- ***matplotlib.pyplot.contour(*args, data=None, **kwargs)***
- ***matplotlib.pyplot.contourf(*args, data=None, **kwargs)***



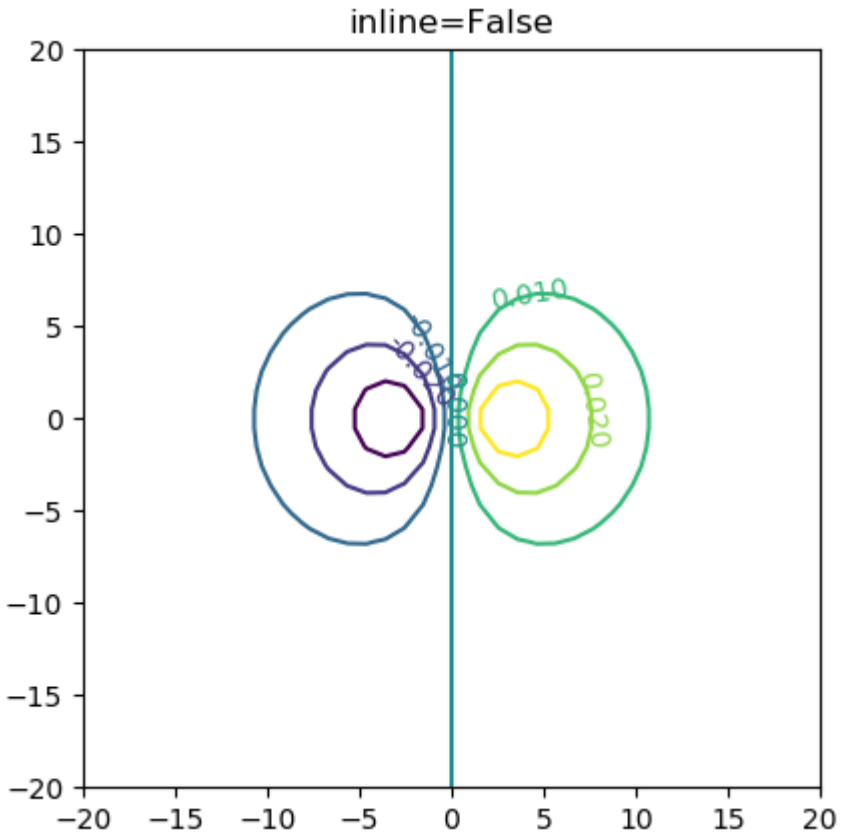
plt.contour(xx,yy,px)

