Numerical Methods for Astrophysics:

VISUALIZATION

Michela Mapelli

Python. Matplotlib.pyplot

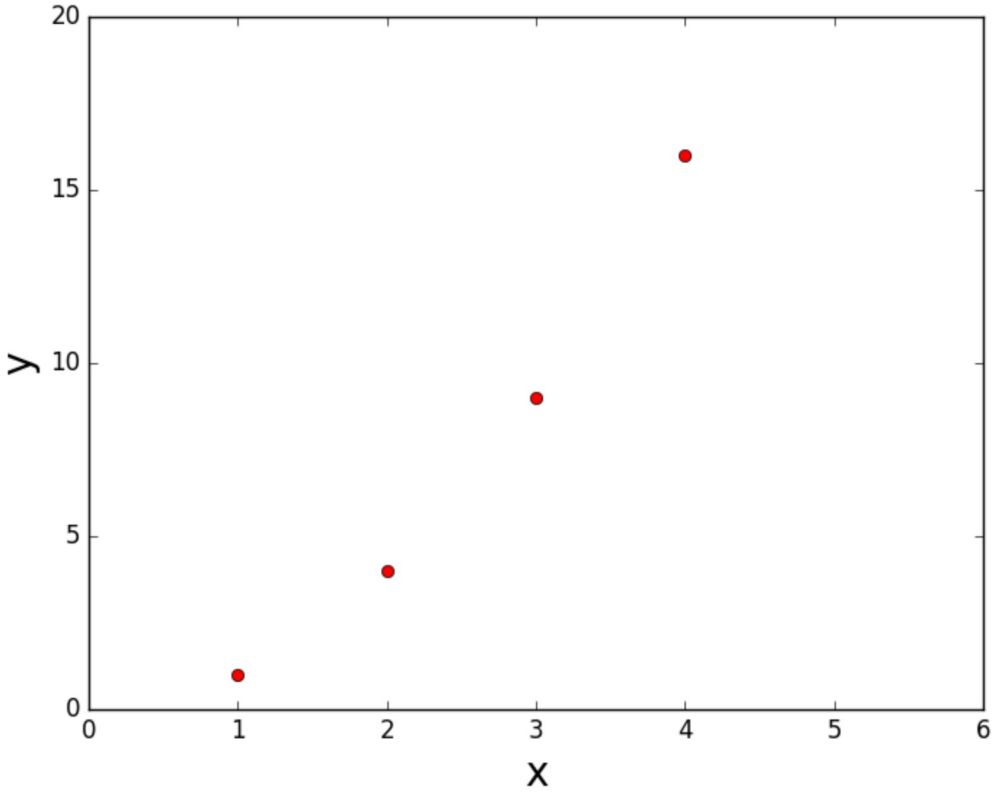
Python is a powerful tool for visualization

Too many options / possibilities → better google them

My preferred one: matplotlib.pyplot

Visualization. Scatter plot

```
#see examples/python/simple_plot.py
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.xlabel('x', fontsize=20)
plt.ylabel('y', fontsize=20)
plt.show()
```

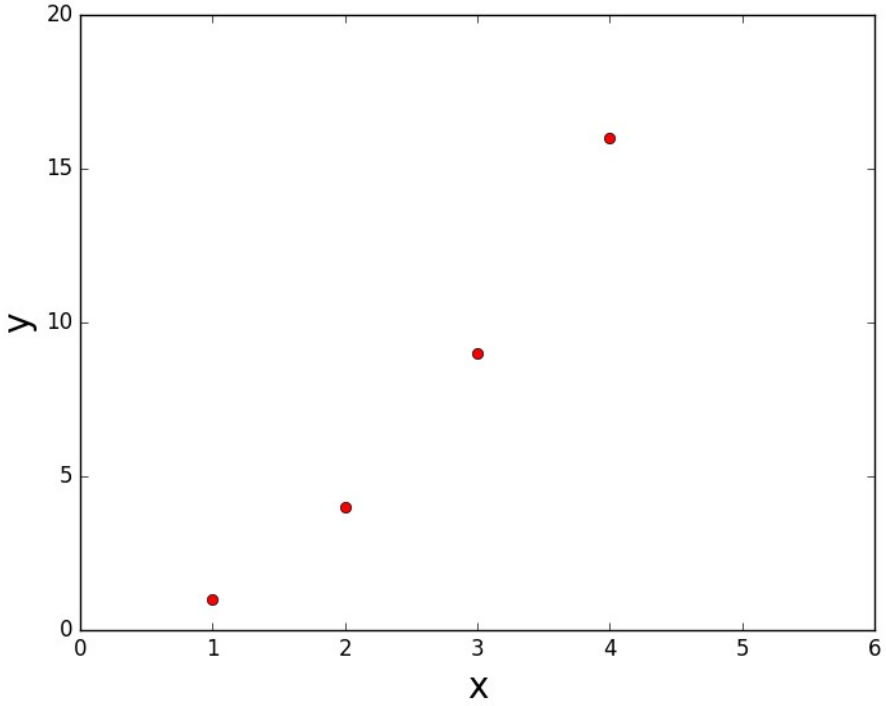


Visualization. Scatter plot

character	description
-	solid line style
--	dashed line style
-.	dash-dot line style
:	dotted line style
.	point marker
,	pixel marker
o	circle marker
v	triangle-down marker
^	triangle-up marker
<	triangle-left marker
>	triangle-right marker
1	tri-down marker
2	tri-up marker
3	tri-left marker
4	tri-right marker
s	square marker
p	pentagon marker
*	star marker
h	hexagon1 marker
H	hexagon2 marker
+	plus marker
x	x marker
D	diamond marker
d	thin-diamond marker
|	vline marker
_	hline marker

Visualization. Scatter plot

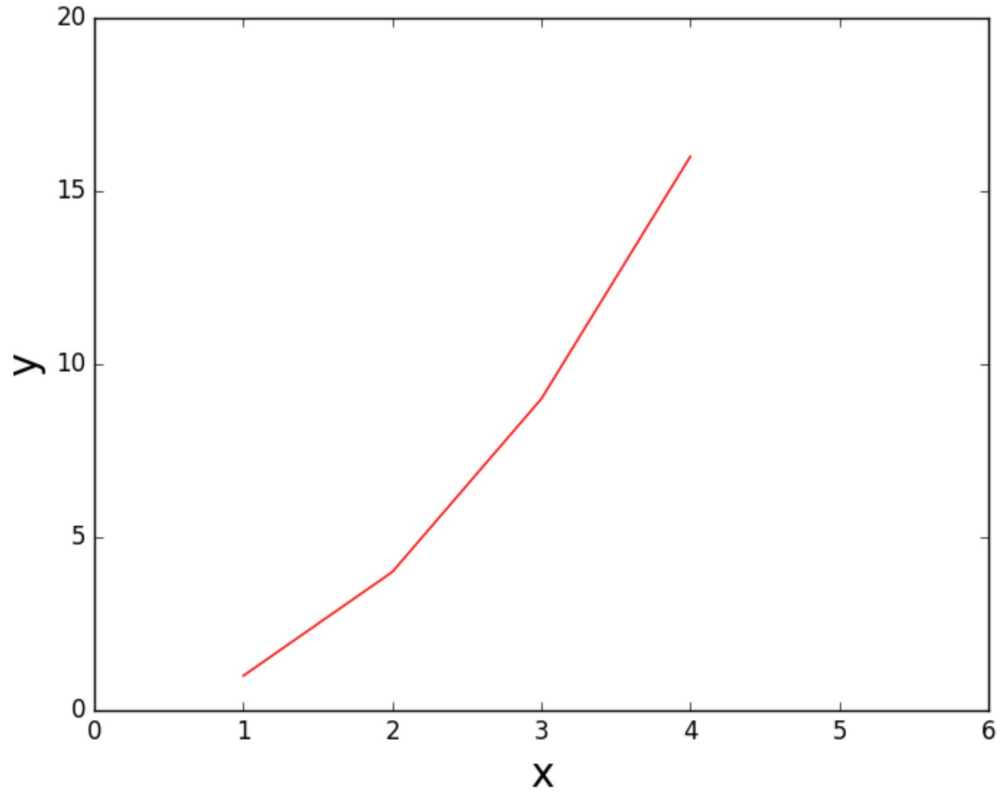
Function matplotlib.pyplot.scatter
is alternative to matplotlib.pyploy.plot




```
#see examples/python/simple_plot.py
import matplotlib.pyplot as plt
plt.scatter([1,2,3,4], [1,4,9,16], color='r',marker='o')
plt.axis([0, 6, 0, 20])
plt.xlabel('x', fontsize=20)
plt.ylabel('y', fontsize=20)
plt.show()
```

Visualization. Line plot

```
#see examples/python/simple_plot.py
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'r-')
plt.axis([0, 6, 0, 20])
plt.xlabel('x', fontsize=20)
plt.ylabel('y', fontsize=20)
plt.show()
```



Visualization. Line type

character	description
-	solid line style
- -	dashed line style
-.	dash-dot line style
:	dotted line style

Visualization. Logarithmic axis, Fontsize

To require the axis is logarithmic

```
plt.yscale('log')
plt.xscale('log')
```

NOTE on FONTSIZE in plots (labels, legends, etc):

The default fontsize of pyplot is usually too small for plots in scientific journals

You can correct it updating the python dictionary that contains figure parameters:

```
plt.rcParams.update({'font.size': 17}) #set default fontsize to 17
```

If you need a smaller/bigger fontsize in just one label, or legend, don't worry: adding the fontisize command in that specific label or legend overrides the above general command rcParams.update

Visualization. Annotating text, mathematical symbols

Use matplotlib.pyplot.text

```
plt.text(1.8,2.0,'$\mathrm{Log-log\,{}plot}$',fontsize=17)
```

NOTE: mathematical symbols in the annotation and in the labels:

Pyplot uses almost the same notation as LATEX for mathematics

When you want to use mathematical notation, you just put it among dollars **$$** as in latex

