SCRIPTING AND PROGRAMMING LABORATORY FOR DATA ANALYSIS

Lecture 5 - Advanced plotting with matplotlib

Bar diagrams

A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically (**bar**) or horizontally (**barh**).

bar(x, height, width=0.8, bottom=None, align='center', **kwargs)

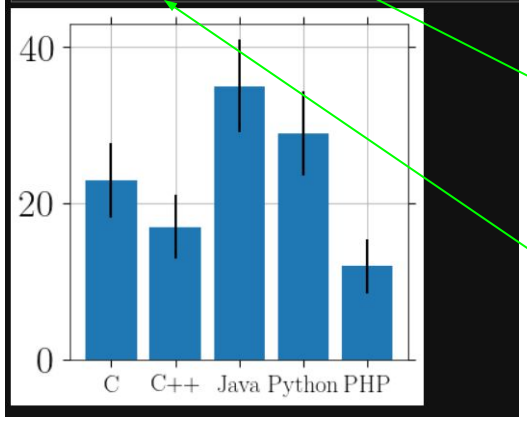
Coordinates, strings or numbers

Height of the bar

Height of the bar

If bars starts from non-zero y

```
fig, ax=plt.subplots(figsize=(5,5))
labs = ["C","C++","Java","Python","PHP"]
students = [23,17,35,29,12]
ax.bar(labs,students,yerr=np.sqrt(students))
ax.tick_params(labelsize=20,axis='x')
```



x-axis alignment

Error bars

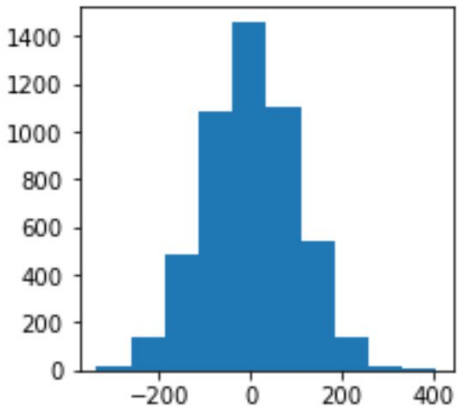
Tick_params is a function that controls many features of the axis

Histograms

To generate a 1D histogram we only need a single vector of numbers and to use the function

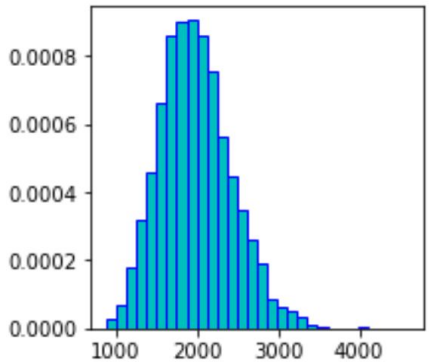
```
hist(x, bins=None, range=None, density=False, weights=None, cumulative=False,
bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None,
log=False, color=None, label=None, stacked=False, *, data=None, **kwargs)
```

```
fig,ax = plt.subplots(figsize=(3,3))
plt.hist(np.random.normal(scale=100,size=5000))
plt.show()
```



An array of data is the only compulsory argument

```
fig,ax = plt.subplots(figsize=(3,3))
val=np.random.gamma(20,scale=100,size=5000)
plt.hist(val, bins=30,density=True,
         color="c", edgecolor="b")
plt.show()
```

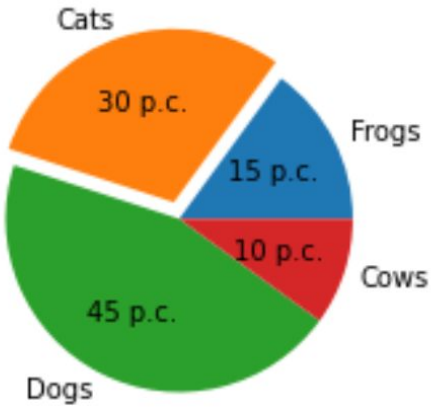


Pies

Matplotlib also allows you to create pie charts with the **pie()** function.

```
pie(x, explode=None, labels=None,
colors=None, autopct=None,
pctdistance=0.6, shadow=False,
labeldistance=1.1, startangle=0,
radius=1, counterclock=True,
wedgeprops=None, textprops=None,
center=(0, 0), frame=False,
rotatelabels=False, *,
normalize=True, data=None)
```

```
fig,ax = plt.subplots(figsize=(3,3))
labels = 'Frogs', 'Cats', 'Dogs', 'Cows'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)  # only "explode"
                          #the 2nd slice (i.e. 'Cats')
plt.pie(sizes, explode=explode, labels=labels,
        autopct='%.0f p.c.')
plt.show()
```



