Quiz

Question: Capitalize all words in a title entered, except a, an, the, at, by, for, in, of, on, to, up, and, as, but, or, and nor. Your output should look like as following.

Enter a title: Welcome to MSDM_5002 of data-driven modeling for MSc students offered by phys&math department in UST. We will continue to learn NumPy.

Capitalized: Welcome to MSDM_5002 of Data-Driven Modeling for MSc Students Offered by Phys&Math Department in UST. We Will Continue to Learn NumPy.

- Find some build-in functions or library online??

- Write as many functions as possible by yourself??

How to represent an image

- The idea is very simple. We can use a function like F(x,y), which represents all the colors in the position of (x,y). In general, F(x,y) should be vector functions.

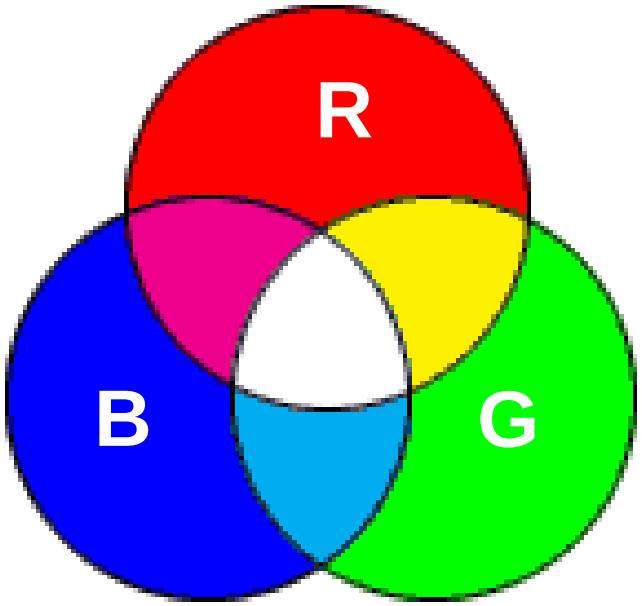
- There are many ways to digitalize colors, and usually we use the following three types:

1. **Grayscale**, F(x,y) is a single value function. The image will only contain the information about dark or light, but no colors.

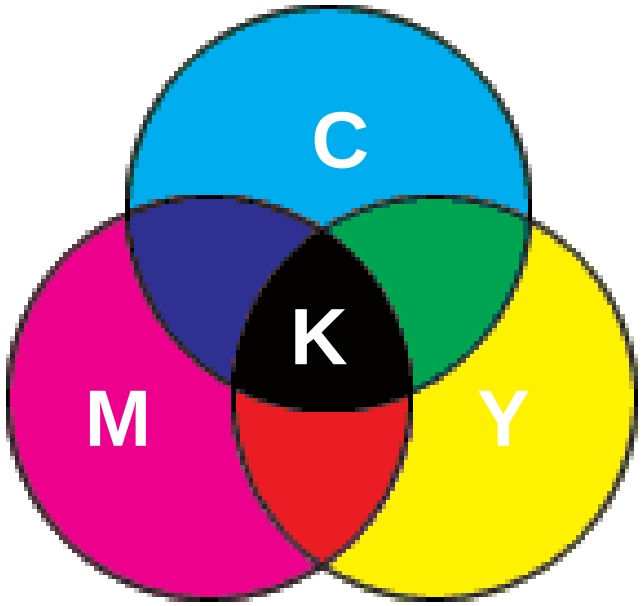
2. **RGB**, F(x,y) is now a vector function and consists of ***Red***(x,y), ***Green***(x,y) and ***Blue***(x,y).

3. **CMYK**, F(x,y) is now a vector function and consists of ***Cyan***(x,y)蓝绿色, ***Magenta***(x,y)品红色, ***Yellow***(x,y)黄色, and ***Key/Black***(x,y)黑色.

Comparison of RGB and CMYK



RGB Colors



CMYK Colors

- In most of cases, we print everything in CMYK or spot colors, so it is better to design your images in CMYK color space, otherwise there will be a color shift when you print your designs.

Four-color printing

Quick demonstration on how CMYK printing works

https://www.youtube.com/watch?v=nHVIL2ZVQIU

An example of a color printer

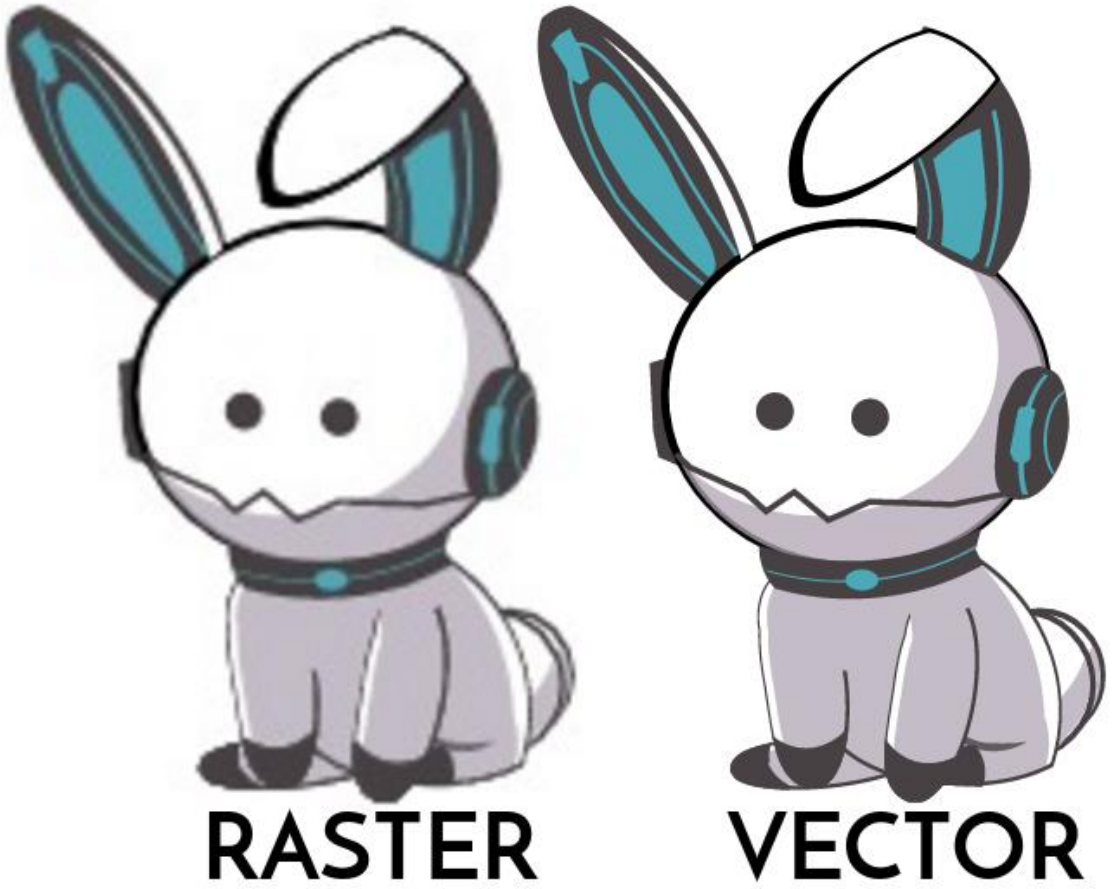
How to store F(x,y) in computer

- The most natural way to store all the information F(x,y) for an image is that we use some matrices or table to store all the values of F(x,y), and then we can convert F(x,y) to be colors when we want to print the images. To do this, we need to first discretize (x, y) to convert the continuous x and y to be finite points. We can only have the information for given (x,y)

- Another way is that somehow we write many equations, from which we can calculate the values of F(x,y) for any given (x,y), and then we will first calculate F(x,y) when we want to print the images. In this way, in principle, we can get the information for any (x,y).