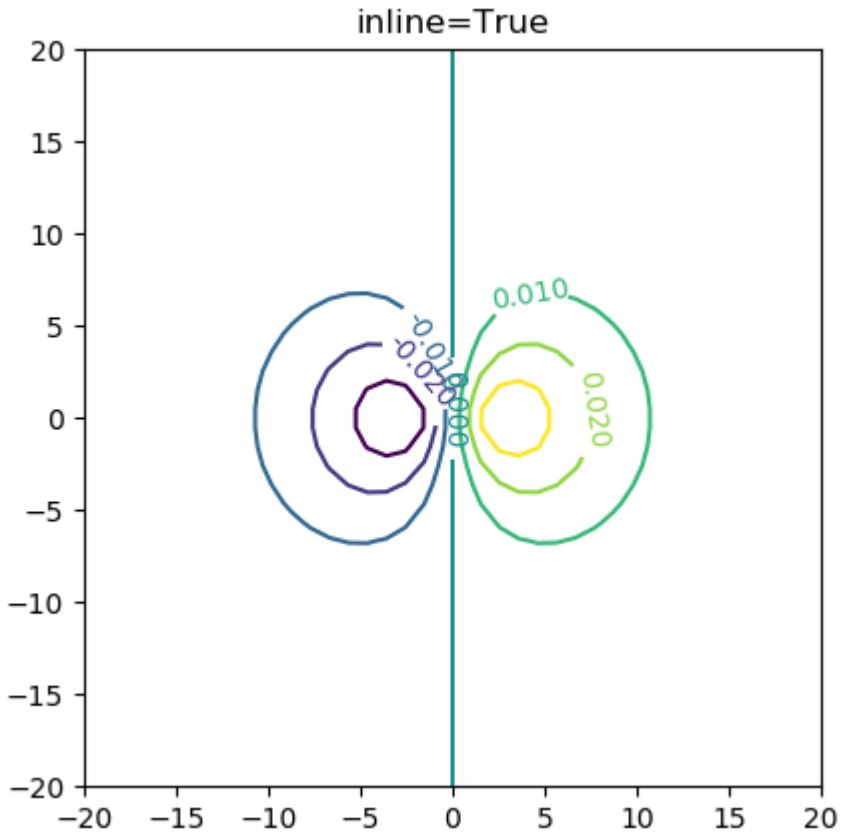


plt.contourf(xx,yy,px)

Show the values of different levels

CS=plt.contour(xx,yy,px)
plt.clabel(CS,CS.levels, inline=False, fontsize=10))

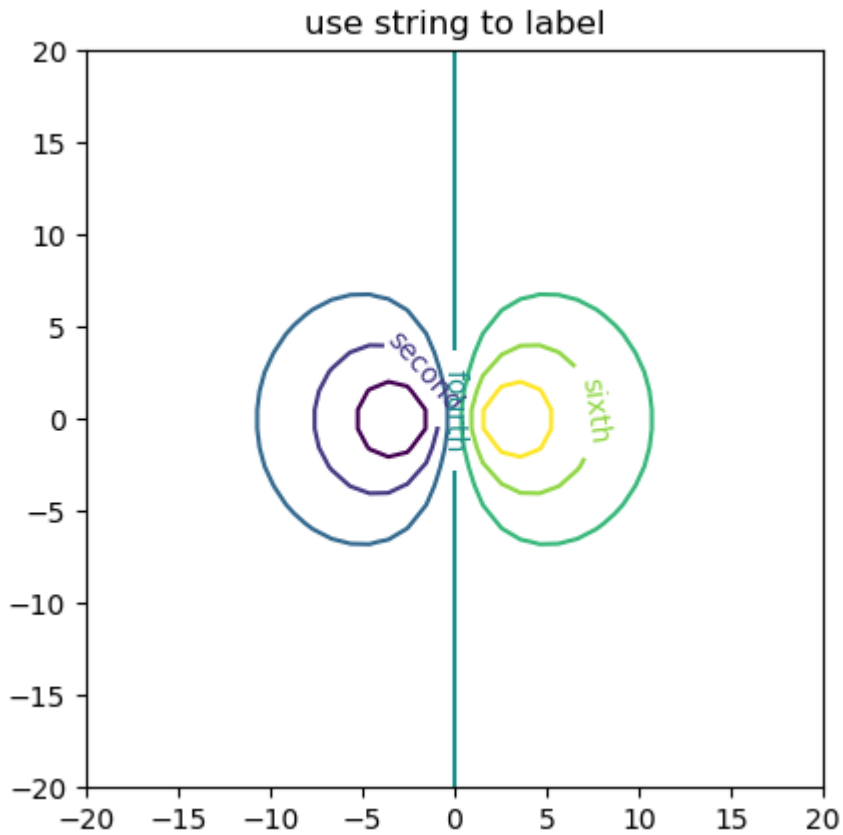




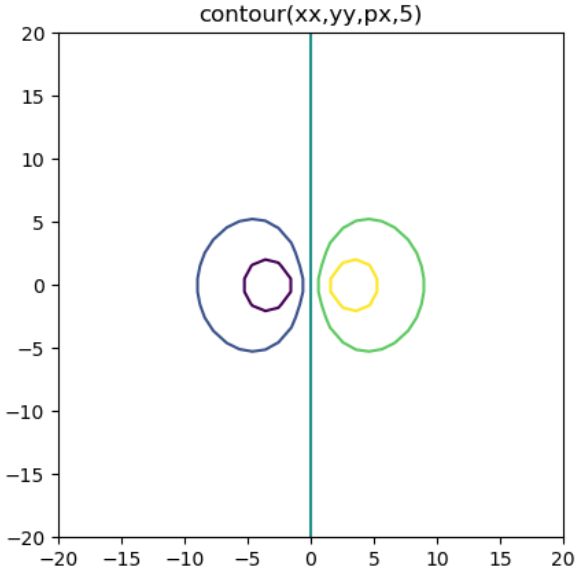
CS=plt.contour(xx,yy,px)
plt.clabel(CS,CS.levels, inline=True, fontsize=10))

