TECNICHE DI RAPPRESENTAZIONE E MODELLIZZAZIONE DEI DATI

— Part 1 —

(2 CFU out of 6 total CFU)

Link moodle: https://moodle2.units.it/course/view.php?id=11703

Teams code: 0ftoqj8

Python: numpy

Are array operations convenient?
Which speed up can I achieve?

```
import numpy as np
import time

#Define large matrices
large_matrix_1 = np.ones((1000, 1000))*5
large_matrix_2 = np.ones((1000, 1000))*2

#Empty matrix to be filled
result_large_matrix_1 = np.zeros((1000, 1000))

#Operate on them with for loop
t0 = time.time()
for i in range(1000):
    for j in range(1000):

            result_large_matrix_1[i, j] = large_matrix_1[i, j]**2 + np.log10(large_matrix_1[i, j]) +
                np.sqrt(large_matrix_2[i, j]) + large_matrix_2[i, j]**3

print('For loop takes {0} seconds'.format(time.time()-t0))

#Operate on them with array operations
t0 = time.time()
result_large_matrix_2 = large_matrix_1**2 + np.log10(large_matrix_1) + np.sqrt(large_matrix_2) + large_matrix_2**3

print('Array operation takes {0} seconds'.format(time.time()-t0))

print('The result is the same: {0}'.format(np.all(result_large_matrix_1 == result_large_matrix_2)))
```

Python: numpy

Are array operations convenient?
Which speed up can I achieve?

```
import numpy as np
import time

#Define large matrices
large_matrix_1 = np.ones((1000, 1000))*5
large_matrix_2 = np.ones((1000, 1000))*2

#Empty matrix to be filled
result_large_matrix_1 = np.zeros((1000, 1000))

#Operate on them with for loop
t0 = time.time()
for i in range(1000):
    for j in range(1000):

            result_large_matrix_1[i, j] = large_matrix_1[i, j]**2 + np.log10(large_matrix_1[i, j]) +
                np.sqrt(large_matrix_2[i, j]) + large_matrix_2[i, j]**3

print('For loop takes {0} seconds'.format(time.time()-t0))

#Operate on them with array operations
t0 = time.time()
result_large_matrix_2 = large_matrix_1**2 + np.log10(large_matrix_1) + np.sqrt(large_matrix_2) + large_matrix_2**3

print('Array operation takes {0} seconds'.format(time.time()-t0))

print('The result is the same: {0}'.format(np.all(result_large_matrix_1 == result_large_matrix_2)))
```

```
For loop takes 2.521970272064209 seconds
Array operation takes 0.018016815185546875 seconds
The result is the same: True
```

Python: numpy

Other relevant features of numpy

```
>>> import numpy as np
>>>
>>> a = np.array([2, 5, 6, 9, 12, 18, 21])
>>> min_a = np.min(a)
>>> max_a = np.max(a)
>>> mean_a = np.mean(a)
>>> std_a = np.std(a)
>>> print('Min = {}, max = {}, mean = {}, std = {}'.format(min_a, max_a, mean_a, std_a))
Min = 2, max = 21, mean = 10.428571428571429, std = 6.477590885002648
```

Python: numpy

Other relevant features of numpy

```
>>> import numpy as np
>>>
>>> a = np.array([2, 5, 6, 9, 12, 18, 21])
>>> min_a = np.min(a)
>>> max_a = np.max(a)
>>> mean_a = np.mean(a)
>>> std_a = np.std(a)
>>> print('Min = {}, max = {}, mean = {}, std = {}'.format(min_a, max_a, mean_a, std_a))
Min = 2, max = 21, mean = 10.428571428571429, std = 6.477590885002648
```

```
>>> b = np.array([7, 5, 3, 2, 6, 1, 4, 8])
>>> sorted_b = np.sort(b)
>>> print(sorted_b)
[1 2 3 4 5 6 7 8]
>>> print(sorted_b[::-1])
[8 7 6 5 4 3 2 1]
```

Python: numpy

Other relevant features of numpy

```
>>> import numpy as np
>>>
>>> a = np.array([2, 5, 6, 9, 12, 18, 21])
>>> min_a = np.min(a)
>>> max_a = np.max(a)
>>> mean_a = np.mean(a)
>>> std_a = np.std(a)
>>> print('Min = {}, max = {}, mean = {}, std = {}'.format(min_a, max_a, mean_a, std_a))
Min = 2, max = 21, mean = 10.428571428571429, std = 6.477590885002648
```

```
>>> b = np.array([7, 5, 3, 2, 6, 1, 4, 8])
>>> sorted_b = np.sort(b)
>>> print(sorted_b)
[1 2 3 4 5 6 7 8]
>>> print(sorted_b[::-1])
[8 7 6 5 4 3 2 1]
```

```
>>> val = np.arange(0, 5, 0.3)
>>> print(val)
[0.  0.3 0.6 0.9 1.2 1.5 1.8 2.1 2.4 2.7 3.  3.3 3.6 3.9 4.2 4.5 4.8]
```

Python: libraries

A Python library is a collection of codes and modules.

It contains bunches of code that can be used repeatedly in different programs for specific operations.

We use libraries so that we don't need to write the code again in our program that is already available.

Python: libraries

matplotlib Plot types User guide Tutorials Examples Reference Contribute Releases

Latest stable release
3.8.0: docs | release notes

Last release for Python 2
2.2.5: docs | changelog

Matplotlib 3.8.0 documentation

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations.