Two different types of graphics

- Based on the method we get F(x,y), there are two qualitatively different types of computer graphics: Raster (Bitmap) and Vector.



https://vectr.com/zh-hant/tutorials/what-are-vector-graphics/

Bitmap (Raster) graphics

- Raster (or bitmap) graphics are made up of tiny squares called pixels. Once a raster graphic is created at a certain size (i.e. a fixed number of pixels), it can't be scaled up without losing image quality.

- The larger the amount of pixels in an image, the larger the file size – they are positively correlated since the computer needs to store information on every single pixel. Widely used raster file formats are .jpg, .png, .gif, .bmp, and .tiff.

Vector graphics

- Vector graphics use mathematical equations to draw out your designs. These mathematical equations are translated into points that are connected by either lines or curves, also known as vector paths, and they make up all the different shapes you see in a vector graphic.

- This allows vector graphics to be scaled to any size without sacrificing image quality as well as maintain a small file size. Common vector file formats are .svg, .cgm, .odg, .eps, and .xml.

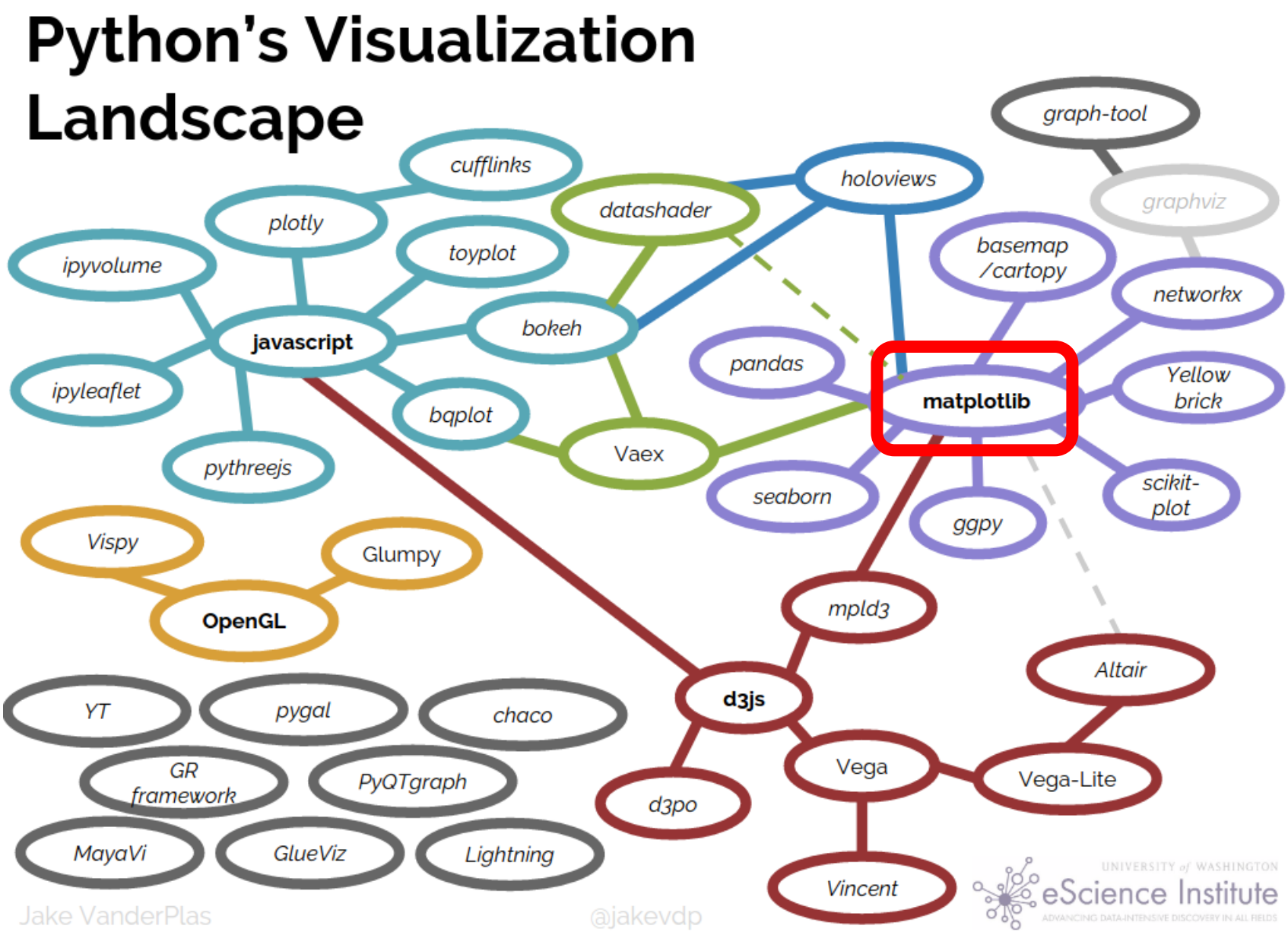
Comparison between Raster and vector graphics

- Raster graphic editors are optimal for digital photograph editing because raster graphics are able to portray better color depth. Each pixel can be any one of the 16 million different colors.
- But if you’re not working with digital photographs, Vector graphics editors would be your best bet for all other types of design editing, especially because vector graphics are able to be scaled and manipulated at any size with clarity.

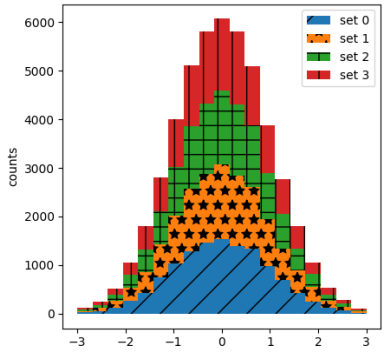
The most used image file format

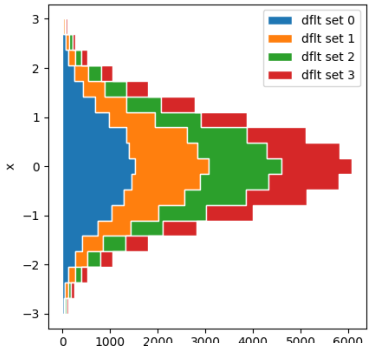
- There are many different file formats for both Raster and Vector graphics, and we should choose the most appropriate one.
- For Raster graphics, we usually use **JPEG**, **TIFF**, **BMP**, PNG, GIF, etc.
- For Vector graphics, we usually use CGM, SVG, AI (Adobe Illustrator Artwork), etc.
- There are also many file format could be used to save both Raster and Vector graphics, such as **EPS** (Encapsulated PostScript), **PDF** (Portable Document Format), **PostScript** (a page description language), etc.