Example:

```
plt.xlabel('Stellar Mass M$_\odot$', fontsize=20)
```

Visualization. Legend

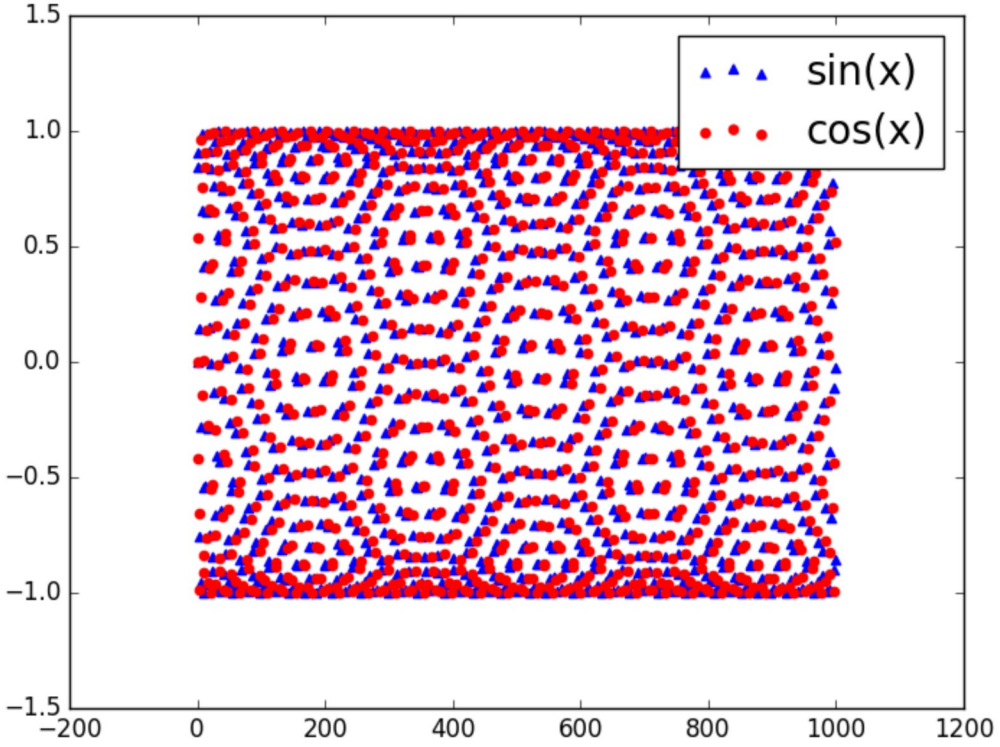
Use matplotlib.pyplot.legend

```
#see examples/python/simple_plot.py
import numpy as np
import matplotlib.pyplot as plt
x=np.zeros(1000,float)
y=np.zeros(1000,float)
y2=np.zeros(1000,float)

for i in range(1,len(x)):
    x[i]=x[i-1]+1.
    y[i]=np.sin(x[i])
    y2[i]=np.cos(x[i])

a= plt.scatter(x,y, color='b',marker='^')
b= plt.scatter(x,y2,color='r',marker='o')
plt.legend([a,b], ['sin(x)','cos(x)'],fontsize='20', \
    loc='upper right')

plt.show()
```



Visualization. Colors

1. Default color cycle is simple:

’b’ (blue), ’g’ (green), ’r’ (red), ’c’ (cyan),
’m’ (magenta), ’y’ (yellow), ’k’ (black).

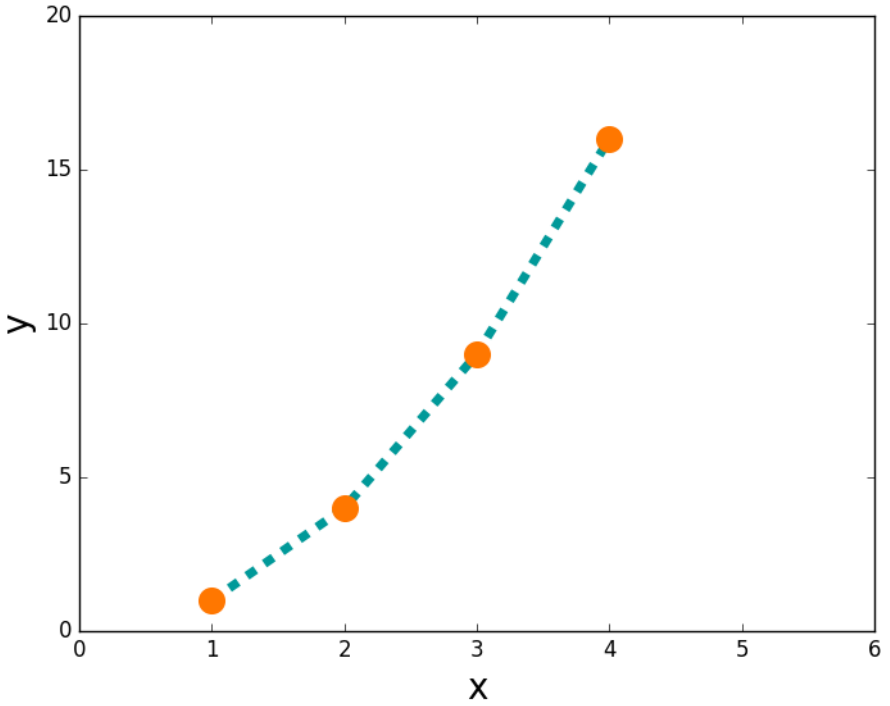
2. Multiple ways to define colors, eg the RGB additive color palette:

color=’#FF7700’
 # means we build a color
First couple of digits define level of Red
Second couple of digits define level of Green
Third couple of digits define level of Blue

where 00 means no Red (or no Green or no Blue)
99 means very luminous Red (or Green or Blue)
FF means most luminous Red (or G or B)

nice but slow to use

Visualization. Colors



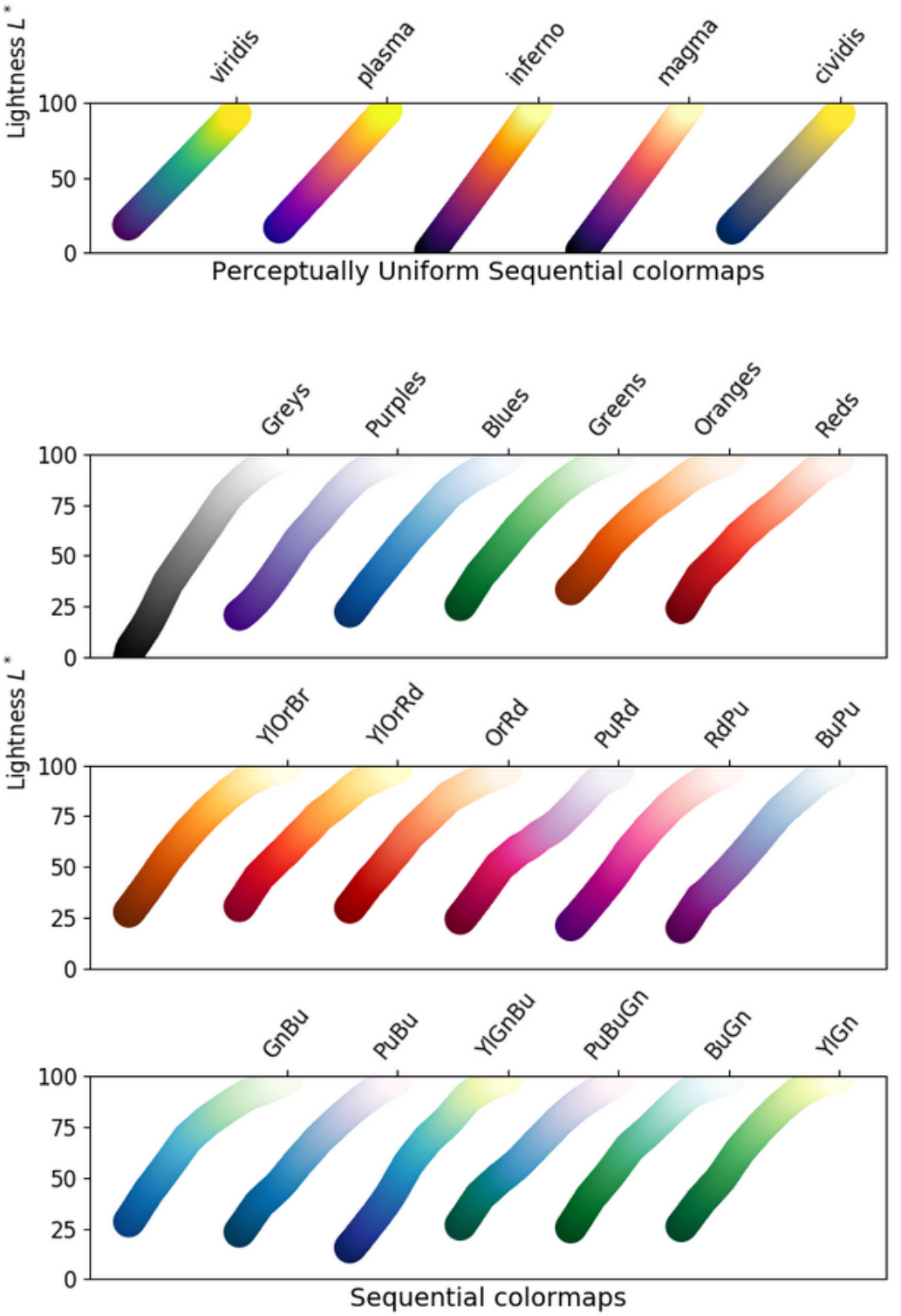

```
#see examples/python/simple_plot.py
import matplotlib.pyplot as plt
plt.scatter([1,2,3,4], [1,4,9,16], color='#FF7700', \
marker='o', s=200, zorder=2)
plt.plot([1,2,3,4], [1,4,9,16], color='#009999',\
linestyle='--', linewidth='5', zorder=1)
plt.axis([0, 6, 0, 20])
plt.xlabel('x', fontsize=20)
plt.ylabel('y', fontsize=20)
plt.show()
```