Install

pip conda other

```
pip install matplotlib
```

matplotlib

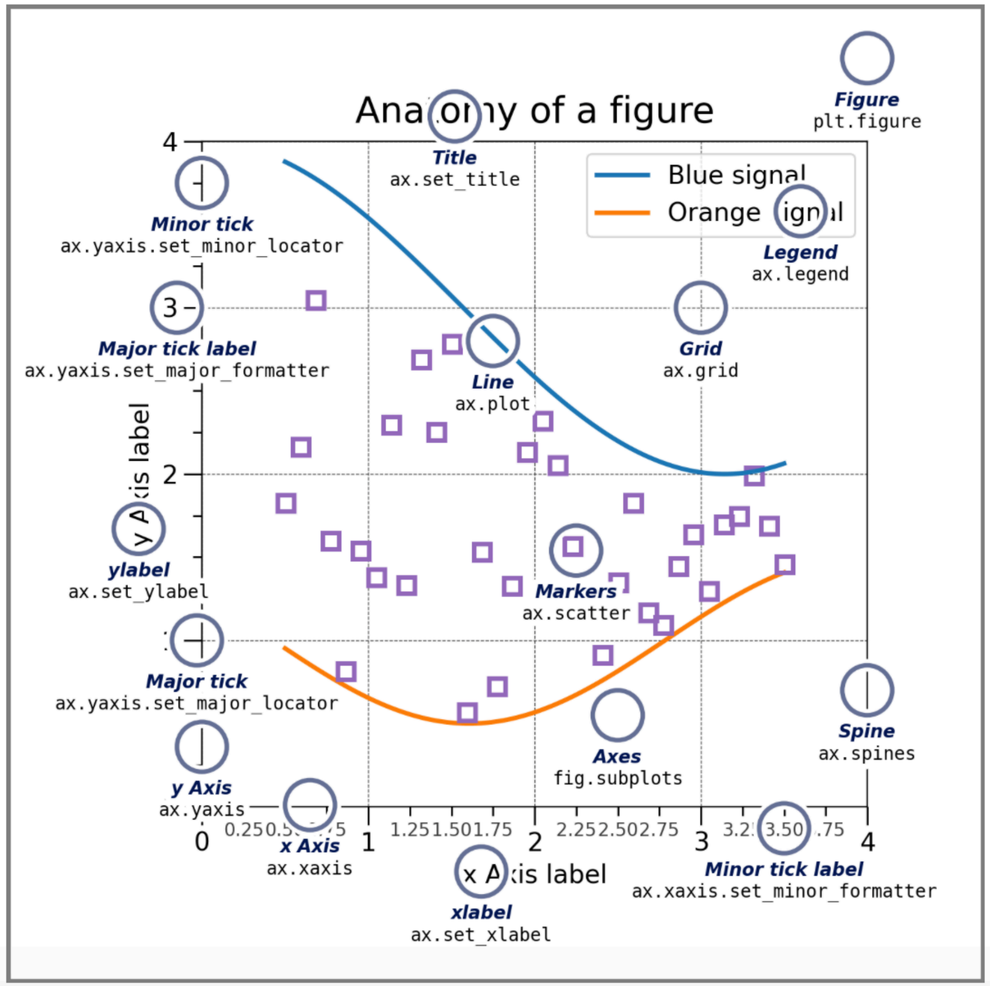
A Python library is a collection of codes and modules.

It contains bunches of code that can be used repeatedly in different programs for specific operations.

We use libraries so that we don't need to write the code again in our program that is already available.

Parts of a Figure

Here are the components of a Matplotlib Figure.



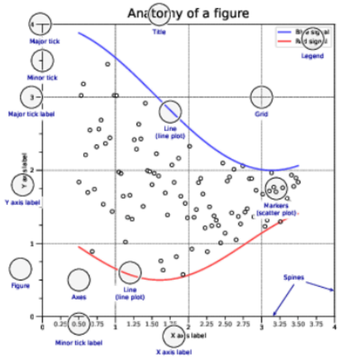
Python: libraries

matplotlib Cheat sheet Version 3.5.0

Quick start

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


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
plt.show()
```

Anatomy of a figure



Subplots layout

```
subplot[s](rows,cols,…)
fig, axs = plt.subplots(3, 3)

G = gridspec(rows,cols,…)
ax = G[0,:]

ax.inset_axes(extent)

d=make_axes_locatable(ax)
ax = d.new_horizontal('10%')
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- https://gitter.im/matplotlib/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

```
plot([X],Y,[fmt],…)
X, Y, fmt, color, marker, linestyle

scatter(X,Y,…)
X, Y, [s]izes, [c]olors, marker, cmap

bar[h](x,height,…)
x, height, width, bottom, align, color

imshow(Z,…)
Z, cmap, interpolation, extent, origin

contour[f]([X],[Y],Z,…)
X, Y, Z, levels, colors, extent, origin

pcolormesh([X],[Y],Z,…)
X, Y, Z, vmin, vmax, cmap

quiver([X],[Y],U,V,…)
X, Y, U, V, C, units, angles

pie(X,…)
Z, explode, labels, colors, radius

text(x,y,text,…)
x, y, text, va, ha, size, weight, transform

fill[_between][x](…)
X, Y1, Y2, color, where
```

Advanced plots

```
step(X,Y,[fmt],…)
X, Y, fmt, color, marker, where

boxplot(X,…)
X, notch, sym, bootstrap, widths

errorbar(X,Y,xerr,yerr,…)
X, Y, xerr, yerr, fmt

hist(X, bins, …)
X, bins, range, density, weights

violinplot(D,…)
D, positions, widths, vert

barbs([X],[Y], U, V, …)
X, Y, U, V, C, length, pivot, sizes

eventplot(positions,…)
positions, orientation, lineoffsets

hexbin(X,Y,C,…)
X, Y, C, gridsize, bins
```

Scales

```
ax.set_[xy]scale(scale,…)
linear  any values
log     values > 0
symlog  any values
logit   0 < values < 1
```

Projections

```
subplot(…,projection=p)
p='polar'
p='3d'
p=ccrs.Orthographic()
import cartopy.crs as ccrs
```

Lines

```
linestyle or ls
"-" ":" "-." "--" (0,(0.01,2))
capstyle or dash_capstyle
"butt" "round" "projecting"
```

Markers

```
markevery
10 [0, -1] (25, 5) [0, 25, -1]
```

Colors

```
'Cn'
'x'
'name'
(R,G,B[,A])
'#RRGGBB[AA]'
'x.y'
```

Colormaps

```
plt.get_cmap(name)
```

Uniform: viridis, magma, plasma

Sequential: Greys, YlOrBr, Wistia

Diverging: Spectral, coolwarm, RdGy

Qualitative: tab10, tab20

Cyclic: twilight

Tick locators

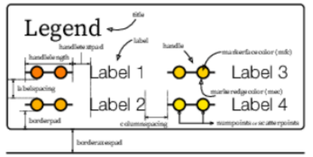
```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
ticker.NullLocator()
ticker.MultipleLocator(0.5)
ticker.FixedLocator([0, 1, 5])
ticker.LinearLocator(numticks=3)
ticker.IndexLocator(base=0.5, offset=0.25)
ticker.AutoLocator()
ticker.MaxNLocator(n=4)
ticker.LogLocator(base=10, numticks=15)
```

Tick formatters

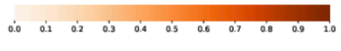
```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
ticker.NullFormatter()
ticker.FixedFormatter(['zero', 'one', 'two', …])
ticker.FuncFormatter(lambda x, pos: "[%.2f]" % x)
ticker.FormatStrFormatter('>%d<')
ticker.ScalarFormatter()
ticker.StrMethodFormatter('{x}')
ticker.PercentFormatter(xmax=5)
```

Ornaments

```
ax.legend(…)
handles, labels, loc, title, frameon
```




```
ax.colorbar(…)
mappable, ax, cax, orientation
```



```
ax.annotate(…)
text, xy, xytext, xycoords, textcoords, arrowprops
```