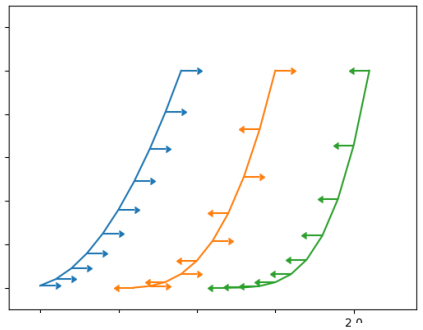
Packages about visualization in Python

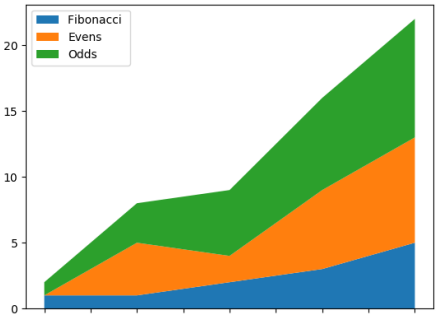


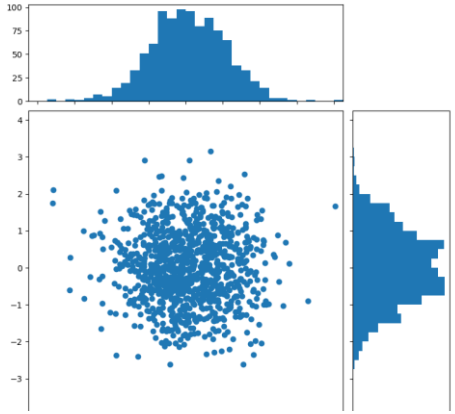
What we can get from Python

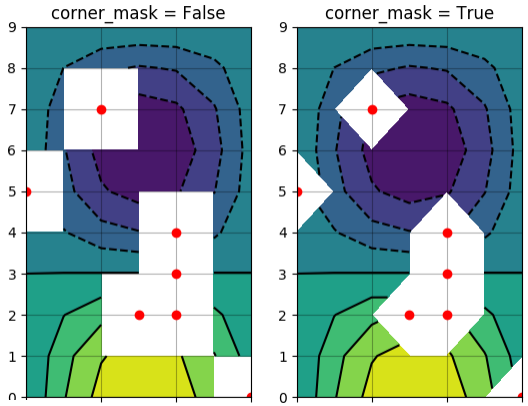


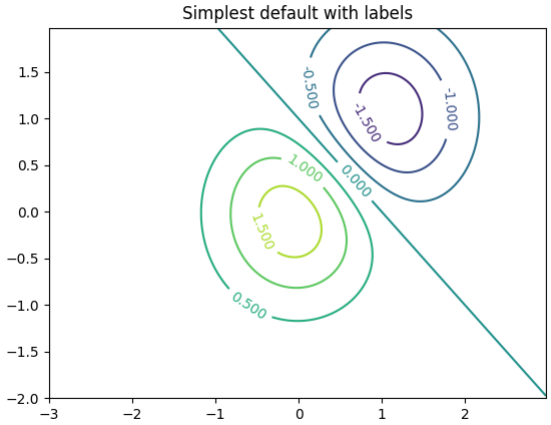


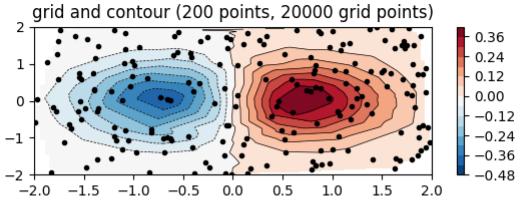


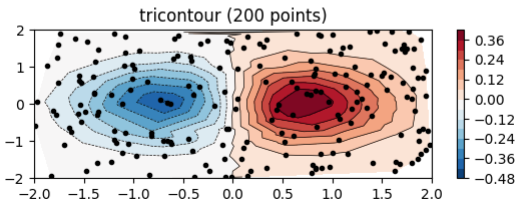


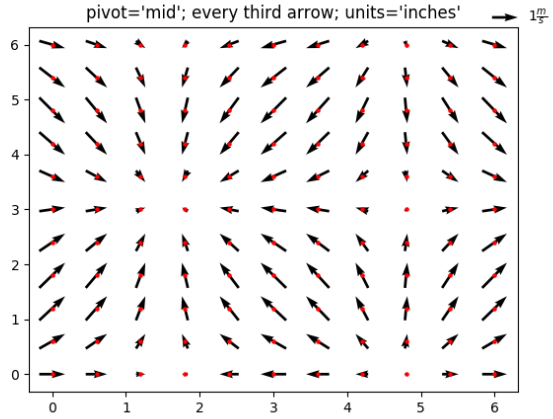


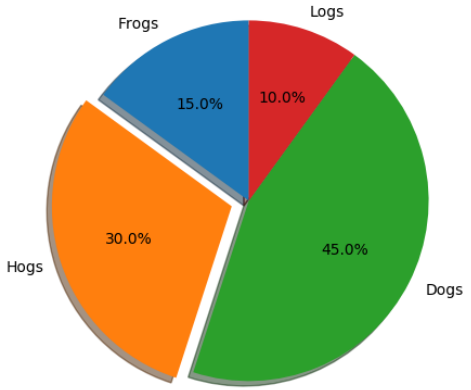












Matplotlib

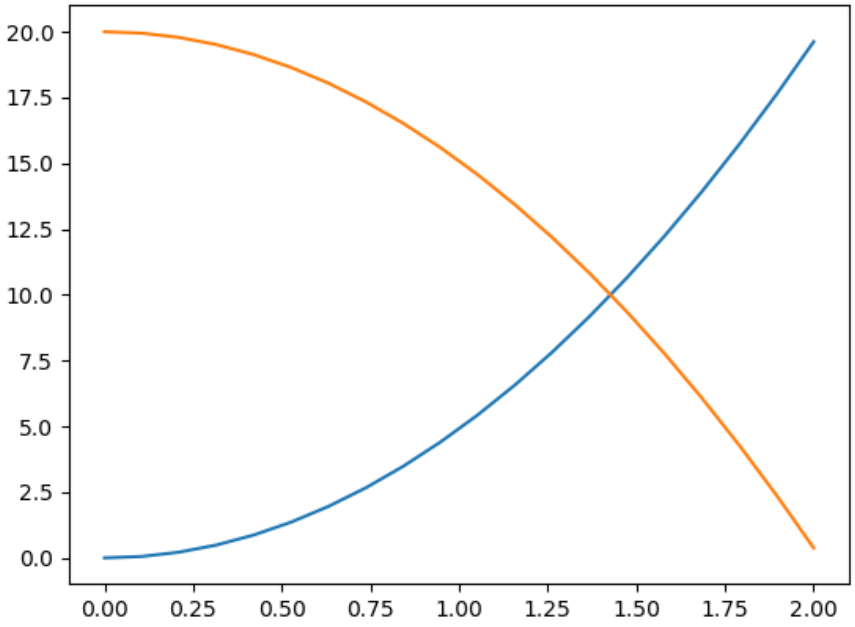
- **Matplotlib** is a Python **2D plotting library** which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in **Python scripts**, the **Python** and **IPython** shells, the **Jupyter notebook**, web application servers, and four graphical user interface toolkits.

- You can generate **plots**, **histograms**, **power spectra**, **bar charts**, **errorcharts**, **scatterplots**, etc., with just a few lines of code by using Matplotlib.

- **pyplot** provides a convenient interface to the matplotlib object-oriented plotting library. It is modeled closely after Matlab(TM). Therefore, the majority of plotting commands in pyplot have Matlab(TM) analogs with similar arguments. Important commands are explained with interactive examples.