Visualization. Colors

3. Many more colors than the default color cycle are available in the default matplotlib and can be called just by using their name

Visualization. Colors

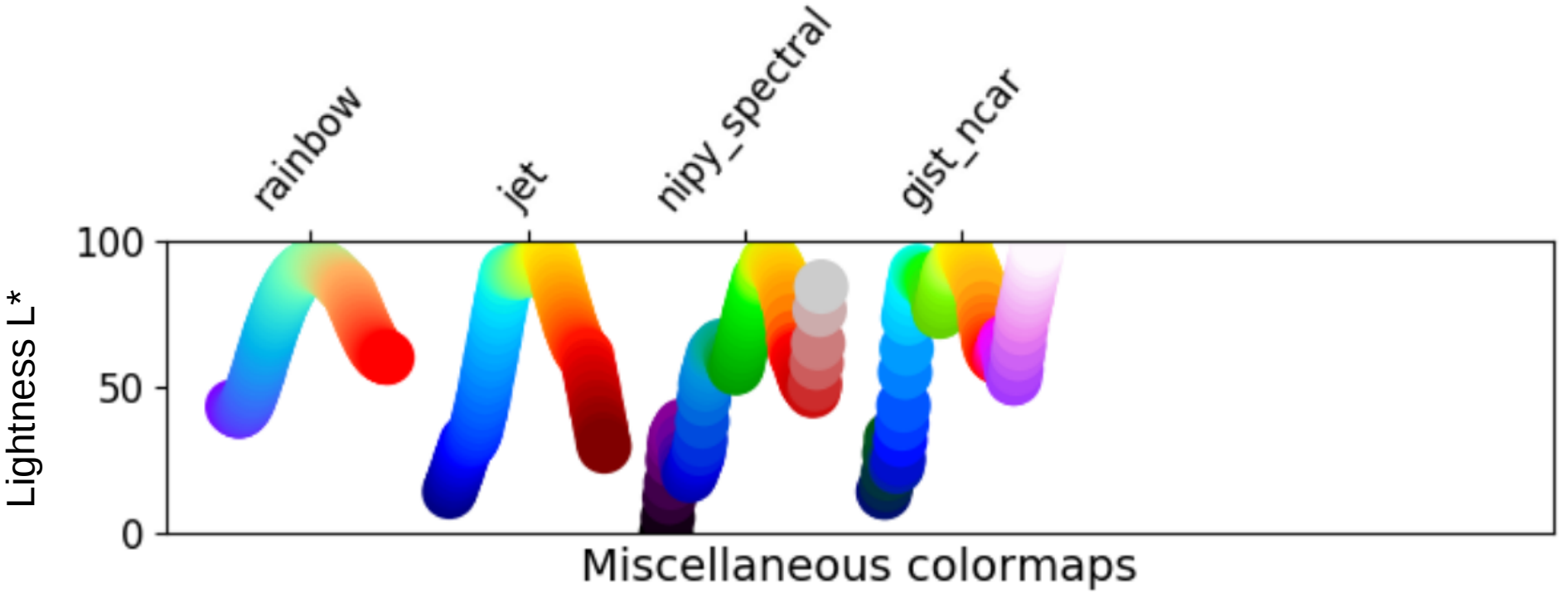
4. You can define additional color maps, better to use the ones which work for color-blind people



Visualization. Colors

4. You can define additional color maps, better to use the ones which work for color-blind people

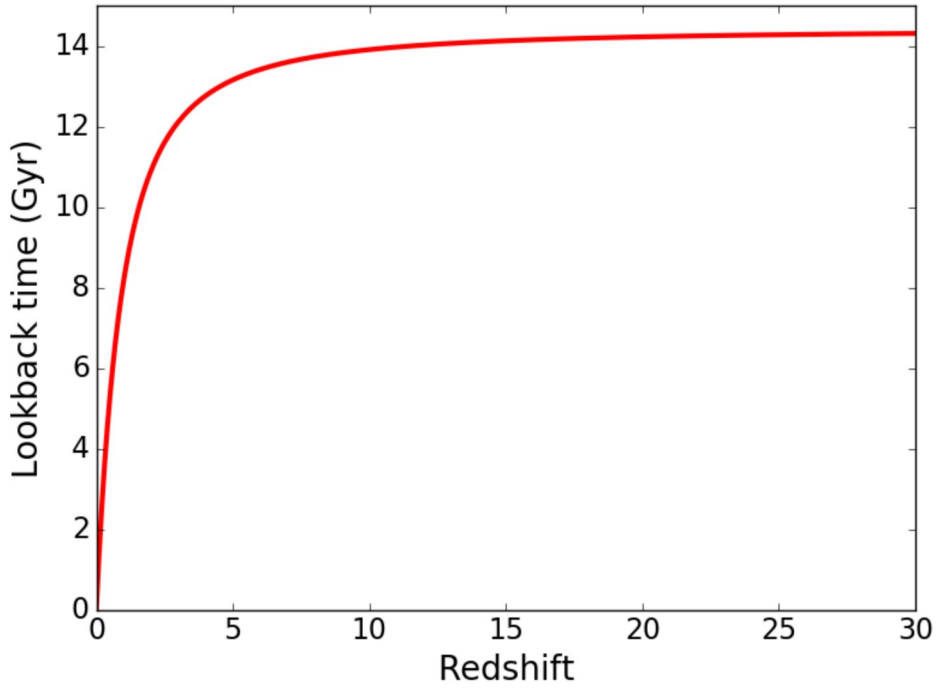
   and avoid those which don't work

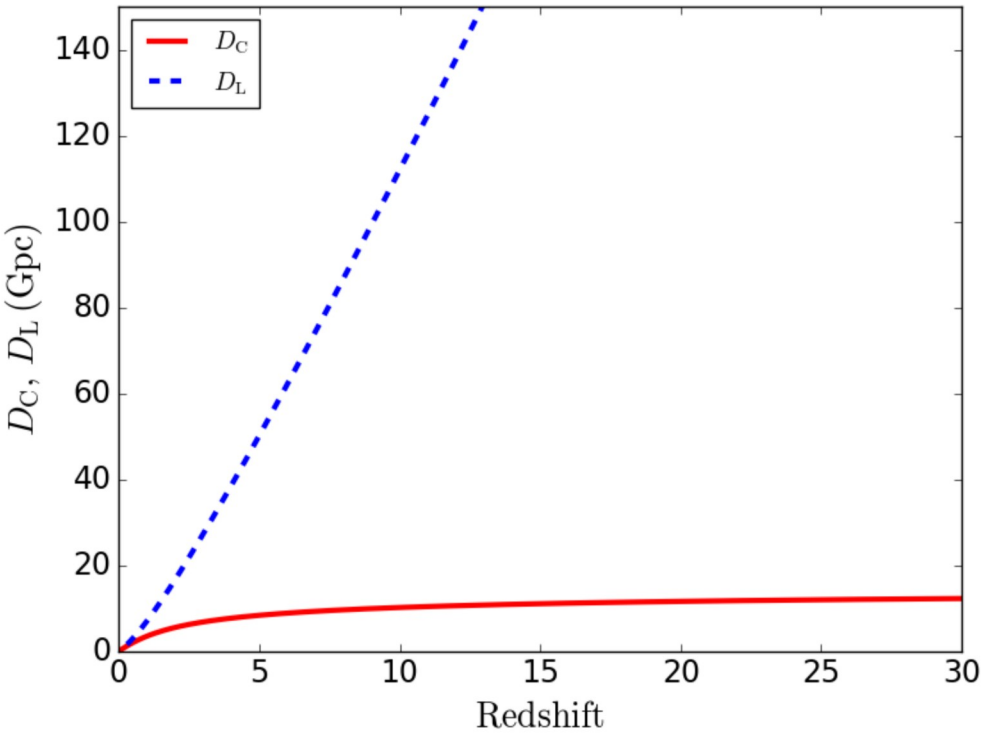


Visualization. Exercise

EXERCISE:

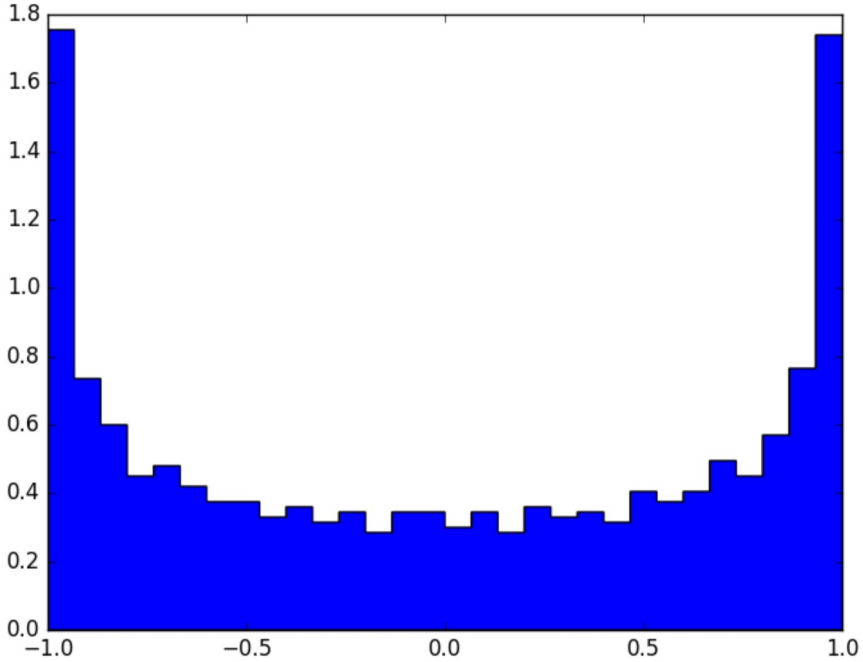
Write a script to plot comoving distance, luminosity distance and look-back time (derived from the previous exercise and example) as a function of redshift. The results should look like Figure 12.