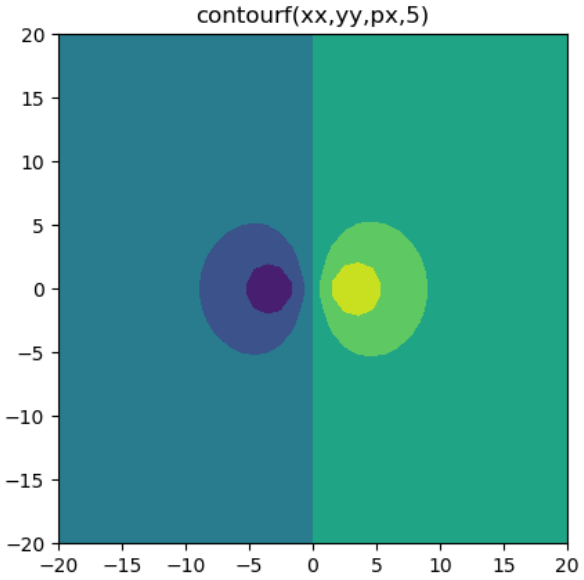
Show string to label different levels

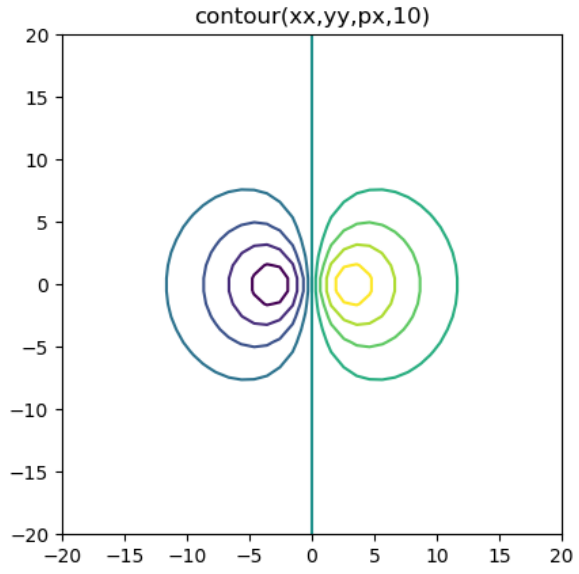
```
CS=plt.contour(xx,yy,px,7)
plt.axis('square')
plt.title('use string to label')

fmt = {}
strs = ['first', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh']
for l, s in zip(CS.levels, strs):
    fmt[l] = s

# Label every other level using strings
plt.clabel(CS, CS.levels[1:7:2], inline=True,
fmt=fmt, fontsize=10)
```