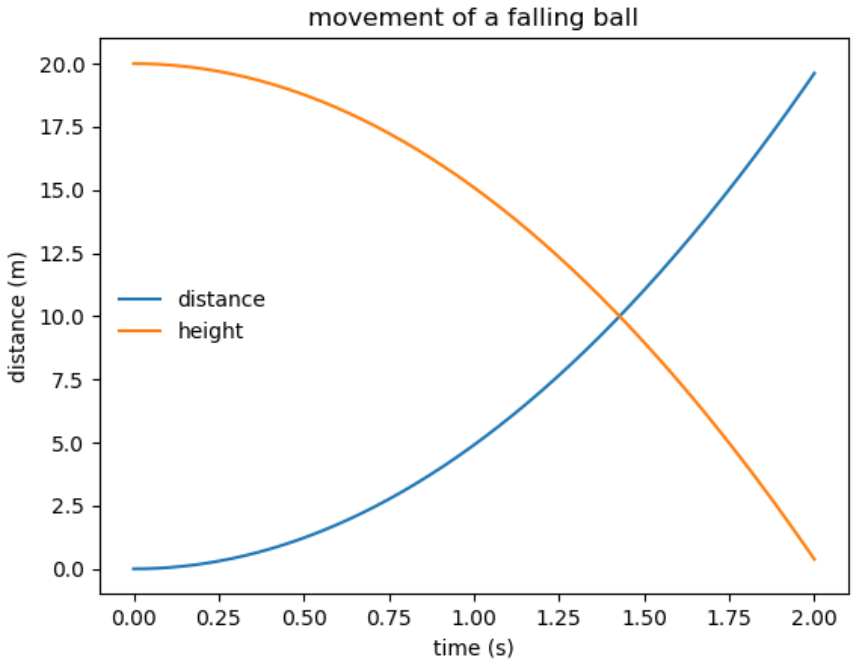
Simple plot

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

h0=10
t = np.linspace(0, 2, 1000, endpoint=True)
s = 9.81*t**2/2
h = h0-s

plt.plot(t,s)
plt.plot(t,h)

plt.show()
```



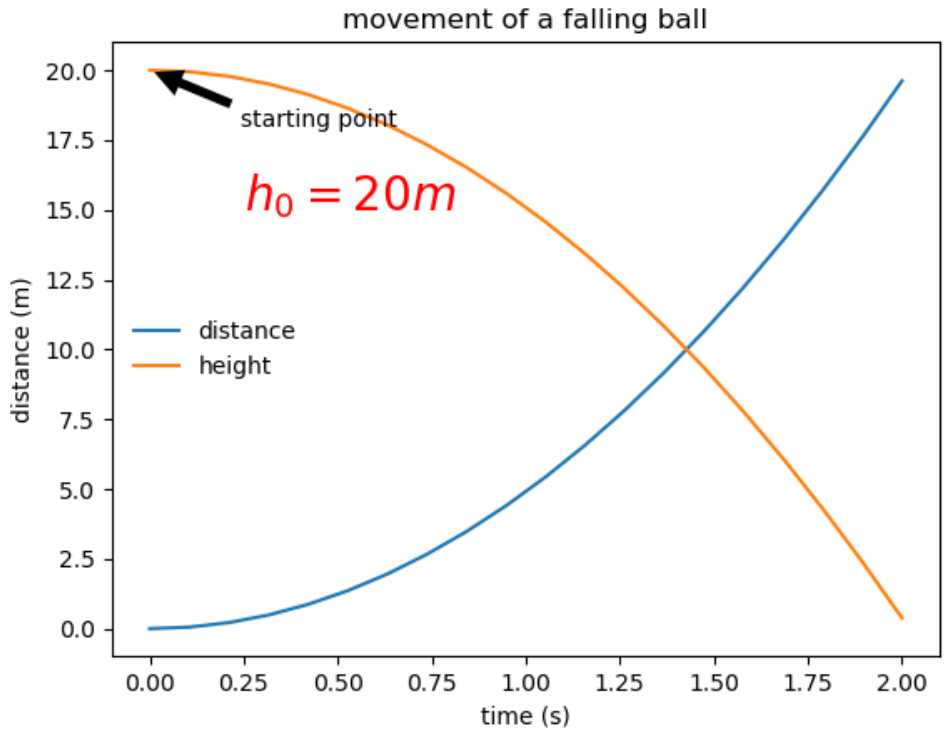
Add descriptions

```
# add the legend to describe different lines
plt.plot(t,s,label='distance')
plt.plot(t,h,label='height')
plt.legend(loc='center left', frameon=False)

#add title for the whole figure
plt.title('movement of a falling ball')

#add the description for x axis
plt.xlabel('time (s)')

#add the description for y axis
plt.ylabel('distance (m)')
```



Add more description

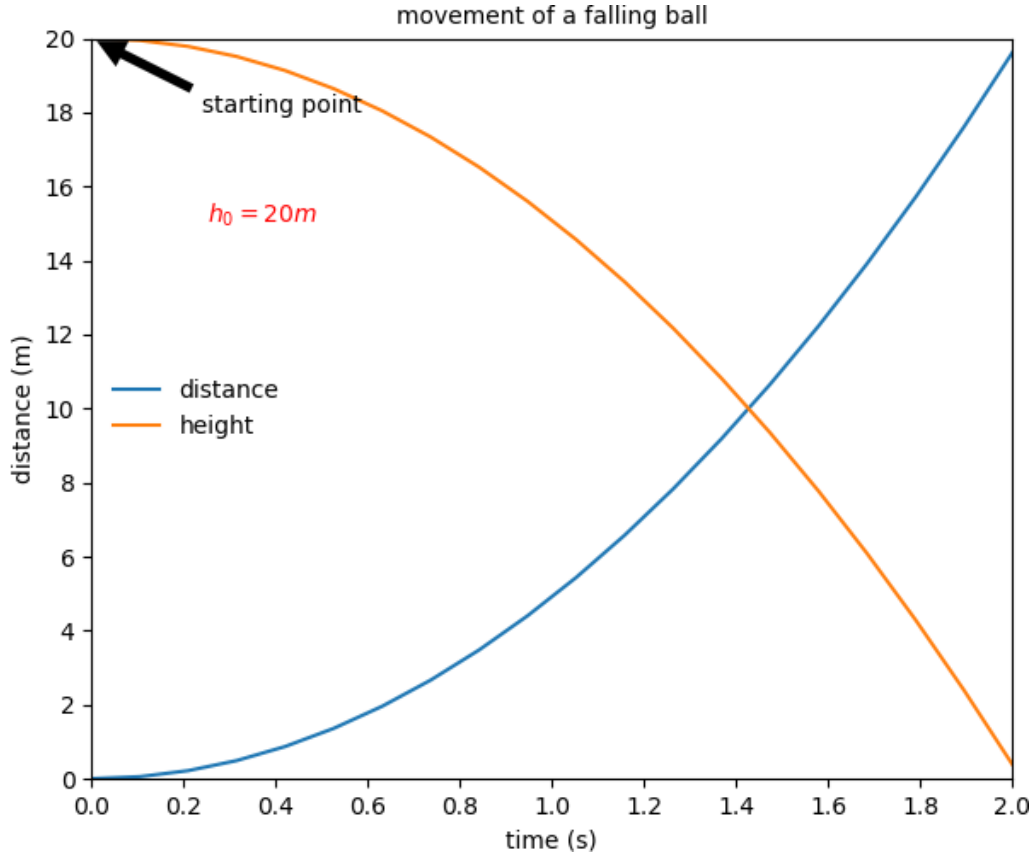
```
#add some text description in the figure
plt.text(0.25, 15, '$h_0=20 m $',color='red',fontsize=20)