Visualization. Histogram

Use matplotlib.pyplot.hist

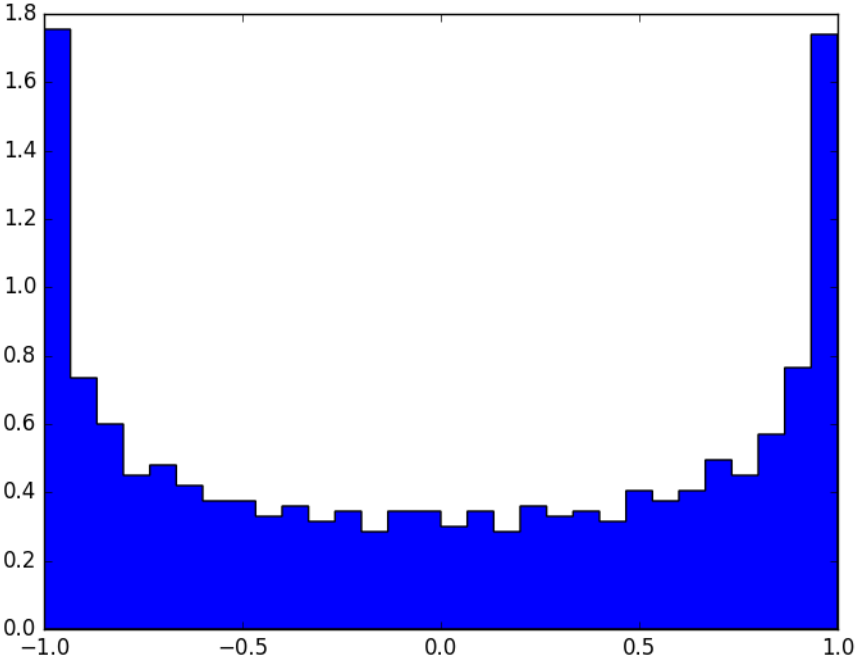



```
#see examples/python/simple_plot.py
import numpy as np
import matplotlib.pyplot as plt
x=np.zeros(1000,float)
y=np.zeros(1000,float)

for i in range(1,len(x)):
    x[i]=x[i-1]+1.
    y[i]=np.sin(x[i])

plt.hist(y, bins=60, density='True',histtype='step')
plt.show()
```

Visualization. Histogram



Use matplotlib.pyplot.hist

Arguments of hist:

bins: integer, number of bins

range: tuple (i.e. set of two arrays), min and max of each bin

density: boolean (True/False), counts normalized to form a probability density (area = 1)

histtype: ‘bar’, ’barstacked’, ’step’, ’stepfilled’

align: ’left’, ’mid’, ’right’ (centered on the left, mid, right bin edge)

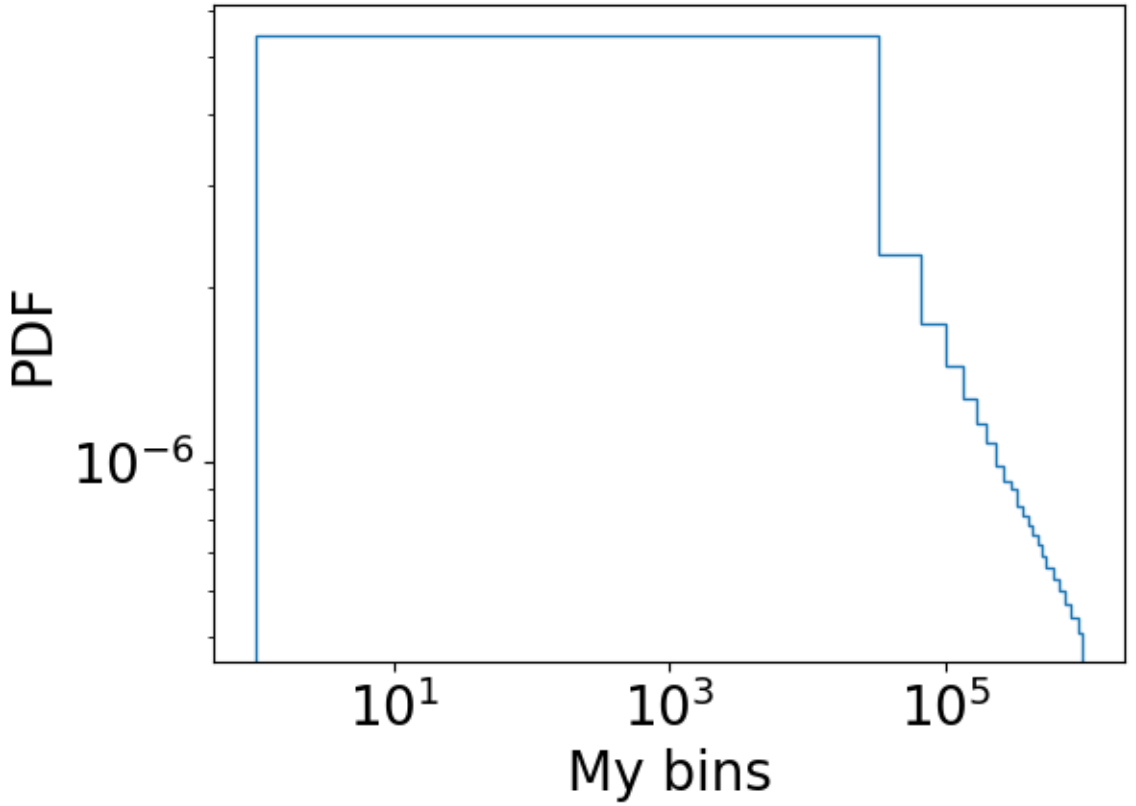
log: log scale

color: color the histogram

Visualization. Log – log histogram

plt.hist(y, bins=30, density='True',histtype='step',log=True)
plt.xscale("log")

It is not sufficient to have a log-log plot!



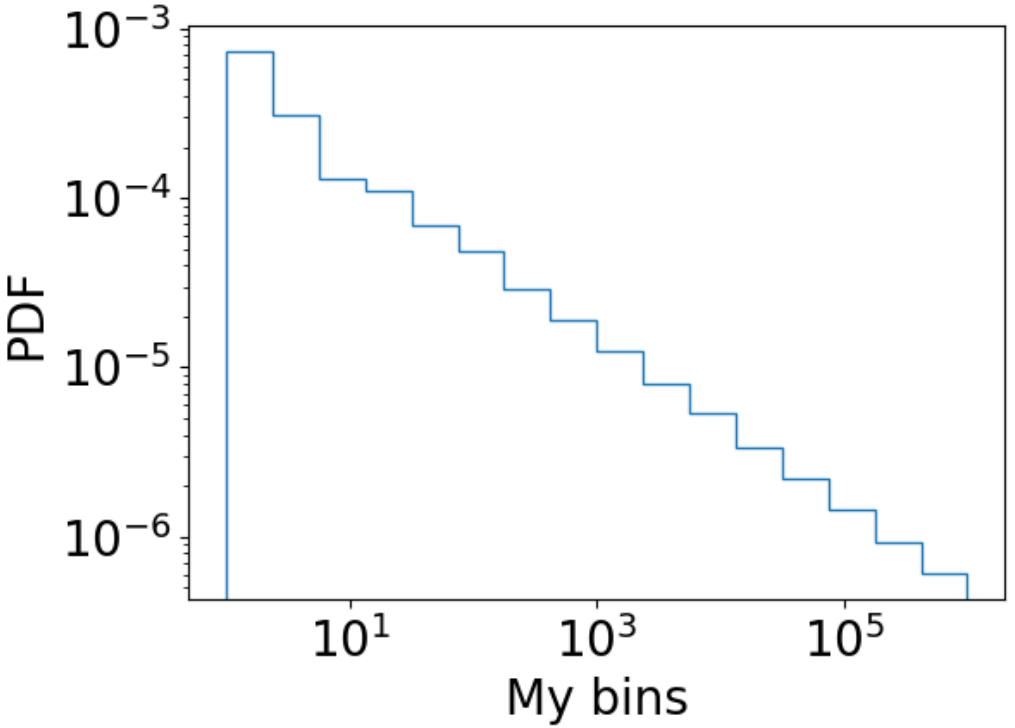
BECAUSE BINS ARE NOT LOGARITHMICALLY SPACED!
They are Δx, they should be Δlog(x)

Visualization. Log – log histogram

examples/python/log_hist.py

Fast way to have logarithmically spaced bins (but of course you can build them “by hand”):

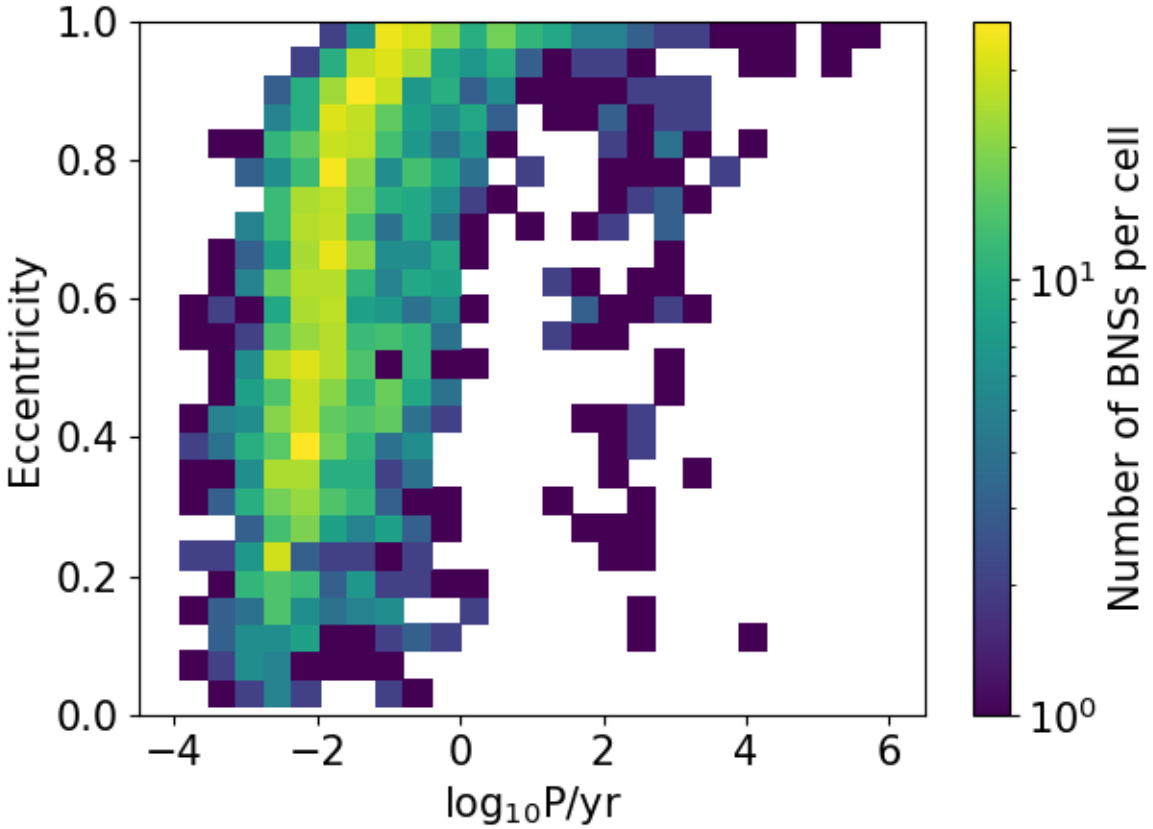
```
mybins=np.logspace(a,b,num=17)
plt.hist(y, bins=mybins, density='True',histtype='step',log=True)
plt.xscale("log")
```