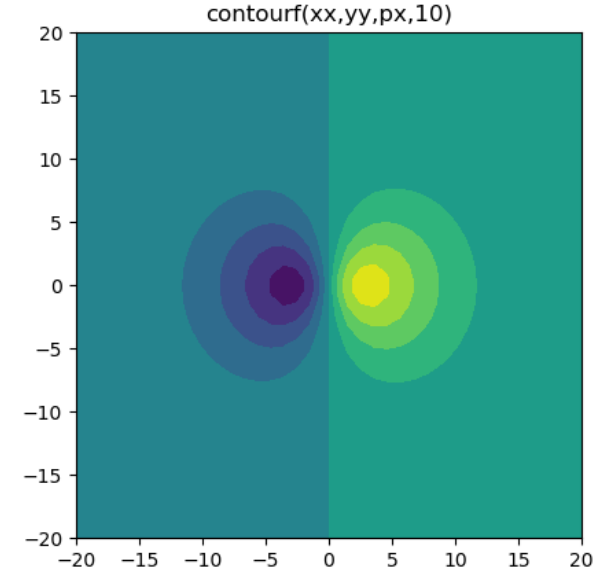


Specify the number of levels



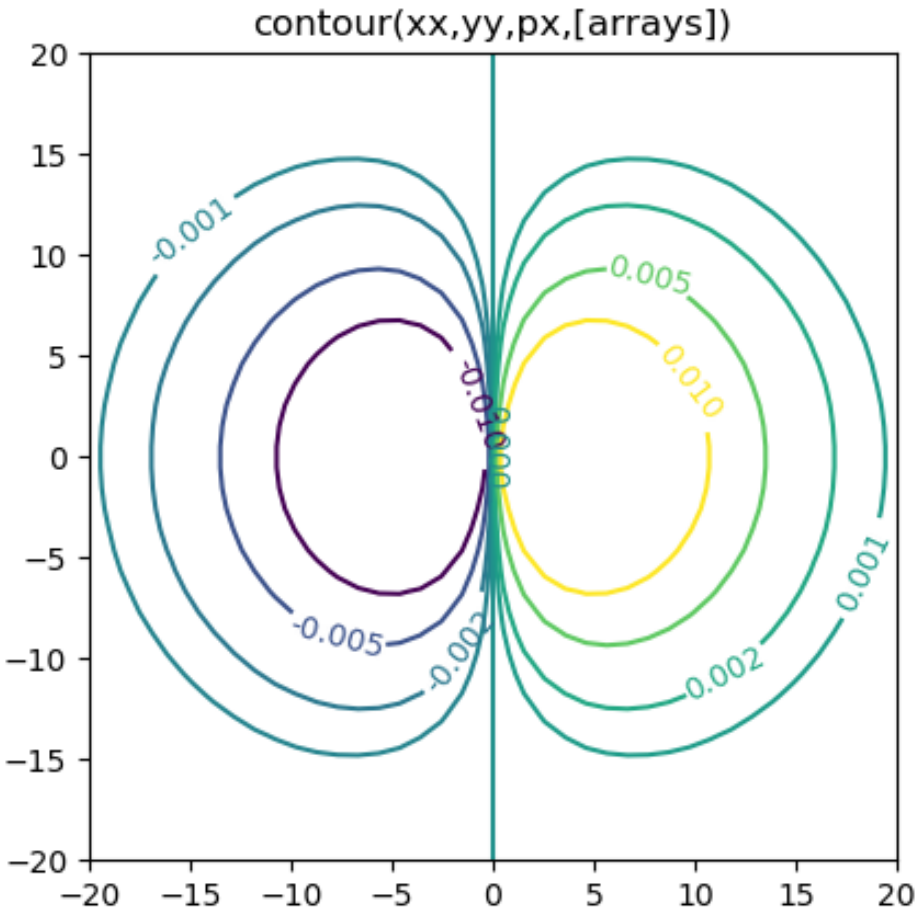






Specify the values of different levels

CS=plt.contour(xx,yy,px,[-0.01, -0.005, -0.002, -0.001, 0, 0.001, 0.002, 0.005, 0.01])
plt.clabel(CS,CS.levels, inline=False, fontsize=10))



Practice 1: 4d and 4f orbital

https://winter.group.shef.ac.uk/orbitron/AOs/4f/equations.html

The symbols used in the following are:

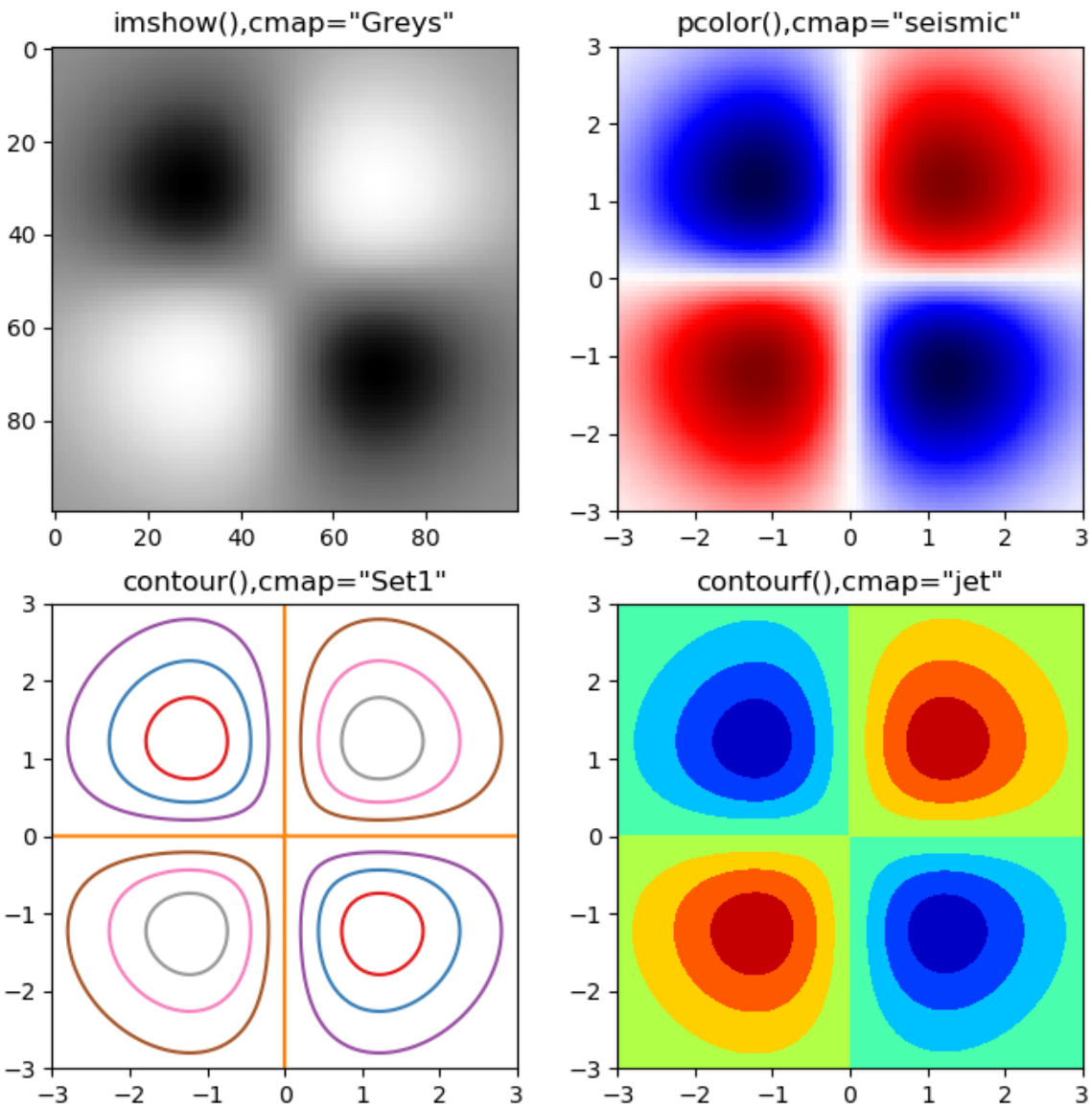
- r = radius expressed in atomic units (1 Bohr radius = 52.9 pm)
- π = 3.14159 approximately
- e = 2.71828 approximately
- Z = effective nuclear charge for that orbital in that atom.
- $\rho = 2Zr/n$ where n is the principal quantum number (4 for the 4d orbitals)

Table of equations for the 4*d* orbitals.