#add some text description in the figure with an arrow
plt.annotate('starting point', xy=(0, 20), xytext=(0.24, 18),
    arrowprops=dict(facecolor='black', shrink=0.05,linewidth=0.05),
    )
```



Change the axis

Set x and y limits
plt.**xlim**(0,2.0); plt.**ylim**(0,20)

Set x and y ticks
plt.**xticks**(np.linspace(0,2,11,endpoint=True))
plt.**yticks**(np.linspace(0,20,11,endpoint=True))



Change the color

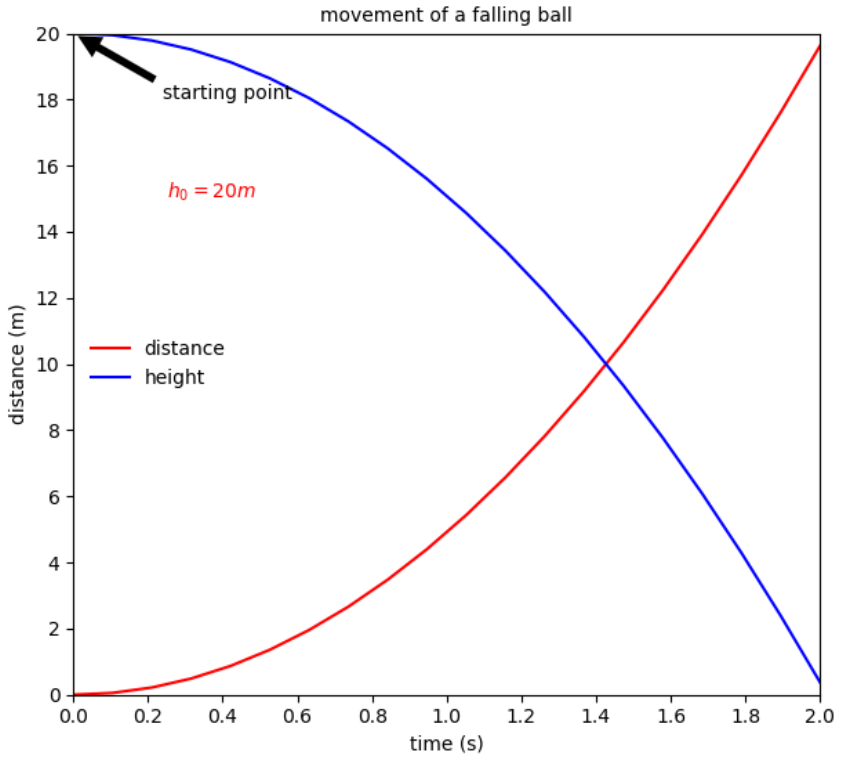
# Plot cosine using blue color with a continuous line of width 1 (pixels)
plt.plot(t, h, ***color***="blue", ***label***='height')

Commands taking color arguments can use several formats to specify colors

- For the basic build-in colors, you can use a single letter.

Alias	Color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

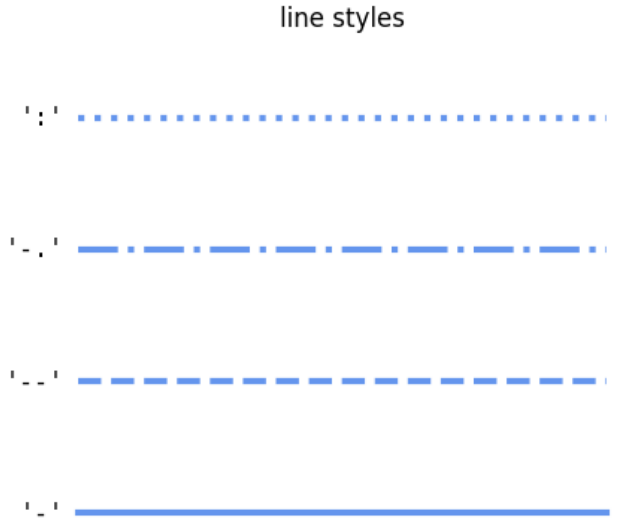
- You can specify the color using an html hex string, as color='#eeefff'
- You can pass an R, G, B tuple, which each of R, G, B are in the [0,1], as color=(0.1, 0.3, 0.5)

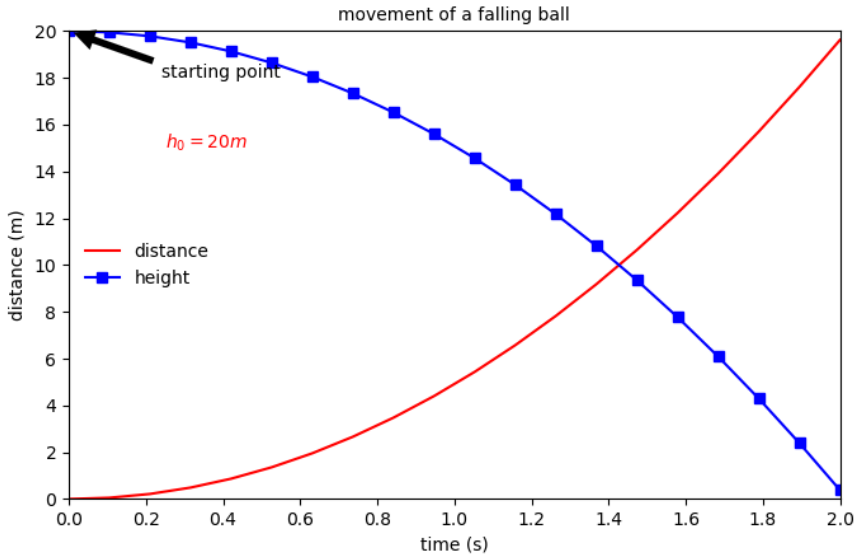


Change the line style and marker

# Plot cosine using blue color with a continuous line of width 1 (pixels)
plt.plot(t, h, ***color***="blue", ***linewidth***=1.0, ***linestyle***="-",***marker***='s',***label***='height')

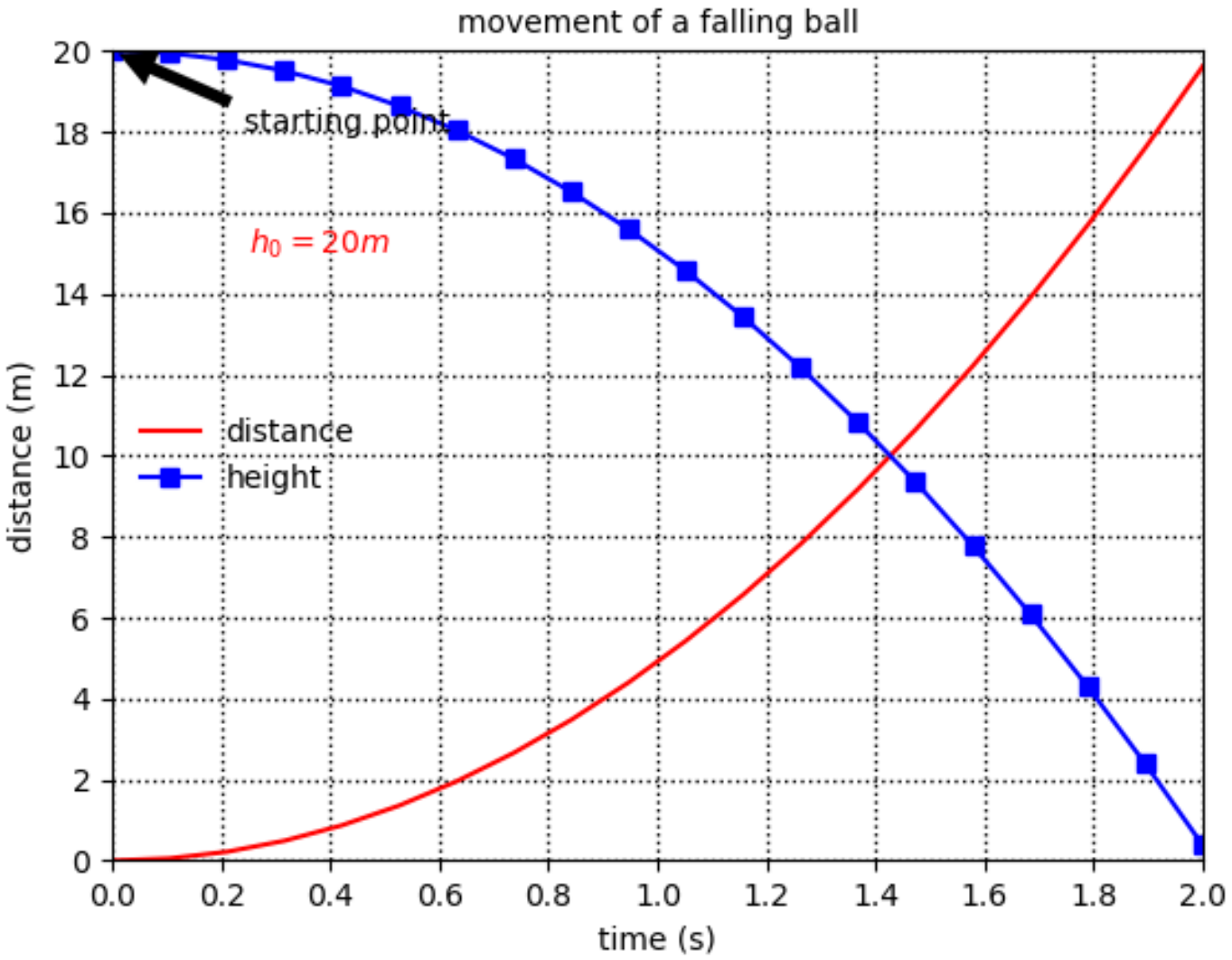
marker	symbol	description
"."	●	point
","	·	pixel
"o"	●	circle
"v"	▼	triangle_down
"^"	▲	triangle_up
"<"	◀	triangle_left
">"	▶	triangle_right
"1"	Y	tri_down
"2"	⅄	tri_up
"3"	≺	tri_left
"4"	≻	tri_right
"8"	●	octagon
"s"	■	square
"p"	⬟	pentagon
"P"	✚	plus (filled)
"*"	★	star
"h"	⬢	hexagon1
"H"	⬣	hexagon2





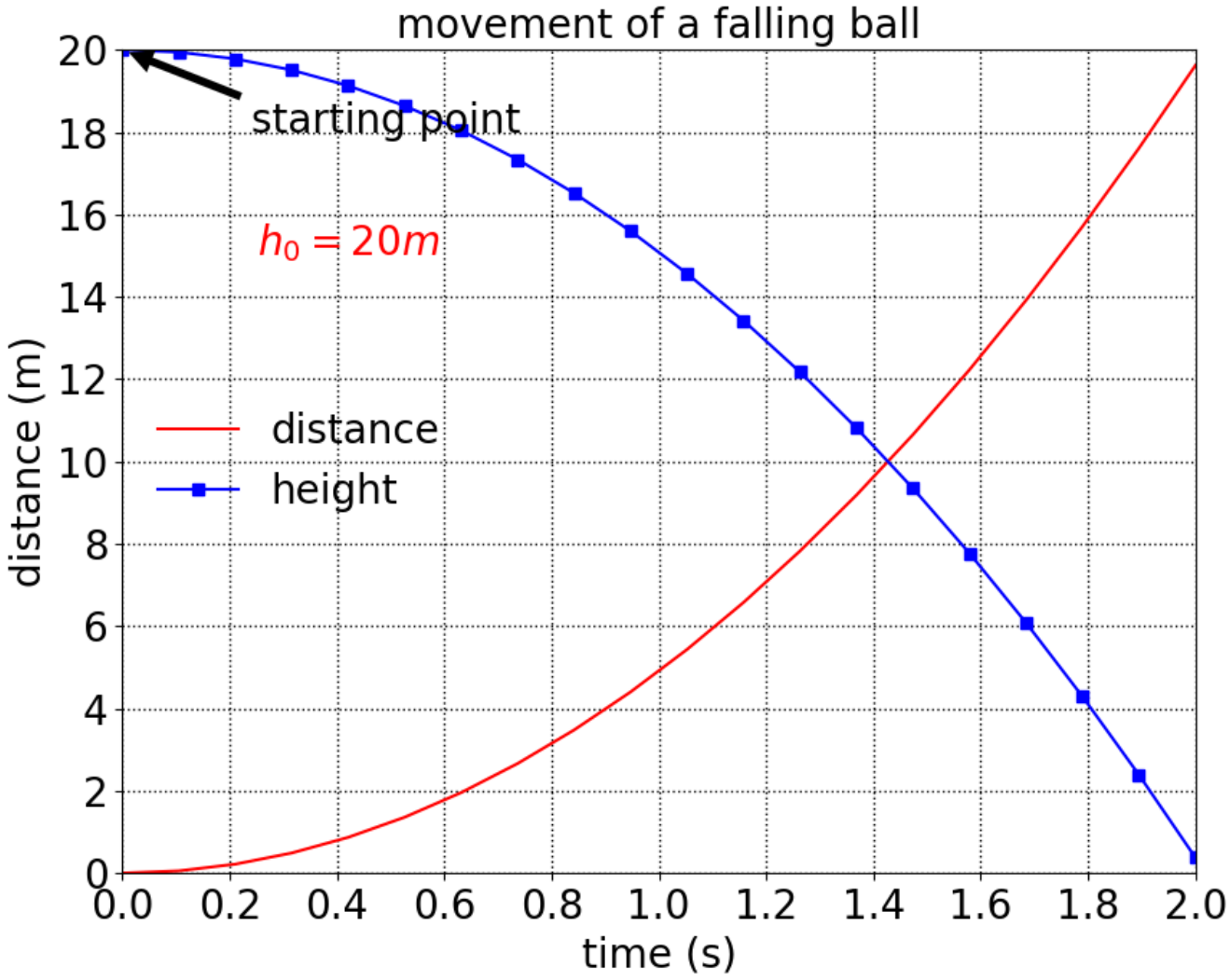
Add the grid to the figure