Event handling

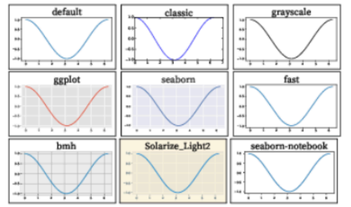
```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

```
plt.style.use(style)
```



Quick reminder

```
ax.grid()
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(ticks, [labels])
ax.set_[xy]ticklabels(labels)
ax.set_title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

fig.suptitle(title)
fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
[fig|ax].patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

Keyboard shortcuts

ctrl + s Save	ctrl + w Close plot
r Reset view	f Fullscreen 0/1
f View forward	b View back
p Pan view	o Zoom to rect
x X pan/zoom	y Y pan/zoom
g Minor grid 0/1	G Major grid 0/1
l X axis log/linear	L Y axis log/linear

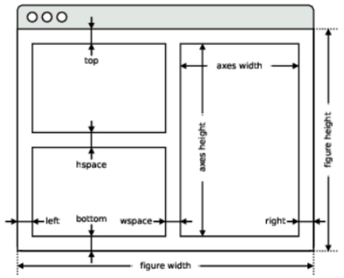
Ten simple rules

1. Know your audience
2. Identify your message
3. Adapt the figure
4. Captions are not optional
5. Do not trust the defaults
6. Use color effectively
7. Do not mislead the reader
8. Avoid "chartjunk"
9. Message trumps beauty
10. Get the right tool

Python: libraries

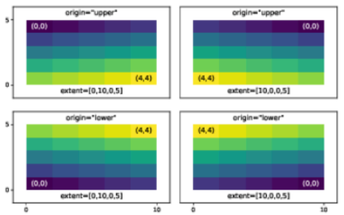
Axes adjustments

plt.subplots_adjust(...)



Extent & origin

ax.imshow(extent=..., origin=...)



Text alignments

ax.text(..., ha=... , va=..., ...)

Text parameters

ax.text(..., family=..., size=..., weight=...)
ax.text(..., fontproperties=...)

The quick brown fox — xx-large (1.73)
The quick brown fox — x-large (1.44)
The quick brown fox — large (1.20)
The quick brown fox — medium (1.00)
The quick brown fox — small (0.83)
The quick brown fox — x-small (0.69)
The quick brown fox — xx-small (0.58)

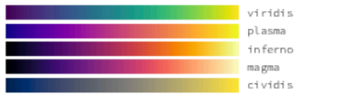
The quick brown fox jumps over the lazy dog — black (900)
The quick brown fox jumps over the lazy dog — bold (700)
The quick brown fox jumps over the lazy dog — semibold (600)
The quick brown fox jumps over the lazy dog — normal (400)
The quick brown fox jumps over the lazy dog — ultralight (100)

The quick brown fox jumps over the lazy dog — monospace
The quick brown fox jumps over the lazy dog — serif
The quick brown fox jumps over the lazy dog — sans
The quick brown fox jumps over the lazy dog — cursive

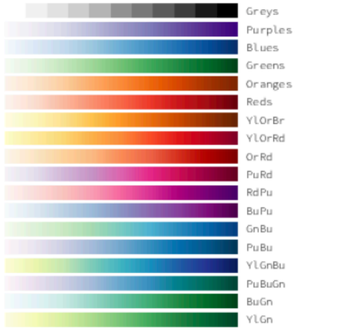
The quick brown fox jumps over the lazy dog — italic
The quick brown fox jumps over the lazy dog — normal

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG — small-caps
The quick brown fox jumps over the lazy dog — normal

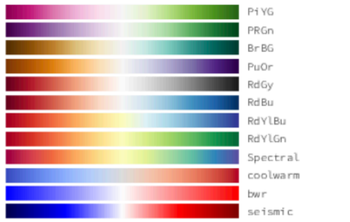
Uniform colormaps



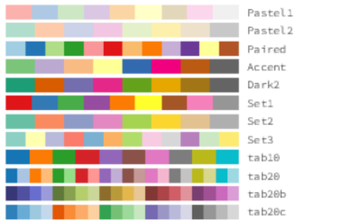
Sequential colormaps



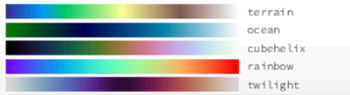
Diverging colormaps



Qualitative colormaps

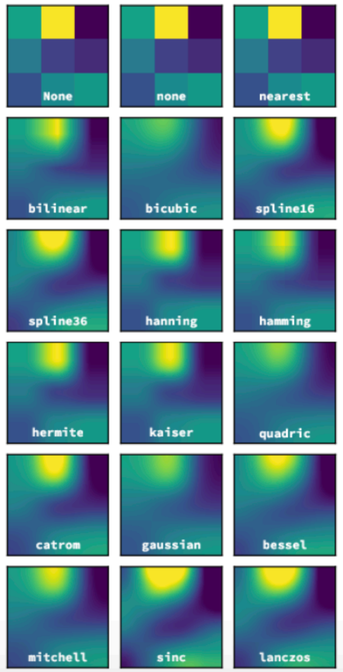


Miscellaneous colormaps

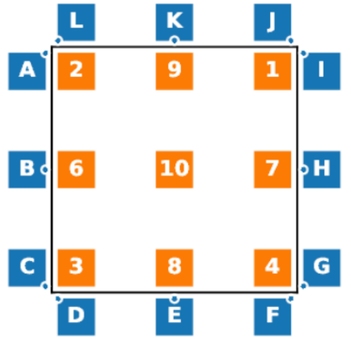


Color names

Image interpolation



Legend placement

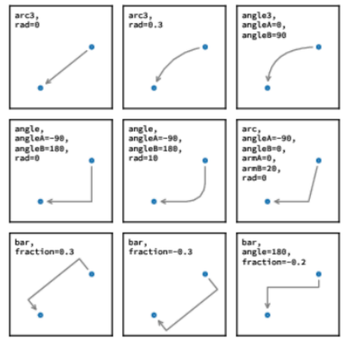


ax.legend(loc="string", bbox_to_anchor=(x,y))

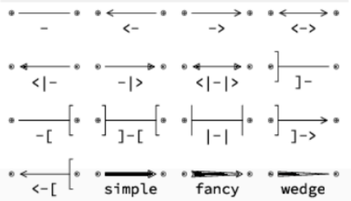
2: upper left — 9: upper center — 1: upper right
6: center left — 10: center — 7: center right
3: lower left — 8: lower center — 4: lower right

A: upper right / (-0.1,0.9) — B: center right / (-0.1,0.5)
C: lower right / (-0.1,0.1) — D: upper left / (0.1, -0.1)
E: upper center / (0.5,-0.1) — F: upper right / (0.9,-0.1)
G: lower left / (1.1,0.1) — H: center left / (1.1,0.5)
I: upper left / (1.1,0.9) — J: lower right / (0.9,1.1)
K: lower center / (0.5,1.1) — L: lower left / (0.1,1.1)

Annotation connection styles



Annotation arrow styles



How do I ...