Function	Equation
Radial wave function, R_{4d}	$= (1/96\sqrt{5}) \times (6 - \rho)\rho^2 \times Z^{3/2} \times e^{-\rho/2}$
Angular wave functions:	
$Y_{4d_{xy}}$	$= \sqrt{(60/4)}xy/r^2 \times (1/4\pi)^{1/2}$
$Y_{4d_{xz}}$	$= \sqrt{(60/4)}xz/r^2 \times (1/4\pi)^{1/2}$
$Y_{4d_{yz}}$	$= \sqrt{(60/4)}yz/r^2 \times (1/4\pi)^{1/2}$
$Y_{4d_{x^2-y^2}}$	$= \sqrt{(15/4)}(x^2 - y^2)/r^2 \times (1/4\pi)^{1/2}$
$Y_{4d_{z^2}}$	$= \sqrt{(5/4)}\{2z^2-(x^2 + y^2)\}/r^2 \times (1/4\pi)^{1/2}$
Wave functions:	
$\psi_{4d_{xy}}$	$= R_{4d} \times Y_{4d_{xy}}$
$\psi_{4d_{xz}}$	$= R_{4d} \times Y_{4d_{xz}}$
$\psi_{4d_{yz}}$	$= R_{4d} \times Y_{4d_{yz}}$
$\psi_{4d_{x^2-y^2}}$	$= R_{4d} \times Y_{4d_{x^2-y^2}}$
$\psi_{4d_{z^2}}$	$= R_{4d} \times Y_{4d_{z^2}}$

Table of equations for the 4*f* orbitals.

Function	Equation
Radial wave function, R_{4f}	$= (1/96\sqrt{35}) \times \rho^3 \times Z^{3/2} \times e^{-\rho/2}$
Angular wave functions (cubic set):	
$Y_{4f_{x^3}}$	$= \sqrt{(7/4)} \times x(5x^2 - 3r^2)/r^3 \times (1/4\pi)^{1/2}$
$Y_{4f_{y^3}}$	$= \sqrt{(7/4)} \times y(5y^2 - 3r^2)/r^3 \times (1/4\pi)^{1/2}$
$Y_{4f_{z^3}}$	$= \sqrt{(7/4)} \times z(5z^2 - 3r^2)/r^3 \times (1/4\pi)^{1/2}$
$Y_{4f_{x(z^2-y^2)}}$	$= \sqrt{(105/4)} \times x(z^2-y^2)/r^3 \times (1/4\pi)^{1/2}$
$Y_{4f_{y(z^2-x^2)}}$	$= \sqrt{(105/4)} \times y(z^2-x^2)/r^3 \times (1/4\pi)^{1/2}$
$Y_{4f_{z(x^2-y^2)}}$	$= \sqrt{(105/4)} \times z(x^2-y^2)/r^3 \times (1/4\pi)^{1/2}$
$Y_{4f_{xyz}}$	$= \sqrt{(105/4)} \times xyz/r^3 \times (1/4\pi)^{1/2}$
Angular wave functions (general set):	
$Y_{4f_{z^3}}$	$= \sqrt{(7/4)} \times z(5z^2 - 3r^2)/r^3 \times (1/4\pi)^{1/2}$
$Y_{4f_{xz^2}}$	$= \sqrt{(42/16)} \times x(5z^2 - r^2)/r^3 \times (1/4\pi)^{1/2}$
$Y_{4f_{yz^2}}$	$= \sqrt{(42/16)} \times y(5z^2 - r^2)/r^3 \times (1/4\pi)^{1/2}$
$Y_{4f_{y(3x^2-y^2)}}$	$= \sqrt{(70/16)} \times y(3x^2-y^2)/r^3 \times (1/4\pi)^{1/2}$
$Y_{4f_{x(x^2-3y^2)}}$	$= \sqrt{(70/16)} \times x(x^2-3y^2)/r^3 \times (1/4\pi)^{1/2}$
$Y_{4f_{xyz}}$	$= \sqrt{(105/4)} \times xyz/r^3 \times (1/4\pi)^{1/2}$
$Y_{4f_{z(x^2-y^2)}}$	$= \sqrt{(105/4)} \times z(x^2-y^2)/r^3 \times (1/4\pi)^{1/2}$


imshow(),cmap="Greys"
pcolor(),cmap="seismic"
contour(),cmap="Set1"
contourf(),cmap="jet"

Using colormap-2

- Use the keyword cmap to change the default the colormap, e.g.