```
# Add the grid to
plt.grid(which='major',axis='both',color='k',linestyle=':',linewidth=1)
```

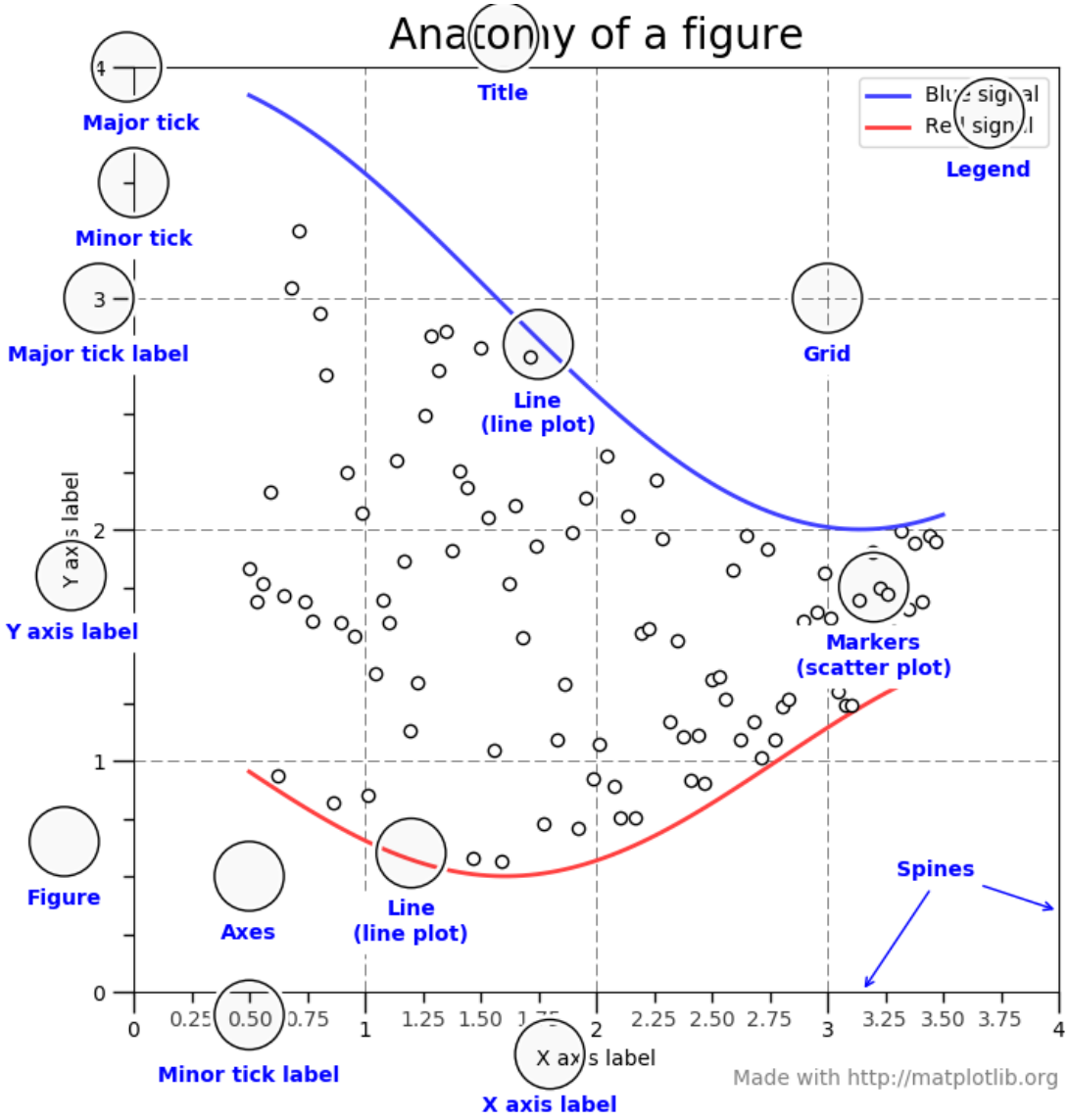


Final plot