Visualization. Two dimensional histogram

examples/python/BNS_plot.py

The color of each cell represents
the number of objects
with x value between
x – Δx and x + Δx and
with y value between
y – Δy and y + Δy




```
import matplotlib.pyplot as plt
import matplotlib.colors as colors

plt.hist2d(P,e,bins=25,norm=colors.LogNorm())
cbar = plt.colorbar()
cbar.set_label('Number of BNSs per cell')
```

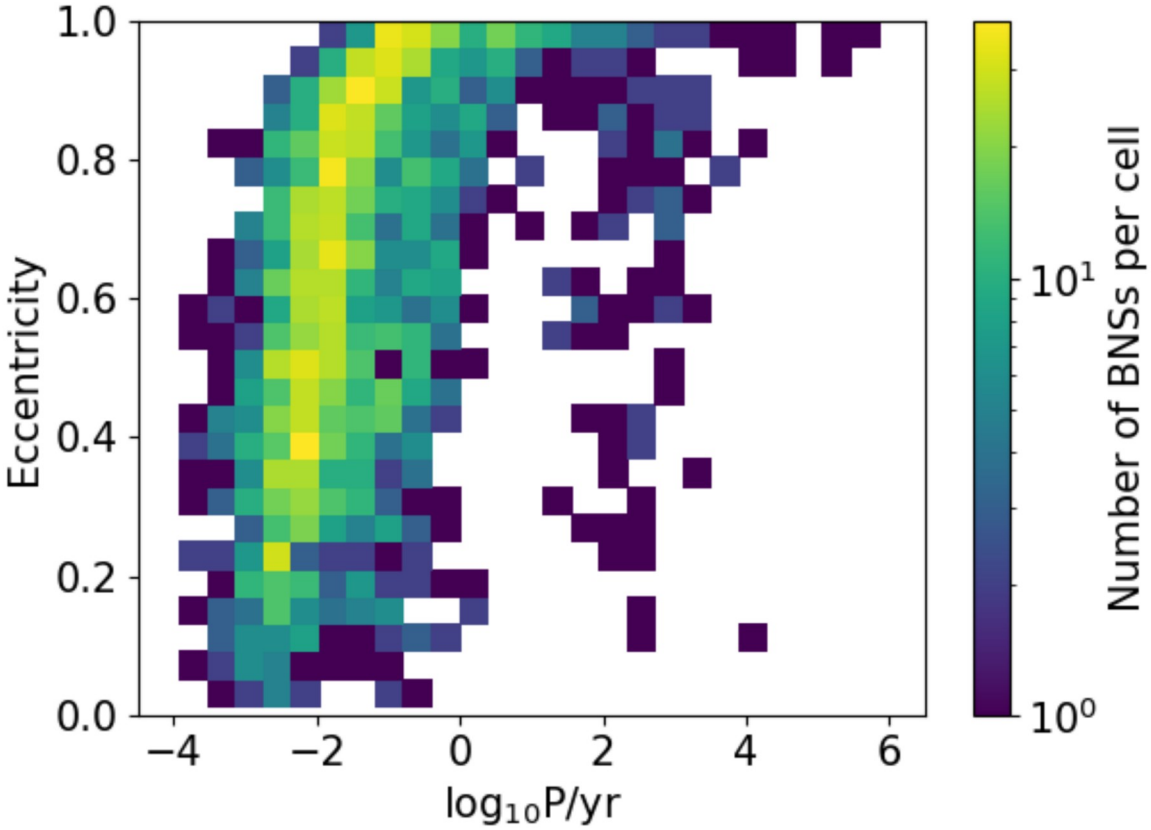
2D histogram from P
and e array (on x and y)
Logarithmic colors

Colorbar and label

Visualization. Two dimensional histogram

examples/python/BNS_plot.py

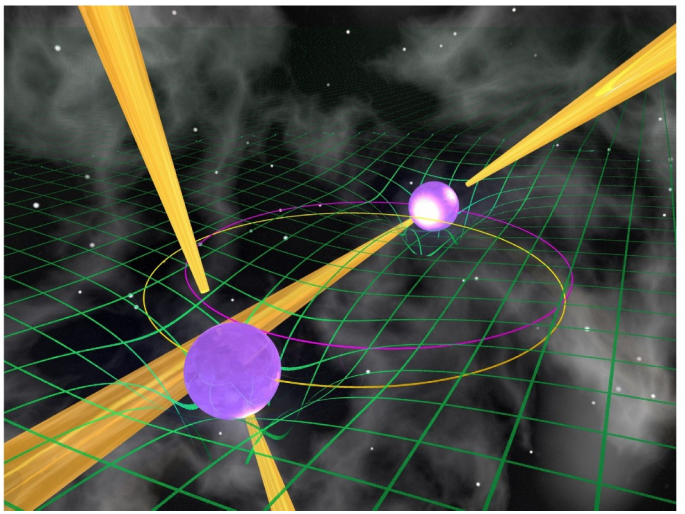
The color of each cell represents
the number of objects
with x value between
$x - \Delta x$ and $x + \Delta x$ and
with y value between
$y - \Delta y$ and $y + \Delta y$



In the example,
number of binary neutron stars
(BNSs:= binary systems composed
of two neutron stars)
with a given eccentricity e and a
given orbital period P
in the Milky Way

Data binary_neutron_stars.txt
come from one of our simulations

In the Milky Way we know
only ~20 BNSs
(one of them is a binary pulsar)



Visualization. Contours

Contours are the same as 2D histograms but smoothed and plotting only some (user-defined) contour levels

Orographic maps are the example of contour plot you are more familiar with since your school days:

x and y are spatial coordinates

colors show height above sea level of a place located at x,y

Points in the same contour-level have the same color

Visualization. Contours

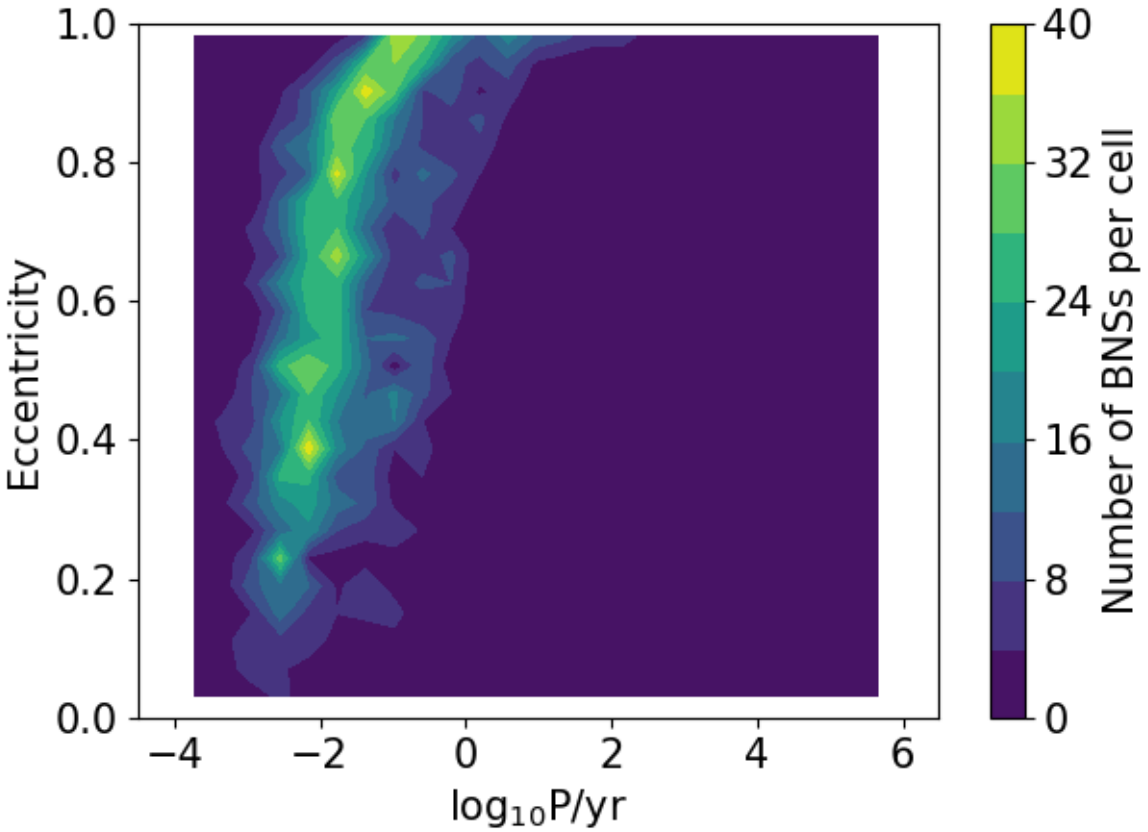
examples/python/simple_contour.py

In our astrophysical example:

x is BNS orbital period

y is BNS eccentricity

Z is number of objects per cell calculated in the previous 2D histogram (it is a “density of BNSs with a given period and eccentricity)