... resize a figure?
→ fig.set_size_inches(w, h)
... save a figure?
→ fig.savefig("figure.pdf")
... save a transparent figure?
→ fig.savefig("figure.pdf", transparent=True)
... clear a figure/an axes?
→ fig.clear() → ax.clear()
... close all figures?
→ plt.close("all")
... remove ticks?
→ ax.set_[xy]ticks([])
... remove tick labels ?
→ ax.set_[xy]ticklabels([])
... rotate tick labels ?
→ ax.tick_params(axis="x", rotation=90)
... hide top spine?
→ ax.spines['top'].set_visible(False)
... hide legend border?
→ ax.legend(frameon=False)
... show error as shaded region?
→ ax.fill_between(X, Y+error, Y-error)
... draw a rectangle?
→ ax.add_patch(plt.Rectangle((0, 0), 1, 1)
... draw a vertical line?
→ ax.axvline(x=0.5)
... draw outside frame?
→ ax.plot(..., clip_on=False)
... use transparency?
→ ax.plot(..., alpha=0.25)
... convert an RGB image into a gray image?
→ gray = 0.2989*R + 0.5870*G + 0.1140*B
... set figure background color?
→ fig.patch.set_facecolor("grey")
... get a reversed colormap?
→ plt.get_cmap("viridis_r")
... get a discrete colormap?
→ plt.get_cmap("viridis", 10)
... show a figure for one second?
→ fig.show(block=False), time.sleep(1)

Performance tips

```
scatter(X, Y)                          slow
plot(X, Y, marker="o", ls="")          fast

for i in range(n): plot(X[i])          slow
plot(sum([x+[None] for x in X],[]))    fast

cla(), imshow(…), canvas.draw()        slow
im.set_data(…), canvas.draw()          fast
```

Beyond Matplotlib

Seaborn: Statistical data visualization
Cartopy: Geospatial data processing
yt: Volumetric data visualization
mpld3: Bringing Matplotlib to the browser
Datashader: Large data processing pipeline
plotnine: A grammar of graphics for Python

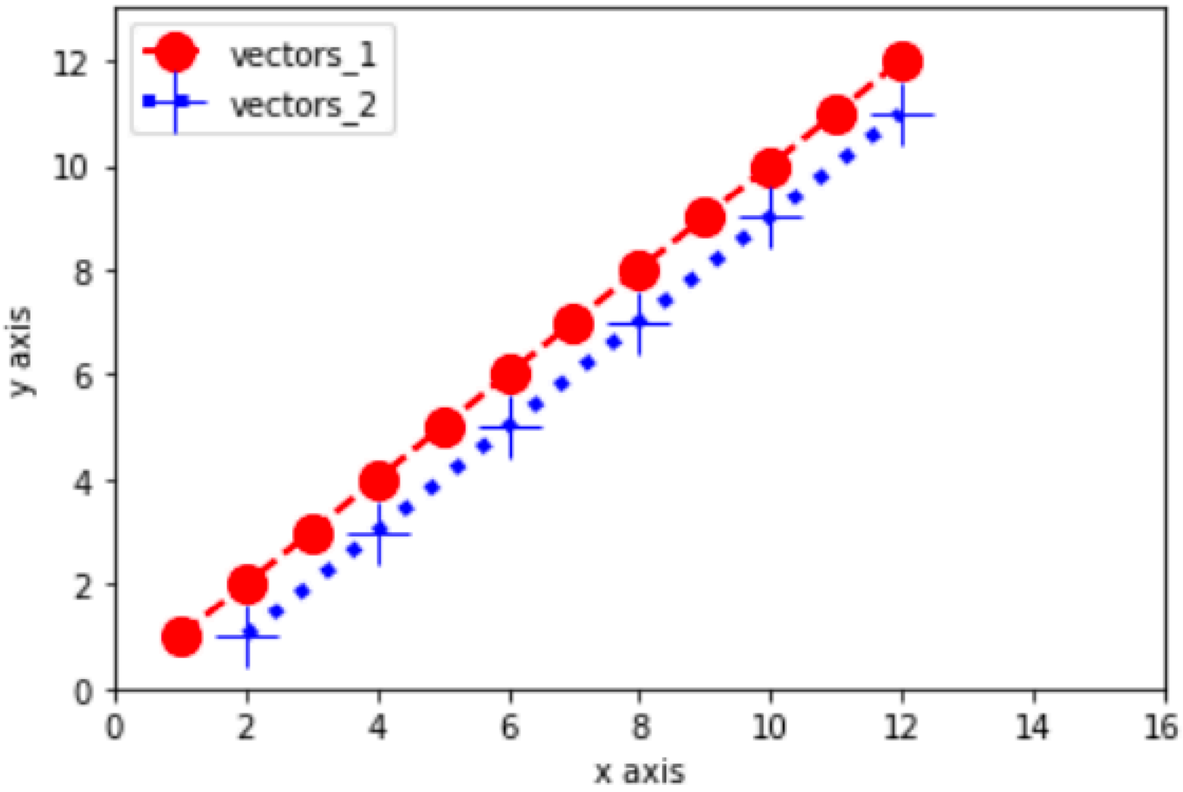
NUMFOCUS
OPEN CODE = BETTER SCIENCE

Python: plotting examples

```
6   @author: milenavalentini
7   """
8
9   import numpy as np
10  import matplotlib.pyplot as mpl
11
12  x_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
13  y_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
14
15  x_vector_2 = np.array([2,4,6,8,10,12])
16  y_vector_2 = np.array([1,3,5,7,9,11])
17
18  mpl.plot(x_vector_1, y_vector_1, color='red', marker='o', linestyle='dashed',
19      linewidth=2, markersize=12, label='vectors_1')
20
21  mpl.plot(x_vector_2, y_vector_2, color='blue', marker='+', linestyle='dotted',
22      linewidth=4, markersize=20, label='vectors_2')
23
24  mpl.xlim(0,16)
25  mpl.ylim(0,13)
26
27  mpl.xlabel('x axis')
28  mpl.ylabel('y axis')
29
30  mpl.legend()
31
32  mpl.show()
33  mpl.savefig('a_first_plot.png')
```