```
# change the font size
fontsize=20
```



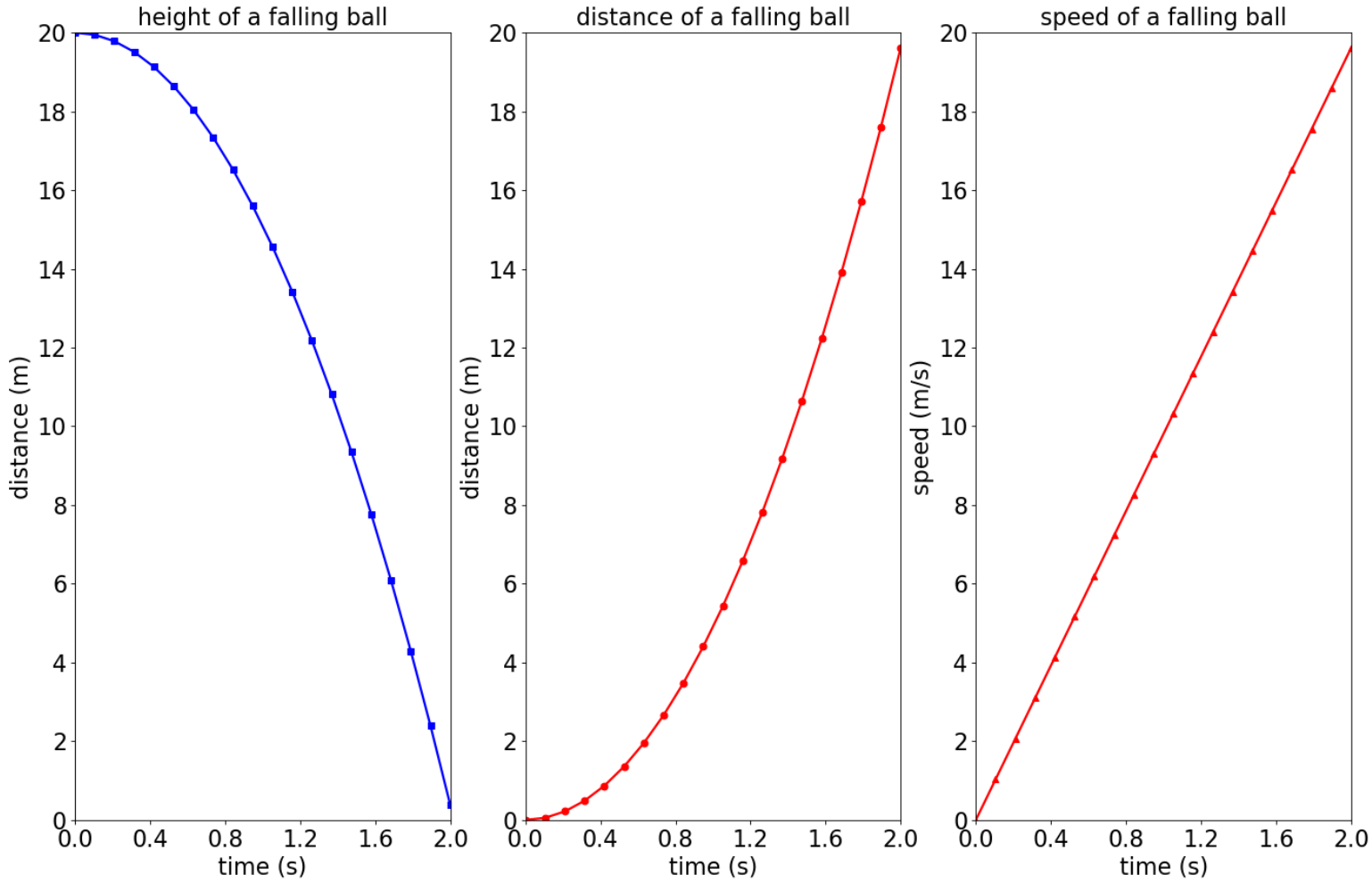
Part of a figure



- You can control/change everything in this figure
- Combine all the commands to make your figure more readable for others

subplot

# We can use subplot to create different panel in one figure
subplot(nrows, ncols, index, **kwargs)
https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html



control the ticks

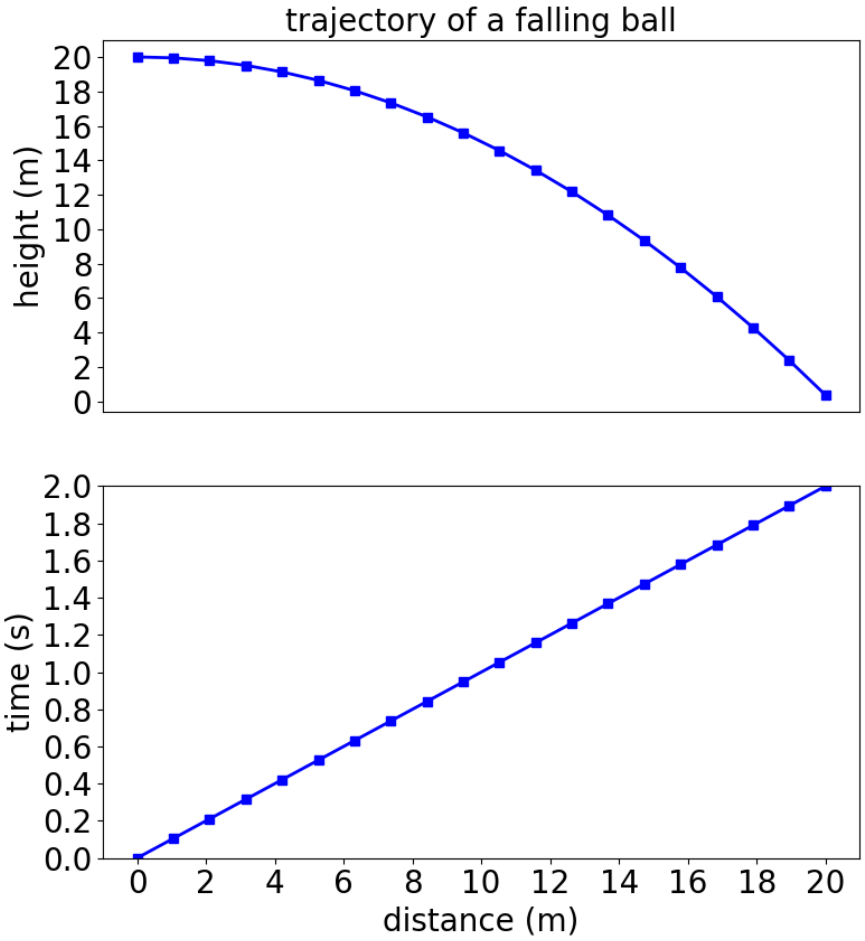
control the ticks, ***plt.tick_params()***

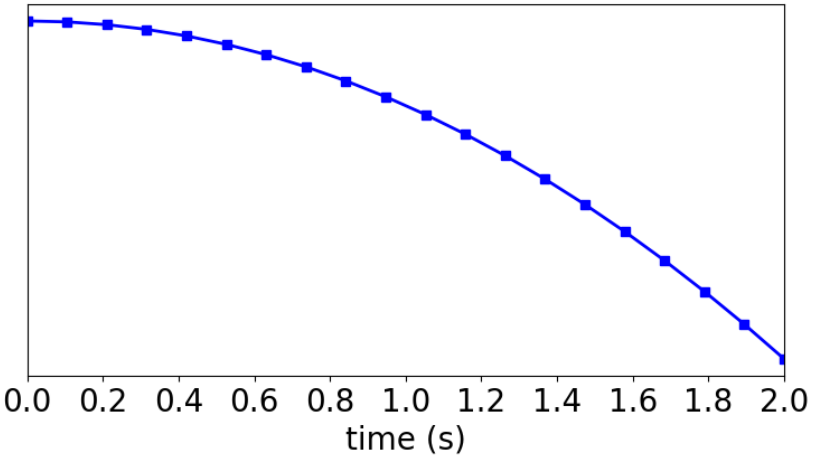
For example,