Visualization. Contours

examples/python/simple_contour.py

```
Z,xedges,yedges=np.histogram2d(P,e,bins=25,density=False)
```

I use numpy.histogram2d to calculate the 2D histogram from which I get the Z values

histogram2d is like hist2d but does not plot anything: It just gives me the Z matrix and x,y binned arrays

```
x=np.zeros(len(xedges)-1)
y=np.zeros(len(xedges)-1)
for i in range(len(xedges)-1):
    x[i]=(xedges[i]+xedges[i+1])/2.
    y[i]=(yedges[i]+yedges[i+1])/2.
```

histogram2d gives me the edges of the bins, I want the middle points

```
# to have the matrix in the form needed by contourf you have to transpose
Z = np.transpose(Z)

cs=plt.contourf(x,y, Z, levels=10)

cbar = plt.colorbar(cs,orientation='vertical')
cbar.solids.set_edgecolor("face")
cbar.set_label('Number of BNSs per cell')
```

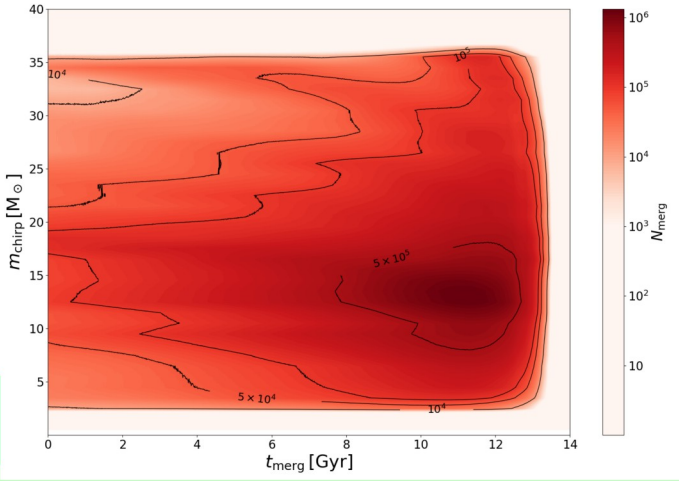
CONTOURF draws the contour plot given x,y,Z and number of levels

Colorbar works both with hist2d and with contour/contourf

NOTE: use **contourf** if you want contours filled with colors and
contour if you want line contours (not filled)

Visualization. Contours

Important note: the files needed to do the exercise are in exercises/python/ (both on gitlab and moodle)



EXERCISE:

Produce a contour map like Fig. 17 with data files chirpmass_bin.dat (array of chirp masses, $M_\odot$), tmerg_bin.dat (array of merger times, Gyr), chirpmass_tmerg_tot.dat (complete matrix to produce the contours). The chirp mass is defined as $m_1^{3/5}\, m_2^{3/5}\, (m_1 + m_2)^{-1/5}$, where m_1 and m_2 are the masses of two compact objects in a binary system. This quantity is important for gravitational waves, because the frequency of gravitational waves changes as $\dot{f}_{GW} \propto m_{chirp}^{5/3}$ during inspiral. Hence, chirp mass can be directly derived from gravitational wave data, given frequency and frequency derivative with time. The merger time is the look-back time when a merger happened. These data come from a theoretical study on the cosmic merger rate of binary black holes [Mapelli et al., 2017].

Visualization. Contours

Chirp mass versus lookback time of merger : What is this?

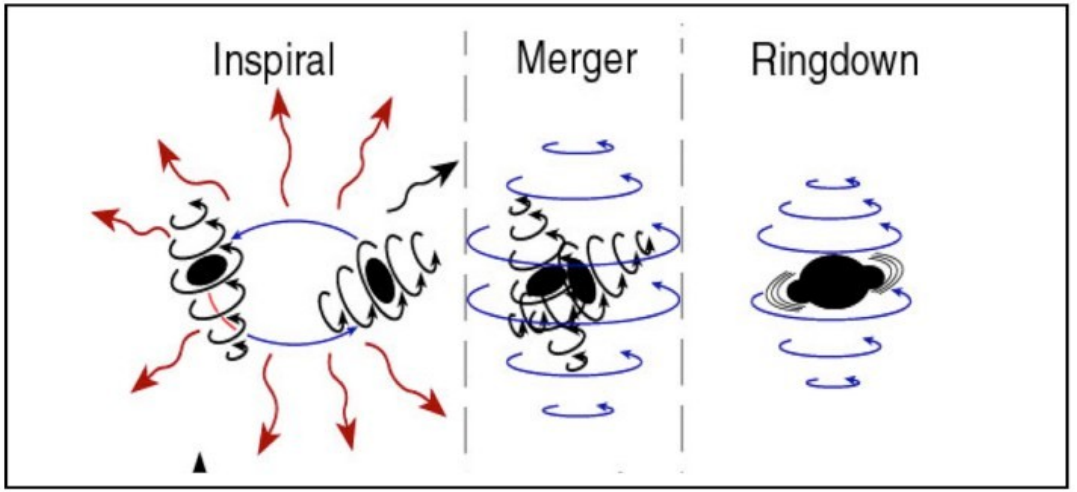
* A binary compact object loses orbital energy by gravitational wave emission → semi-major axis shrinks

during inspiral

$$\frac{\mathrm{d}f_{\mathrm{GW}}}{\mathrm{d}t} \propto f_{\mathrm{GW}}^{11/3}\, m_{\mathrm{chirp}}^{5/3}$$

where

$$m_{\mathrm{chirp}} = m_1^{3/5} m_2^{3/5} (m_1 + m_2)^{-1/5}$$



©Kip Thorne

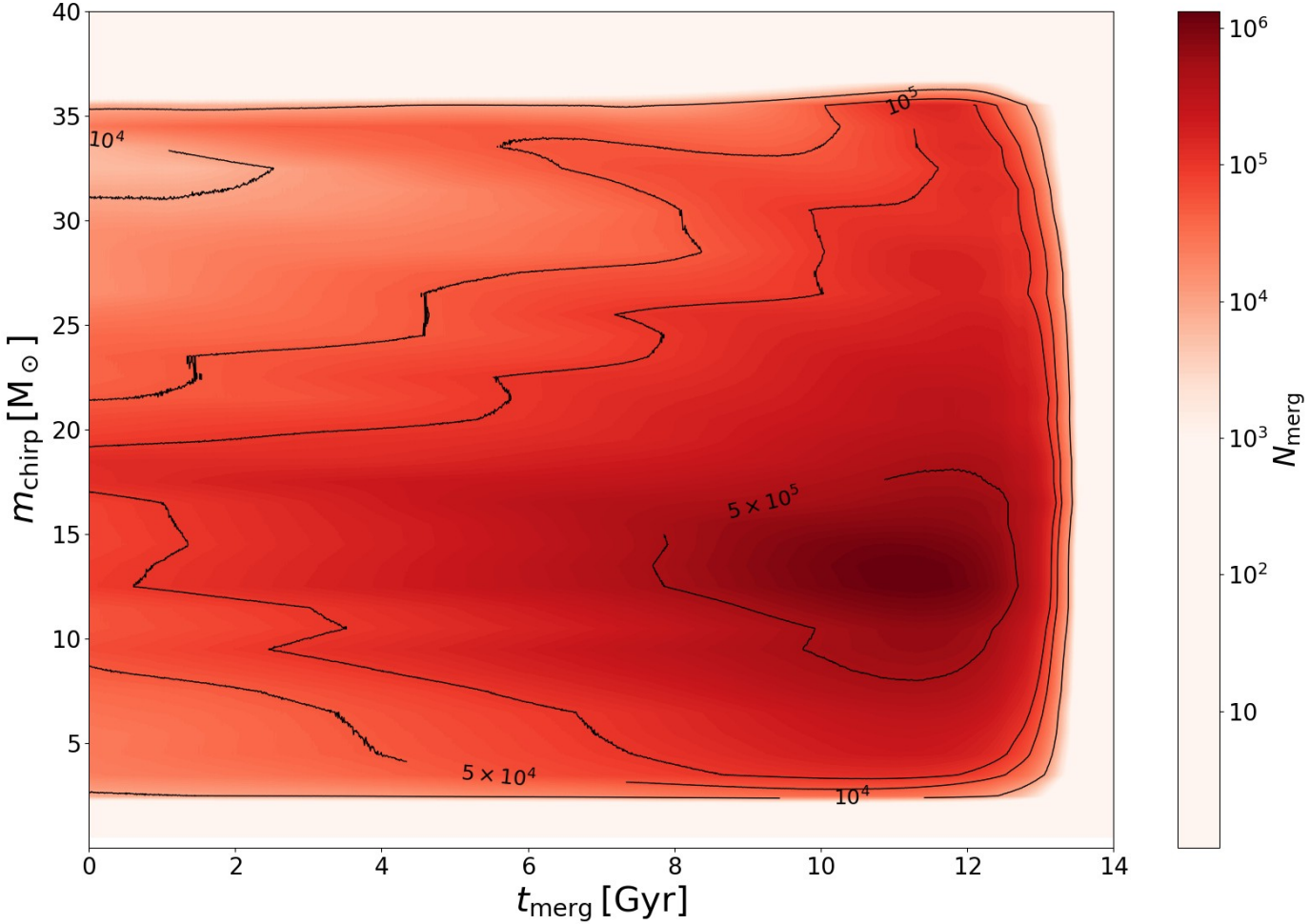
→ from the measure of GW frequency and frequency derivative we derive the chirp mass

* The plot I showed assumes a cosmological evolution model + plants compact binaries with their properties in the model + plots when they merge (in lookback time) versus their chirp mass