Python: plotting examples

```
6   @author: milenavalentini
7   """
8
9   import numpy as np
10  import matplotlib.pyplot as mpl
11
12  x_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
13  y_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
14
15  x_vector_2 = np.array([2,4,6,8,10,12])
16  y_vector_2 = np.array([1,3,5,7,9,11])
17
18  mpl.plot(x_vector_1, y_vector_1, color='red', marker='o', linestyle='dashed',
19       linewidth=2, markersize=12, label='vectors_1')
20
21  mpl.plot(x_vector_2, y_vector_2, color='blue', marker='+', linestyle='dotted',
22       linewidth=4, markersize=20, label='vectors_2')
23
24  mpl.xlim(0,16)
25  mpl.ylim(0,13)
26
27  mpl.xlabel('x axis')
28  mpl.ylabel('y axis')
29
30  mpl.legend()
31
32  mpl.show()
33  mpl.savefig('a_first_plot.png')
```

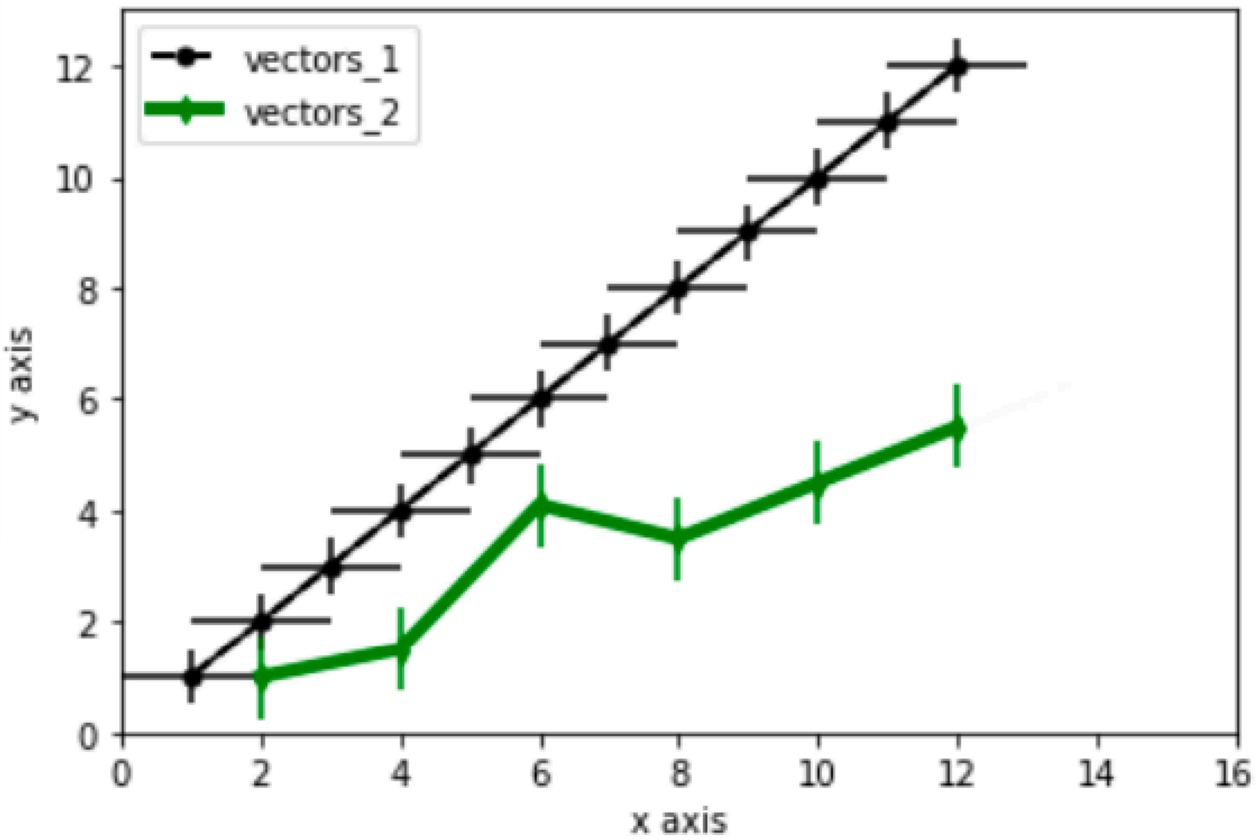


Python: plotting examples

```
6     @author: milenavalentini
7     """
8
9     import numpy as np
10    import matplotlib.pyplot as mpl
11
12    x_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
13    y_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
14
15    x_vector_2 = np.array([2,4,6,8,10,12])
16    y_vector_2 = np.array([1,1.5,4.1,3.5,4.5,5.5])
17
18    mpl.plot(x_vector_1, y_vector_1, color='black', marker='o', linestyle='dashed',
19        linewidth=2, markersize=5, label='vectors_1')
20
21    mpl.plot(x_vector_2, y_vector_2, color='green', marker='d', linestyle='solid',
22        linewidth=4, markersize=5, label='vectors_2')
23
24    mpl.errorbar(x_vector_1, y_vector_1, yerr=0.5, xerr=1, color='black')
25
26    mpl.errorbar(x_vector_2, y_vector_2, yerr=.75, color='green')
27
28    mpl.xlim(0,16)
29    mpl.ylim(0,13)
30
31    mpl.xlabel('x axis')
32    mpl.ylabel('y axis')
33
34    mpl.legend()
35
36    mpl.show()
37    mpl.savefig('a_first_plot.png')
```

Python: plotting examples

```
6     @author: milenavalentini
7     """
8
9     import numpy as np
10    import matplotlib.pyplot as mpl
11
12    x_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
13    y_vector_1 = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
14
15    x_vector_2 = np.array([2,4,6,8,10,12])
16    y_vector_2 = np.array([1,1.5,4.1,3.5,4.5,5.5])
17
18    mpl.plot(x_vector_1, y_vector_1, color='black', marker='o', linestyle='dashed',
19        linewidth=2, markersize=5, label='vectors_1')
20
21    mpl.plot(x_vector_2, y_vector_2, color='green', marker='d', linestyle='solid',
22        linewidth=4, markersize=5, label='vectors_2')
23
24    mpl.errorbar(x_vector_1, y_vector_1, yerr=0.5, xerr=1, color='black')
25
26    mpl.errorbar(x_vector_2, y_vector_2, yerr=.75, color='green')
27
28    mpl.xlim(0,16)
29    mpl.ylim(0,13)
30
31    mpl.xlabel('x axis')
32    mpl.ylabel('y axis')
33
34    mpl.legend()
35
36    mpl.show()
37    mpl.savefig('a_first_plot.png')
```