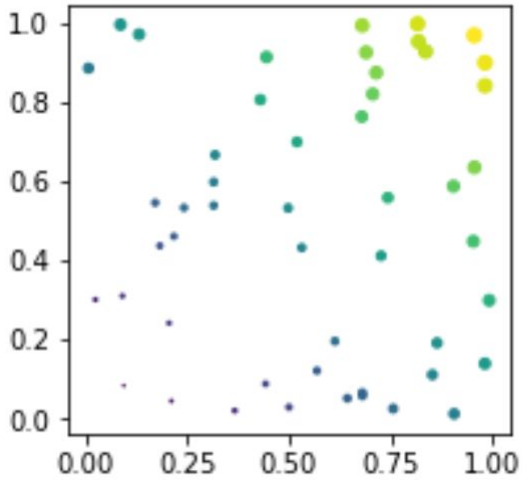
Scatter plots

With Pyplot, you can use the scatter() function to draw a scatter plot.

The scatter() function plots one dot for each “observation”. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis.

Particularly useful if you want each point to have a different feature (color, size…)

```
fig,ax = plt.subplots(figsize=(3,3))
x = np.random.rand(50)
y = np.random.rand(50)
plt.scatter(x,y, c=x+y, s=15*x**2+15*y**2)
plt.show()
```



Scatter plots

You can use the scatter() function to draw a scatter plot (see here).

Compulsory arguments: the data

Size: float or array-like

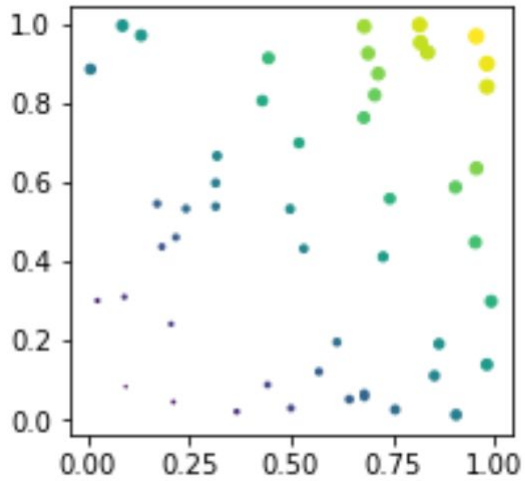
Size: array-like (floats) or list of colors or color

```
scatter(x, y, s=None, c=None,
marker=None, cmap=None,
norm=None, vmin=None,
vmax=None, alpha=None,
linewidths=None, *,
edgecolors=None,
plotnonfinite=False,
data=None, **kwargs)
```

Color map: default is viridis

Marker type

```
fig,ax = plt.subplots(figsize=(3,3))
x = np.random.rand(50)
y = np.random.rand(50)
plt.scatter(x,y, c=x+y, s=15*x**2+15*y**2)
plt.show()
```



Density and Contour plots

Sometimes it is useful to display three-dimensional data in two dimensions using contours or color-coded regions. Matplotlib has the following tools for this job:

- `plt.contour` -> plot contour levels
- `plt.contourf` -> plot filled contours
- `plt.imshow` -> display data (in 2D arrays) as an image, i.e., on a 2D regular raster.
- `plt.hist2d` -> make a 2D histogram plot where each pixel is color-coded

Density and Contour plots

- plt.contour (doc)-> plot contour levels
- plt.contourf (doc)-> plot filled contours

Optional

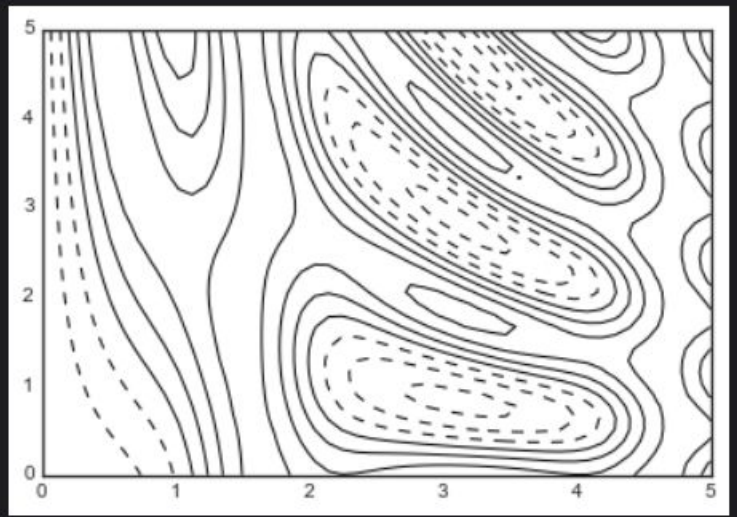
contour([X, Y,] Z, [levels], **kwargs)

- **X** and **Y** must both be 2D with the same shape as **Z** (e.g. created via numpy.meshgrid), or they must both be 1-D such that len(**X**) == N is the number of columns in **Z** and len(**Y**) == M is the number of rows in **Z**. **X** and **Y** must both be ordered monotonically.
- **Z** (M, N) array-like, the height values over which the contour is drawn.
- **Levels**: Determines the number and positions of the contour lines / regions. If integer, automatically tries to set n levels between min and max. If array-like, draw contour lines at the specified levels. The values must be in increasing order.