(for more details see https://arxiv.org/abs/1708.05722 or ask me)

Visualization. Contours

Important note: the files needed to do the exercise are in exercises/python/ (both on gitlab and moodle)



Chirp mass versus lookback time of merger

Visualization. Contours

Now you build not only the contourf but also the Z matrix (which represents the number of mergers per cell):

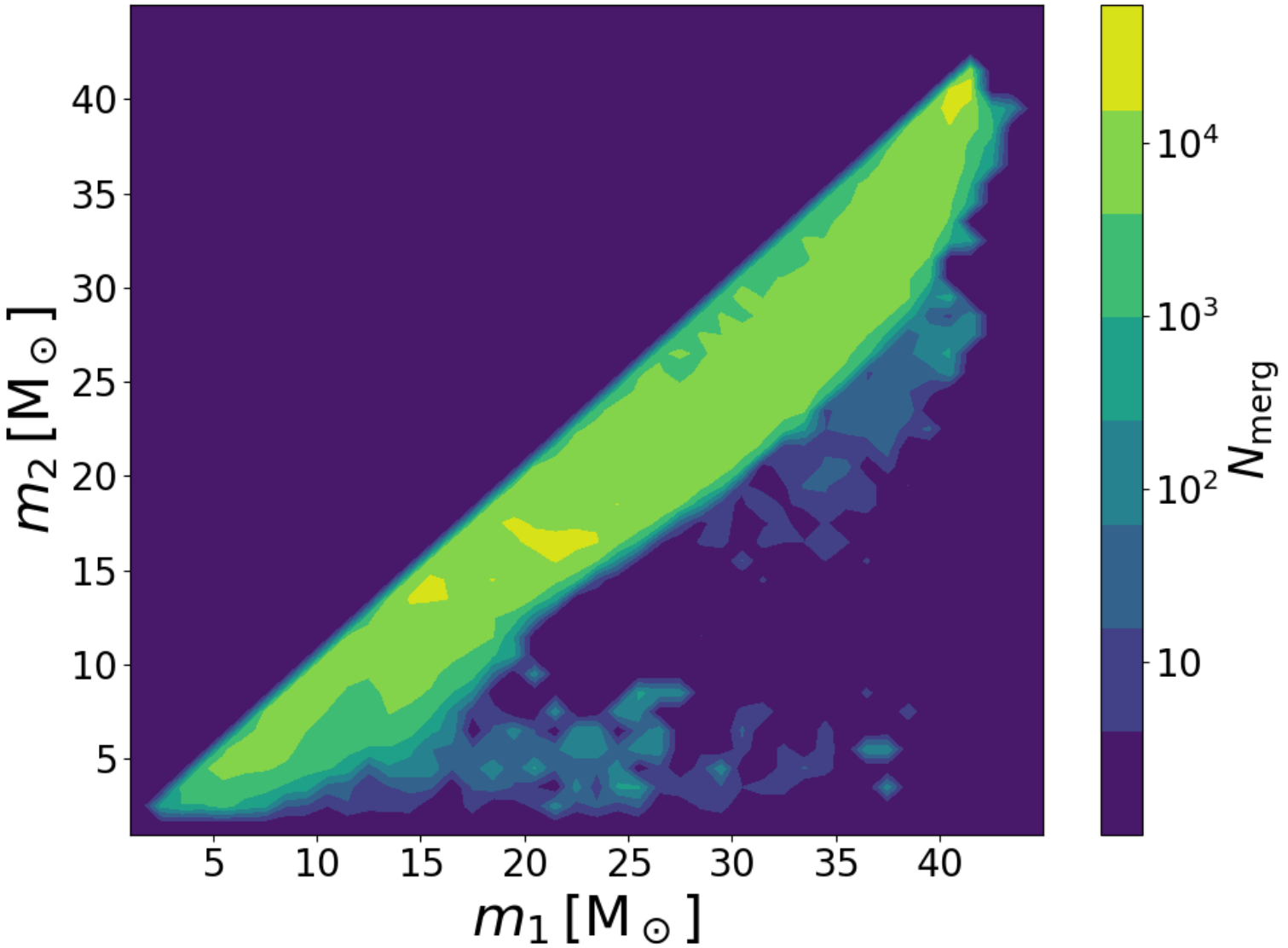
EXERCISE:

Produce a new contour map of the mass of the secondary black hole (i.e. the lighter one) versus the mass of the primary black hole (i.e. the more massive one) considering a sample of theoretically generated binary black holes. Unlike the previous exercise, here you have to generate the matrix z (containing the number of binary black holes in each cell with primary mass between $m_1 + \delta m$ and $m_1 - \delta m$ with $\delta m = 0.5\ M_\odot$ and with secondary mass between $m_2 + \delta m$ and $m_2 - \delta m$ with $\delta m = 0.5\ M_\odot$). The file you should start from is time_BHillustris1_30.dat (look at the comments in the first line to understand the meaning of the columns). Columns 7 and 8 are the masses of the two black holes. Note that the black hole in column 7 is not necessarily the most massive: you should swap the two black holes if the one in column 7 is lighter than the one in column 8. If you succeed, you should be able to recover a contour plot as the one in Figure 18.

Important note: the file time_BHillustris1_30.dat is in exercises/python/ (both on gitlab and moodle)

Visualization. Contours

Here is how the result should look like



Important note: the file time_BHillustris1_30.dat is in exercises/python/ (both on gitlab and moodle)

Visualization. Contours

Suggestion: Note that time_BHillustris1_30.dat is quite a large file. If you read the arrays of m_1 and m_2 and then you do a couple of nested for loops to calculate the matrix z, your program will be very slow. To speed it up significantly you can use the following consideration.
I define the edges of the mass bins as

```
bin[0]=0
bin[1]=bin[0]+dm
bin[2]=bin[1]+dm
...
bin[n]=bin[n-1]+dm,
```

where dm = $(m_{\max} - m_{\min})$/N and N = number of bins.
Then I can assign the indexes as

```
index1 = int(mass1/dm)
index2 = int(mass2/dm)
```

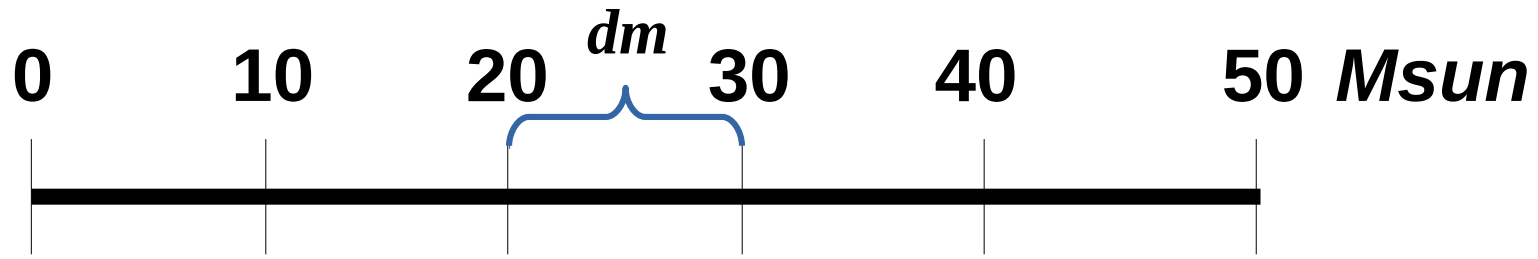
Finally, I calculate the table as

```
nmerg[index2][index1] +=1
```

Visualization. Contours

*** First read the masses m1, m2. They are millions so to develop the script use only the first ones**

*** Choose the bins for the contour plots on x and y axis**



binx =np.zeros(6,float)
binx[0]=0, binx[1]=10.,...

biny =np.zeros(6,float)
biny[0]=0, biny[1]=10.,...

*** build the contour matrix: contains in each cell ij the number of binary black holes with binx[i]<m1<=binx[i+1] and biny[j]<m2<=biny[j+1]**

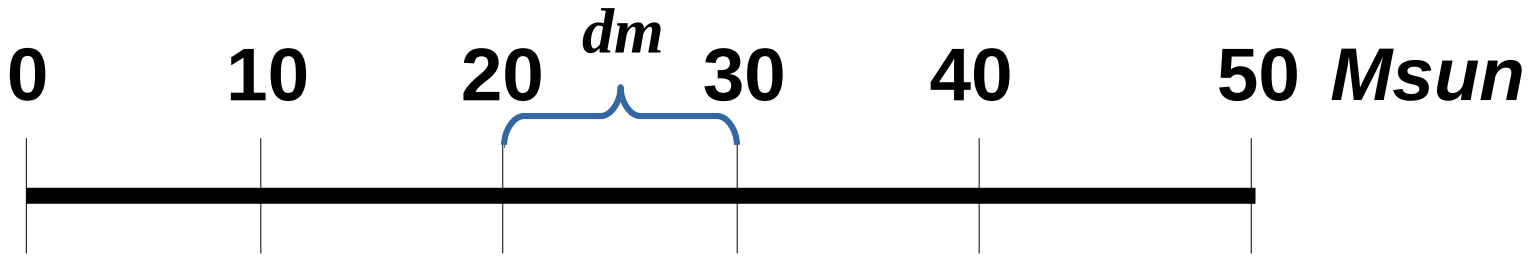
you need to check for each binary black hole if (m1[k]>binx[i]) and (m1[k]<=binx[i+1]) and (m2[k]>biny[j]) and (m2[k]<=biny[j+1])

DOUBLE LOOP SUICIDAL FOR THE SIZE OF THE DATA

*** To avoid the double loop, use the properties of the indexes**

Visualization. Contours

*** To avoid the double loop, use the properties of the indexes**
Suppose m1[0]=14 Msun, m2[0]=33 Msun



m1[0] ends up in binx[1]
m2[0] ends up in biny[3]

calculate index1=int(m1[0]/dm)=1
calculate index2=int(m2[0]/dm)=3
corresponds to the correct bins