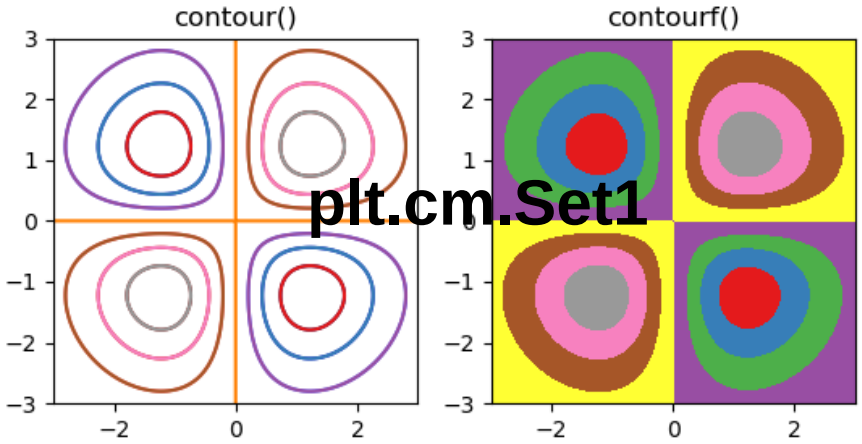
plt.contour(xx,yy,dxy,cmap=plt.cm.jet)
plt.contourf(xx,yy,dxy, cmap='jet')

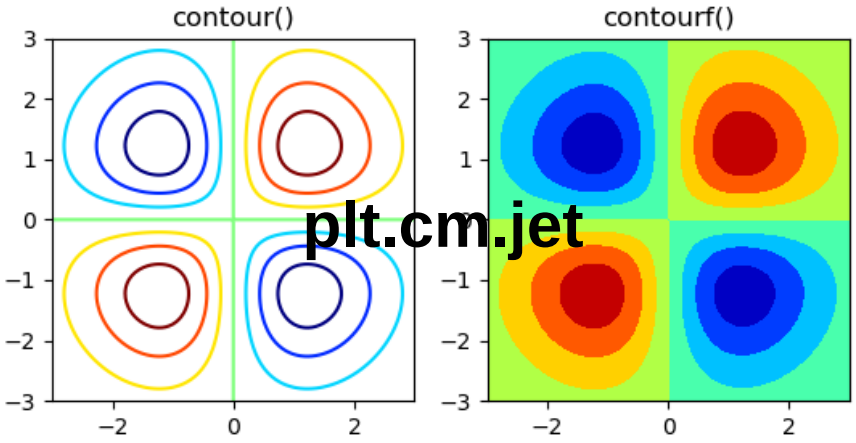
- Use the function in matplotlib at the beginning of the codes

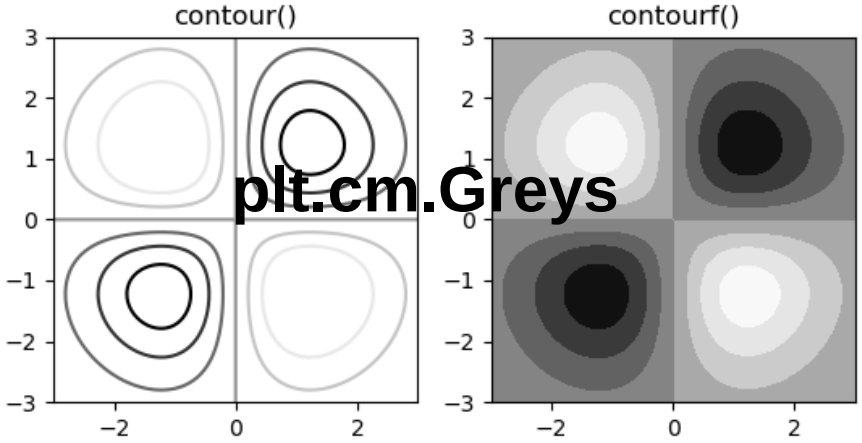
import matplotlib as mpl
mpl.rc('image', cmap='jet')

- Use the function in pyplot

pyplot.set_cmap('jet')