Density and Contour plots

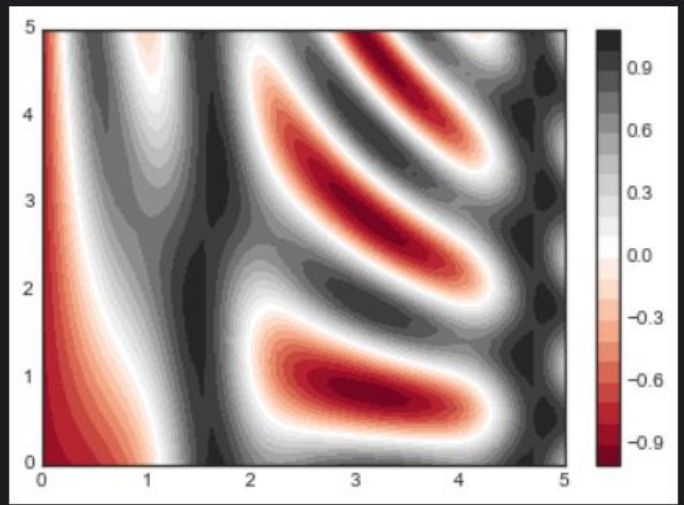
- `plt.contour` (doc)-> plot contour levels
- `plt.contourf` (doc)-> plot filled contours

```
plt.contour(X, Y, Z, colors='black');
```



Optional

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



Density and Contour plots

- plt.imshow (doc)-> display data (organised in 2D arrays) on a 2D regular grid.

With plt.contour the color steps are discrete rather than continuous, which is not always what is desired. plt.imshow interprets a two-dimensional grid of data as an image.

```
matplotlib.pyplot.imshow(Z, cmap=None, norm=None, *, aspect=None,
interpolation=None, alpha=None, vmin=None, vmax=None, origin=None,
extent=None, interpolation_stage=None, filternorm=True,
filterrad=4.0, resample=None, url=None, data=None, **kwargs)
```

The limits of the plot: array of floats (left, right, bottom, top)

Here only array with 2D data

normalisation

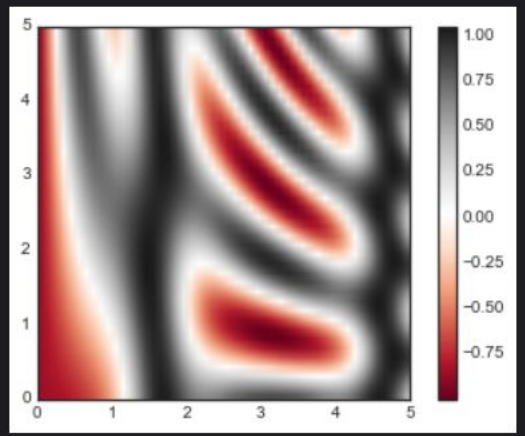
Produce smoother plots by using different kinds of interpolations (be careful)

Ordering of the data, i.e. the corner where X[0,0] is placed

Density and Contour plots

- plt.imshow (doc)-> display data (organised in 2D arrays) on a 2D regular grid.

```
plt.imshow(Z, extent=[0, 5, 0, 5], origin='lower',
           cmap='RdGy')
plt.colorbar()
plt.axis(aspect='image');
```



are discrete rather than
what is desired. plt.imshow
d of data as an image.

```
=None, norm=None, *, aspect=None,
vmin=None, vmax=None, origin=None,
=None, filternorm=True,
l=None, data=None, **kwargs)
```

imshow by default put the Z[0,0] at the top left corner!

Ordering of the data, i.e. the corner where Z[0,0] is placed

Density and Contour plots

- plt.hist2d -> make a 2D histogram plot where each pixel is color-coded

Scatter plot are very useful, but when point are densely concentrated you lose info about how many of them are located in a certain area. This info is recovered with a 2D hist, where you color code according to density.

range:
array-like shape(2, 2)
like[[xmin, xmax], [ymin, ymax]]

matplotlib.pyplot.hist2d(x, y, bins=10, range=None, density=False, weights=None, cmin=None, cmax=None, *, data=None, **kwargs)

Two 1D arrays as input!