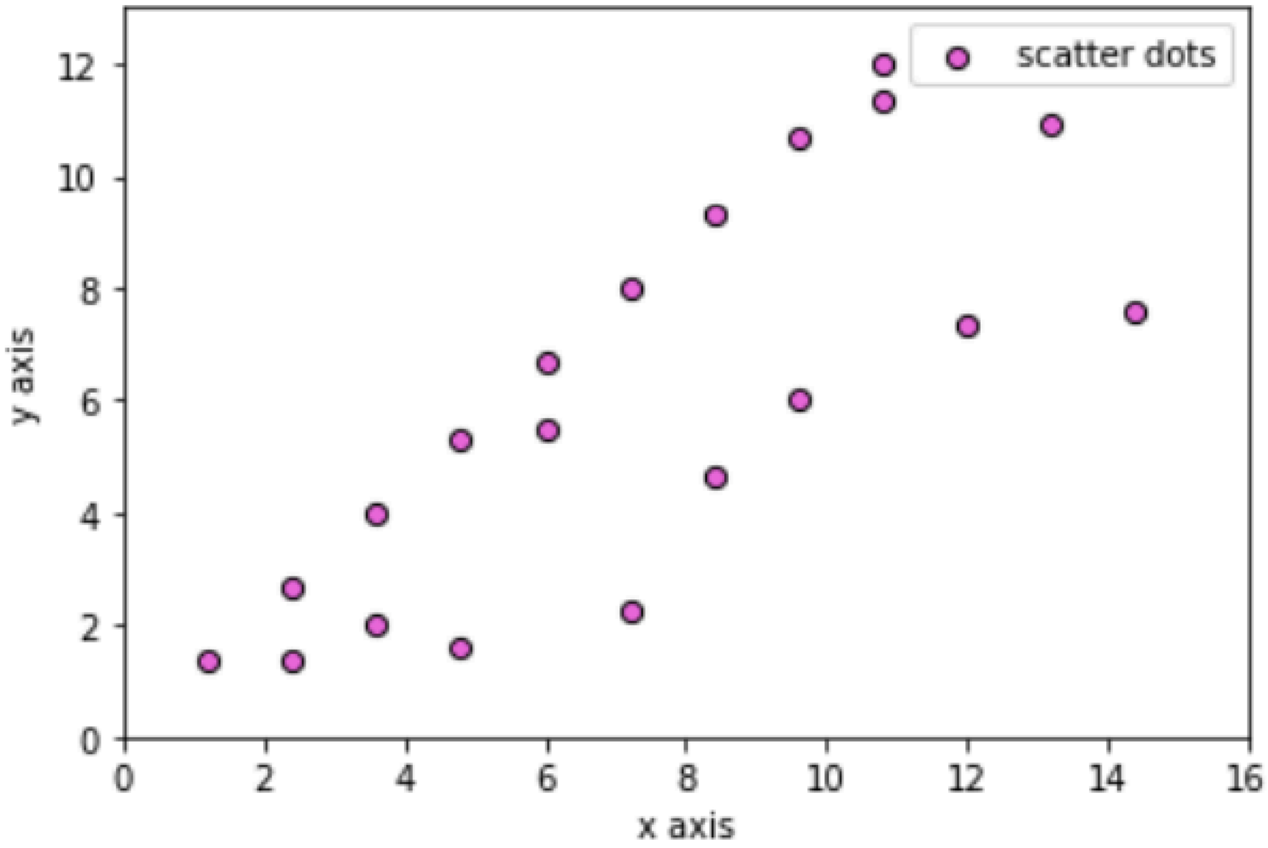


Python: plotting examples

```
6     @author: milenavalentini
7     """
8
9     import numpy as np
10    import matplotlib.pyplot as mpl
11
12    list_1 = [1,2,3,4,5,6,7,8,9,10,11,12]
13    list_2 = [1,2,3,4,5,6,7,8,9,10,11,12]
14
15    list_3 = [2,3,4,5,6,7,8,9,10,11,12]
16    list_4 = [1,1.5,1.2,4.1,1.7,3.5,4.5,8.5,5.5,8.2,5.7]
17
18    x_vector_1 = np.array(list_1 + list_3) * 1.2
19    y_vector_1 = np.array(list_2 + list_4) / 0.75
20
21    mpl.scatter(x_vector_1, y_vector_1, c='orchid', edgecolors='black', marker='.',
22    s=150, label='scatter dots')
23
24    mpl.xlim(0,16)
25    mpl.ylim(0,13)
26
27    mpl.xlabel('x axis')
28    mpl.ylabel('y axis')
29
30    mpl.legend()
31
32    mpl.show()
33    mpl.savefig('a_scatter_plot.png')
```

Python: plotting examples

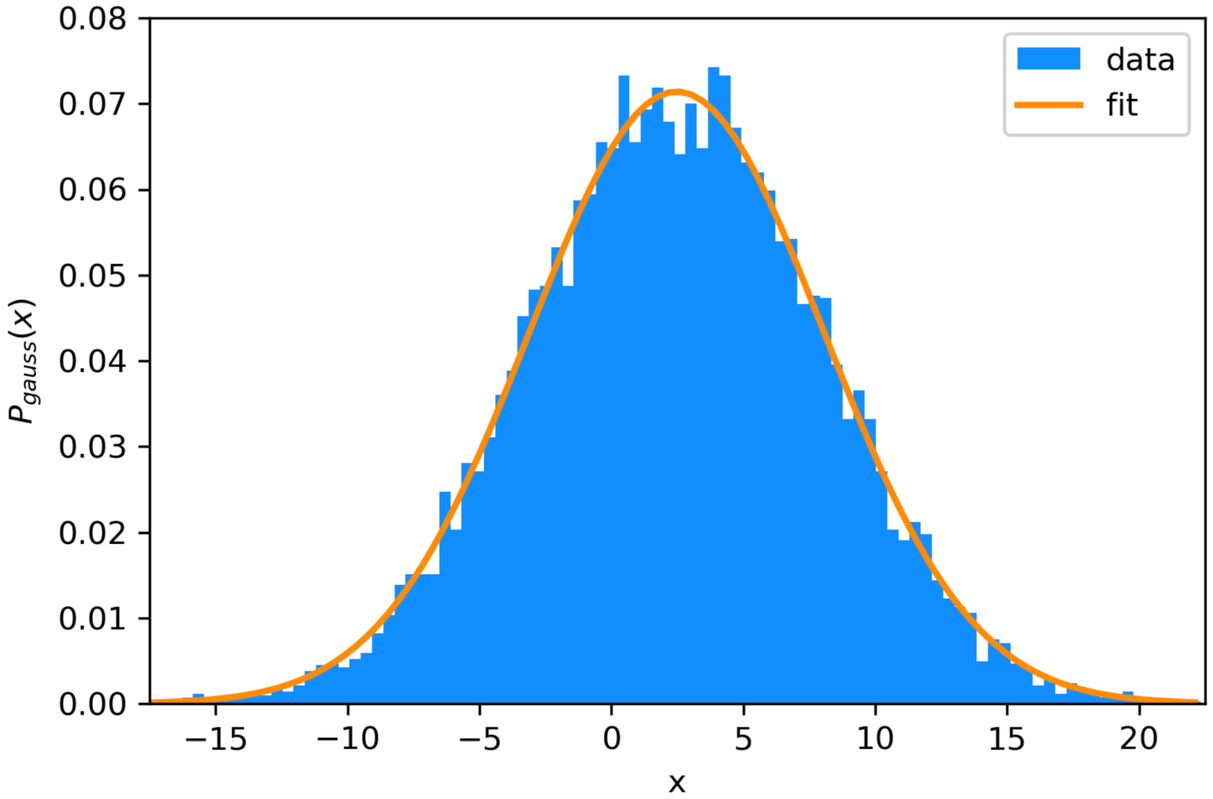
```python
@author: milenavalentini
"""

import numpy as np
import matplotlib.pyplot as mpl

list_1 = [1,2,3,4,5,6,7,8,9,10,11,12]
list_2 = [1,2,3,4,5,6,7,8,9,10,11,12]

list_3 = [2,3,4,5,6,7,8,9,10,11,12]
list_4 = [1,1.5,1.2,4.1,1.7,3.5,4.5,8.5,5.5,8.2,5.7]

x_vector_1 = np.array(list_1 + list_3) * 1.2
y_vector_1 = np.array(list_2 + list_4) / 0.75

mpl.scatter(x_vector_1, y_vector_1, c='orchid', edgecolors='black', marker='.',
s=150, label='scatter dots')

mpl.xlim(0,16)
mpl.ylim(0,13)

mpl.xlabel('x axis')
mpl.ylabel('y axis')

mpl.legend()

mpl.show()
mpl.savefig('a_scatter_plot.png')
```