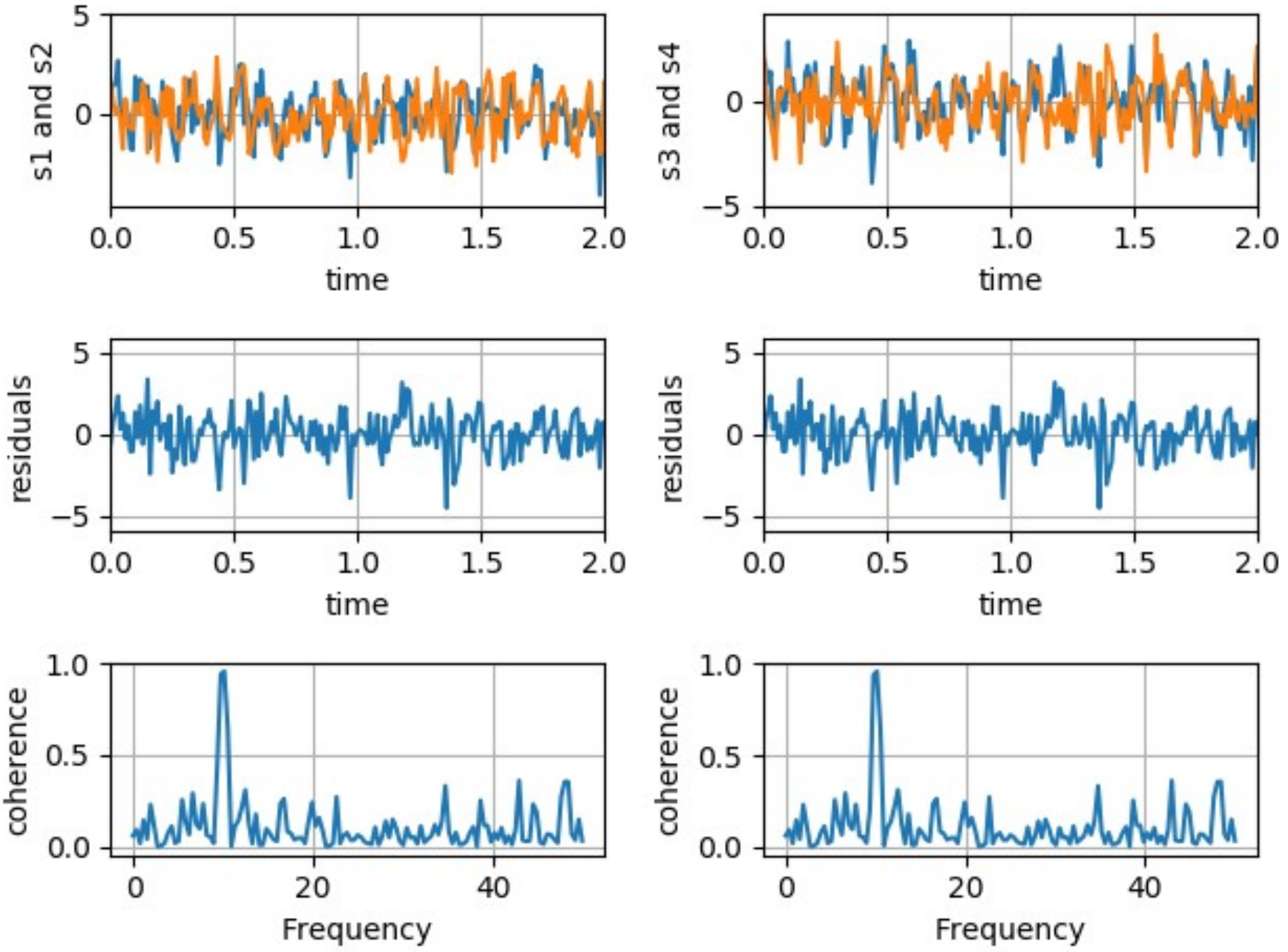
Then you can first calculate the indexes of the matrix and then assign the matrix of the contours num=np.zeros([N,N],float)

```
index1=int(m1[0]/dm)
index2=int(m2[0]/dm)
num[index1][index2]+=1
```

In the same loop when you read the masses!

Visualization. Subplots

see examples/python/subplots.py



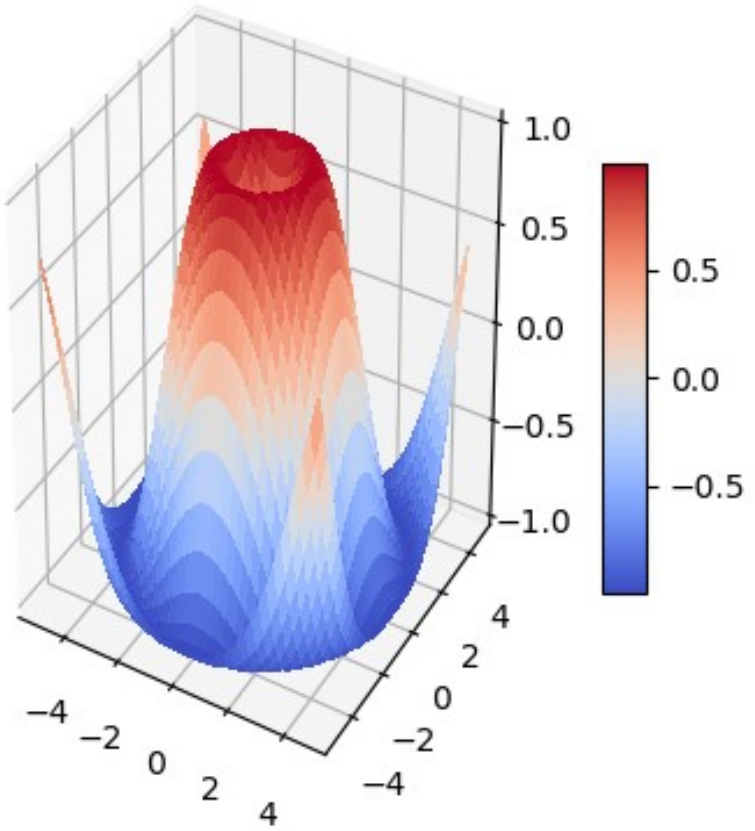
Visualization. Subplots

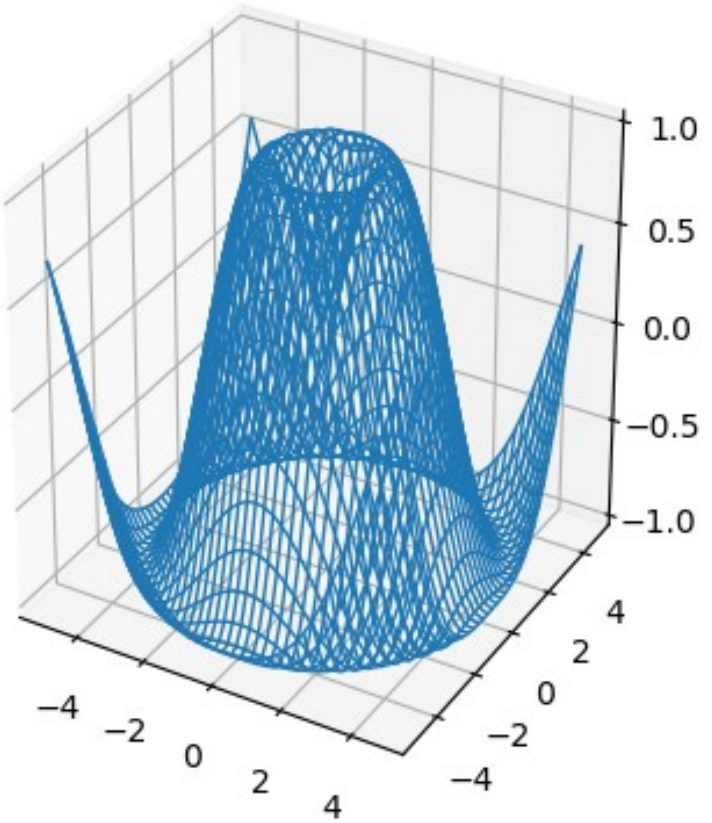
EXERCISE:

Come back to the file time_BHillustris1_30.dat you read to perform the previous EXERCISE and produce two subplots (just one row, two columns). In the first column, you report the contour plot of the mass of the secondary black hole (i.e. the lighter one) versus the mass of the primary black hole (i.e. the more massive one). In the second column, you should plot the chirp mass versus the total mass ($m_1 + m_2$). If you want to add other subplots, you are free to experiment with the quantities provided in the file time_BHillustris1_30.dat.

Visualization. Three dimensional plots

see examples/python/plots3d.py





Visualization. Three dimensional plots

EXERCISE:

Come back to the file time_BHillustris1_30.dat and produce a 3D scatter plot.
Unlike the surface 3D plot presented in the main lecture, a 3D scatter plot needs the function **scatter** (instead of plot_surface or plot_wireframe). The function scatter reads data from three mono-dimensional arrays (or lists) that contain the three different quantities of the same object we want to plot on three different axes.
In this exercise you will consider column 4 of time_BHillustris1_30.dat (metallicity of the progenitor stars of the simulated black holes, in absolute values Z), column 7 and 8 (mass of the first and mass of the second black hole in $M_\odot$) and column 9 (delay time in Gyr, i.e. time elapsed from the formation of the progenitor stars to the merger of the two black holes). **IMPORTANT: please read only the first 10'000 lines of the file time_BHillustris1_30.dat, otherwise your plot becomes too crowded and heavy.**
You will produce a 3D scatter plot similar to the one in Figure 21, where the x-axis shows the total mass of the binary $(m_1 + m_2)$, the y-axis shows the metallicity of the progenitor star (Z) and the z-axis shows the delay time. Finally, the Figure also shows the chirp mass of the system $(m_1\, m_2)^{3/5}\, (m_1 + m_2)^{-1/5}$ as colour gradient (see the colour map). You should be able to find out this feature of the scatter function by looking at the matplotlib manuals on the internet. Otherwise, just ask me.

Visualization. Three dimensional plots

How the result of exercise should look like