Bins:
None or int or [int, int] or array-like or [array, array]

Density and Contour plots

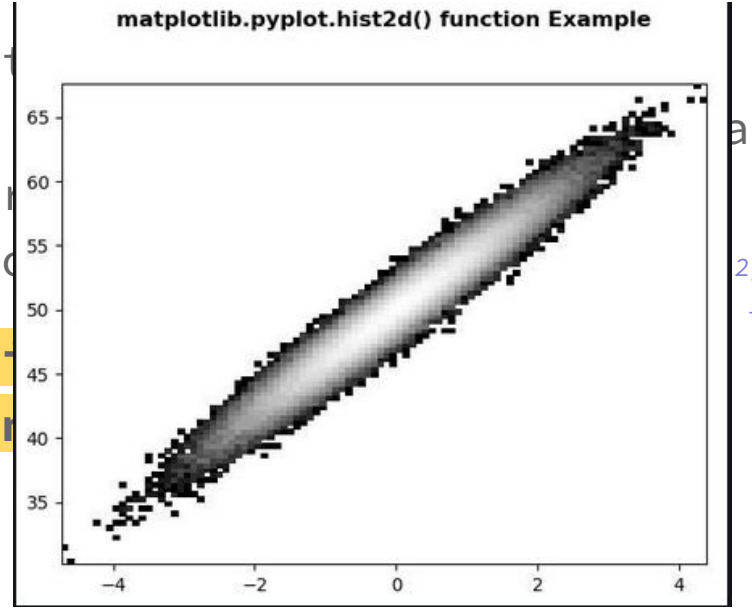
- `plt.hist2d` -> make a 2D histogram plot where each pixel is color-coded

```
N_points = 100000
x = np.random.randn(N_points)
y = 4 * x + np.random.randn(100000) + 50

plt.hist2d(x, y,
           bins = 100,
           norm = colors.LogNorm(),
           cmap ="gray")

plt.title('matplotlib.pyplot.hist2d() function \
Example\n\n', fontweight ="bold")

plt.show()
```



3D PLOTS

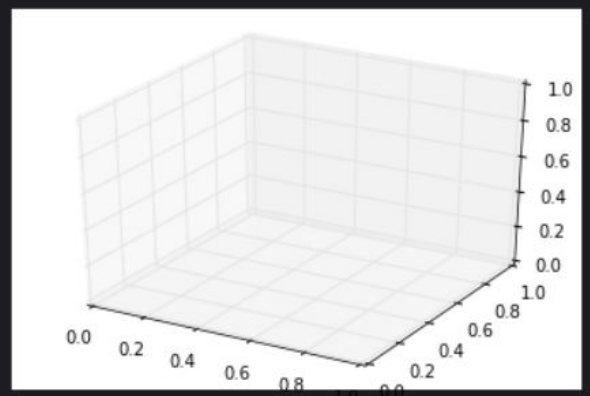
Three-dimensional plots are enabled by importing the **mplot3d** toolkit, included with the main Matplotlib installation. Three-dimensional axes can then be created by passing the keyword **projection='3d'** to any of the normal axes creation routines (see also here):

- **plt.plot(x,y,z)**
- **plt.scatter(x,y,z)**

```
from mpl_toolkits import mplot3d

fig3d = plt.figure(figsize=(8,8))
ax3d = plt.axes(projection='3d')
```

```
fig = plt.figure()
ax = plt.axes(projection='3d')
```



Animations

Matplotlib provides a framework to create animations by repeatedly calling a function *func*.

```
class matplotlib.animation.FuncAnimation(fig, func,
frames=None, init_func=None, fargs=None, save_count=None, *,
cache_frame_data=True, **kwargs)
```

Frame number: could be an integer or an iterable object

Function to initialise the plot

A figure to plot data

Function to be called for each frame

Animations

```
import matplotlib.animation as animation
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import HTML

# creating a blank window for the animation
fig = plt.figure(figsize=(8,8))
axis = plt.axes(xlim =(-50, 50), ylim =(-50, 50))
axis.set_aspect(1)

line, = axis.plot([], [], lw = 2)
```

Frame number: could be an integer or an iterable object

Function to initialise the pl

work to create animations by

```
# Initialization function: plot the background of each frame
def init():
    line.set_data([], [])
    return line,

# Animation function which updates figure data. This is called sequentially
def animate(i):
    # t is a parameter which varies with the frame number
    t = 0.1 * i

    # x, y values to be plotted at time t
    x = t * np.sin(t)
    y = t * np.cos(t)

    # appending values to the previously empty x and y data holders
    xdata.append(x)
    ydata.append(y)
    line.set_data(xdata, ydata)

    return line,
```

```
# initializing empty values for x and y co-ordinates
xdata, ydata = [], []

# Call the animator. blit=True means only re-draw the parts that have changed.
anim = animation.FuncAnimation(fig, animate, init_func = init, frames = 500, interval = 20, blit = True)
```