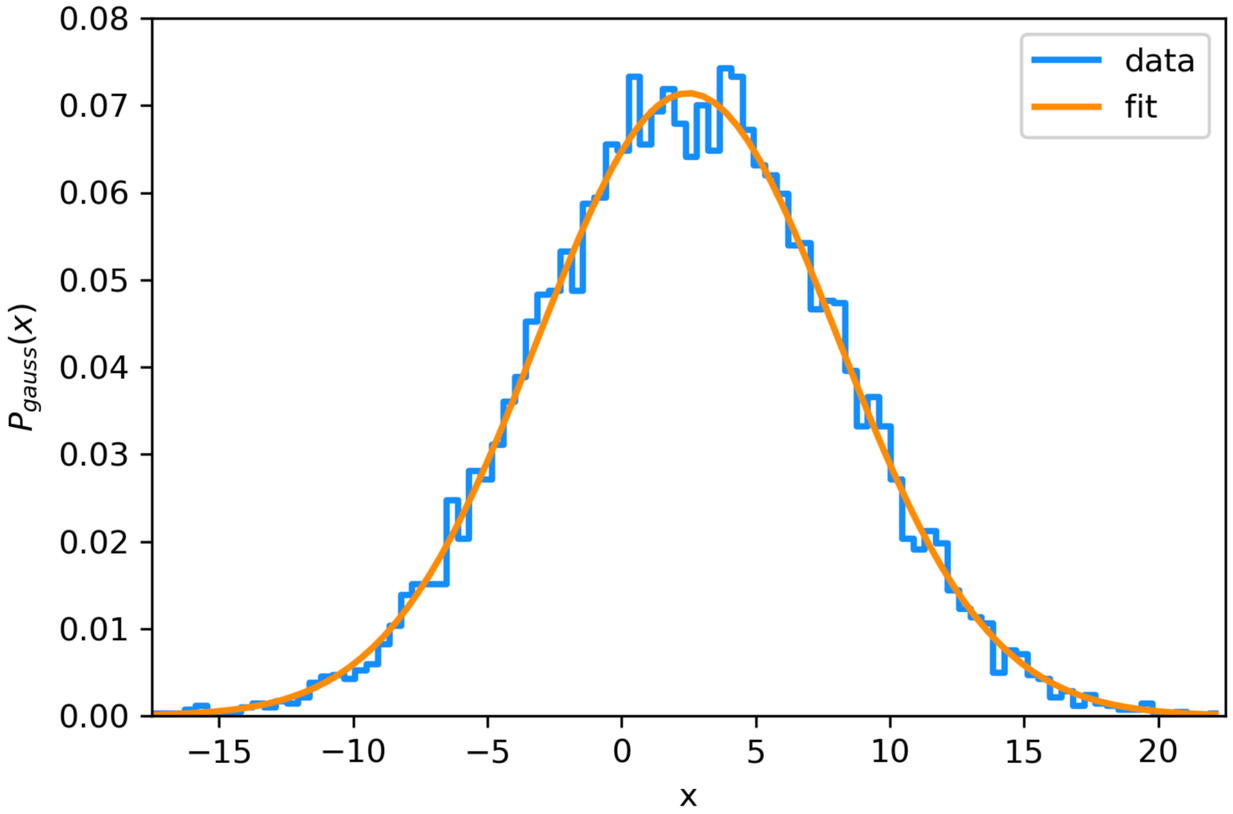


Python: plotting examples

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as sopt

#We generate random data, drawn from a gaussian centered at 2.5, standard deviation of 5.5
original_mu = 2.5
original_sigma = 5.5
my_data = np.random.normal(original_mu, original_sigma, 10000)

#We create an histogram of these data
#We do not use weights, but we ask for the probability density function to be returned. The integral of this
curve will be 1.
#We use 100 bins, we could have defined bins ourselves
gauss_hist, bin_edges = np.histogram(my_data, bins = 100, density = True)

…

#Fit curve to data using scipy.optimize
#We need to define a function to use for the fit. In our case, we fit a gaussian
#I place the function here for example, but this should really go at the top of the script
#This function returns the probability density of a Gaussian curve, i.e. consistent with our 'observed'
values (our histogram)
def gauss_for_fit(xvals, mu, sigma):

    p_of_x = (1./np.sqrt(2*np.pi*sigma**2))*np.exp(-(((xvals-mu)**2)/(2*sigma**2)))

    return p_of_x

#We perform the fit: scipy.optimize is only one of the many methods to do this!
fit_params, fit_covariance_matrix = sopt.curve_fit(gauss_for_fit, bin_centers, gauss_hist)
bestfit_mu, bestfit_sigma = fit_params

…
```

Python: plotting examples





Python: input/output

How to read from a file