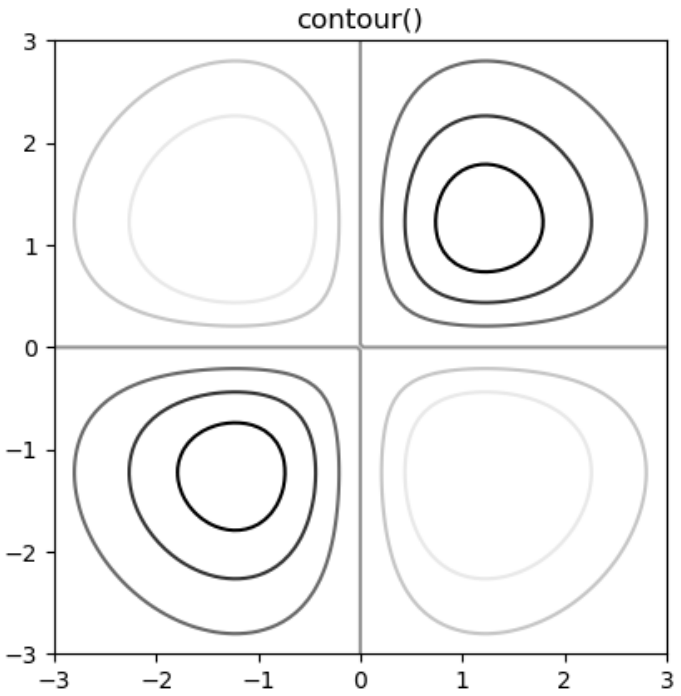


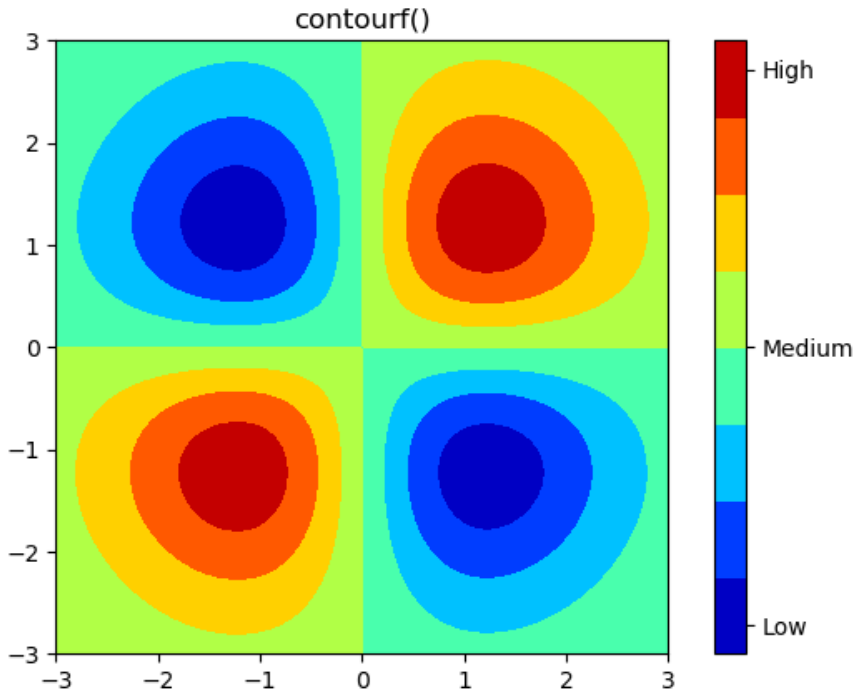


colorbar

- We can use the following commands to show the colorbar

plt.contourf(xx,yy,dxy)
plt.set_cmap('jet')
cbar = plt.colorbar(ticks=[dxy.min(), 0, dxy.max()], orientation='vertical')
cbar.ax.set_yticklabels(['Low', 'Medium', 'High'])





Build your own colormap

```
startcolor=(1,1,1);
midcolor=(1,1,1);
endcolor=(247/255, 220/255, 111/255);
my_cmap =
matplotlib.colors.LinearSegmentedColormap.from_list('my_cmap',[startcolor,midcolor,endcolor],2)
```