```
#Inizio dal file contenente le proprieta' delle parent particles
filename = ‘my_file.dat'
# load data from the file my_file.dat, whose header is as follows:
# M(M_Sun) rho(gr/cm3) T(K) …
mass = []
density = []
temperature = []
…

with open(filename, 'r') as ppf:
    header = ppf.readline() # skip the first line of the file (header)
    for line in ppf:
        line = line.strip()
        columns = line.split()

        mass.append(float(columns[0]))
        density.append(float(columns[1]))
        temperature.append(float(columns[2]))
        …
#Transform lists into arrays
mass_array = np.array(mass)
density_array = np.array(density)
…
```

Python: input/output

How to read from a file

with numpy.loadtxt

```
206 import numpy as np
207
208 data_filename = 'path/filename'
209 data = np.loadtxt(data_filename, delimiter=' ', usecols=(0, 1, 8, 12), unpack=True)
210 1st_column = data[0]
211 2nd_column = data[1]
212 8th_column = data[2]
213 12th_column = data[3]
```

Python: input/output

How to write a file

```
# here I open my file
datafile = "/path/filename.txt"
datafile_id = open(datafile, 'w+')

array1 = …
array2 = …
array3 = x_s[ids]
array4 = …
array5 = …
…
array13 = …
…

data = np.array([array1, array2, array3, array4, array5, …, array13, …])
data = data.T
# here I transpose my data, so to have it in columns

np.savetxt(datafile_id, data, fmt=['%d','%e','%e','%e','%e', …,'%e', ...], header='ID
                    d_sun (pc)    x_star(pc)    y_star(pc)   z_star(pc)   …   [M/H]    …')

# here the ascii file is populated

datafile_id.close()
# close the file
```

Python: exercise

Exercise 1.

Create two arrays (for instance with numpy.arange).
Let's call them array_1 and array_2.
Put some zeros in at least 10 entries of array_2.

Goal: compute array_1 / array_2

Intermediate steps to perform:

I. — use numpy.where to identify the zeros in array_2, and then compute array_1 / array_2
only when the zero entries are not involved.

II. — Use numpy.where to replace the zeros in array_2 with 0.001, and then compute array_1 / array_2
for all the entries.

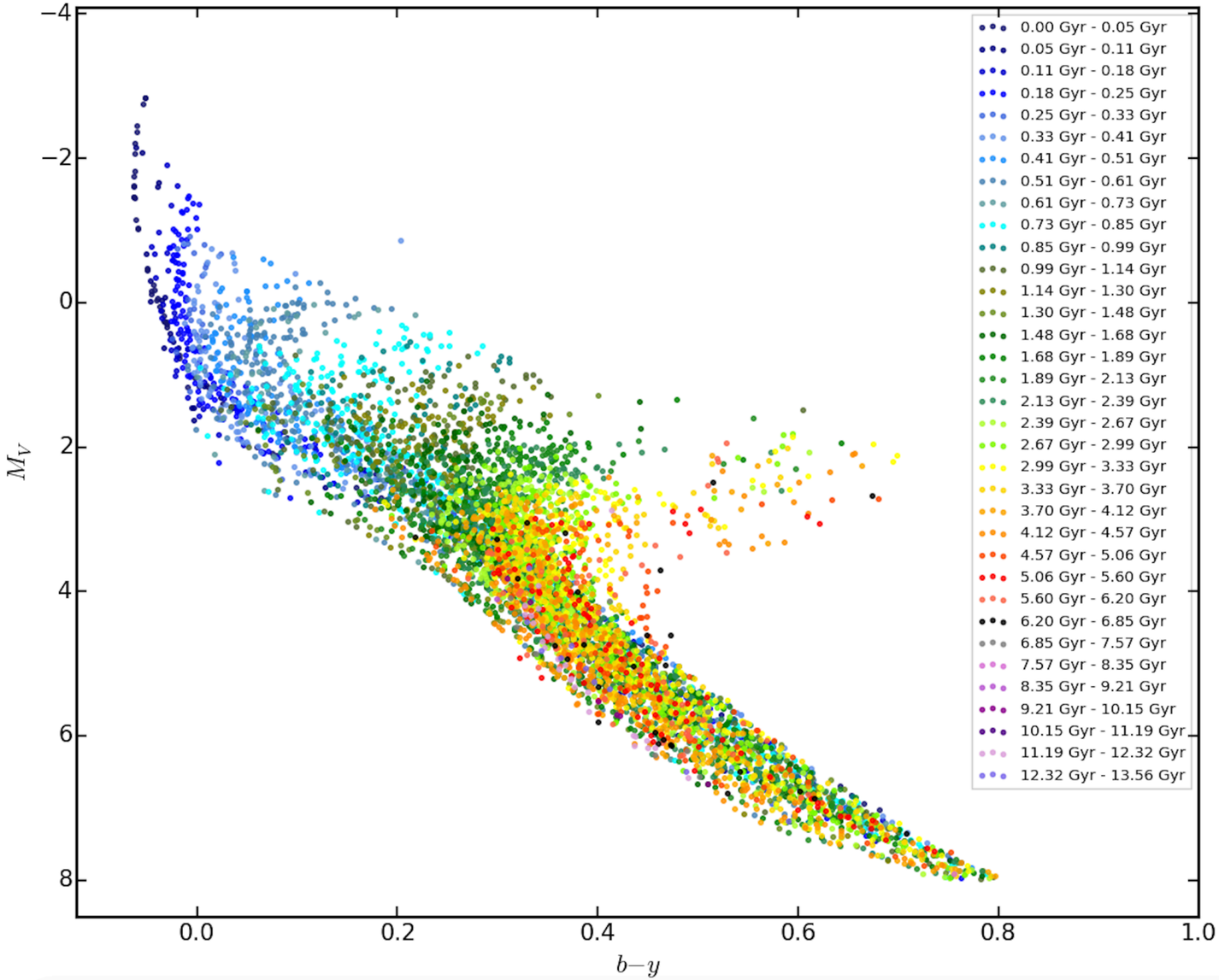
III. — Try to compute array_1 / array_2 (with array_1 and array_2 being the original arrays without modifications)
and print the indices of the resulting array where the division by zero occurs (use numpy.isinf).

Python: exercises

https://github.com/MilenaValentini/TRM_Dati/blob/main/Nemo_6670.dat

Columns of interest in the file:

M_ass
b-y
age_parent [Gyr]



Python: exercises

https://github.com/MilenaValentini/TRM_Dati/blob/main/Nemo_6670.dat

Split the data in age bins (use np.where),
and use a for loop to (over)plot data in different age bins.

matplotlib.pyplot.figure("my_figure", figsize=(14,11))

https://matplotlib.org/stable/gallery/color/named_colors.html

Useful for the scatterplot:

a = matplotlib.pyplot.scatter(array_x, array_y, c=array_z, cmap="colormap")

https://matplotlib.org/stable/gallery/color/colormap_reference.html

matplotlib.pyplot.colorbar(a)