```
plt.tick_params(
    axis='x',           # changes apply to the x-axis
    which='both',       # both major and minor ticks are affected
    bottom=False,       # ticks along the bottom edge are off
    top=False,          # ticks along the top edge are off
    labelbottom=False)  # labels along the bottom edge are off
```

practice

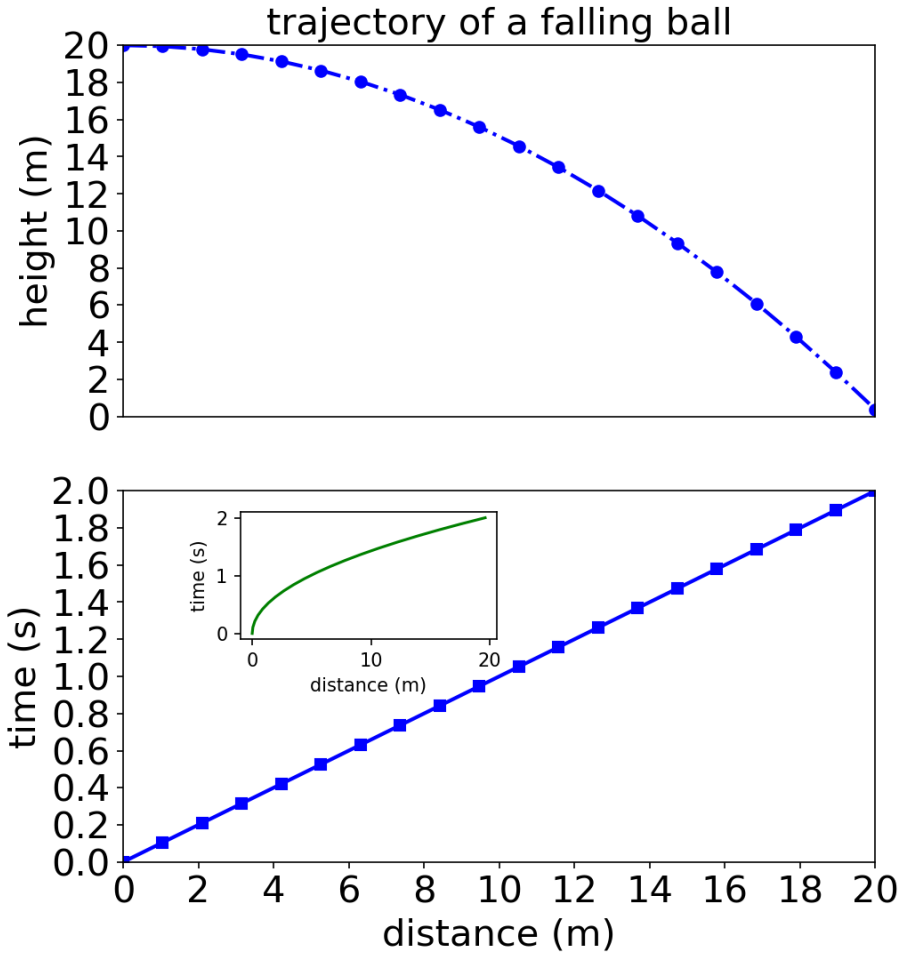
throw a ball with initial speed along x direction v0=10 at the height h=20, plot the movement of this ball from t=0 to t=2.

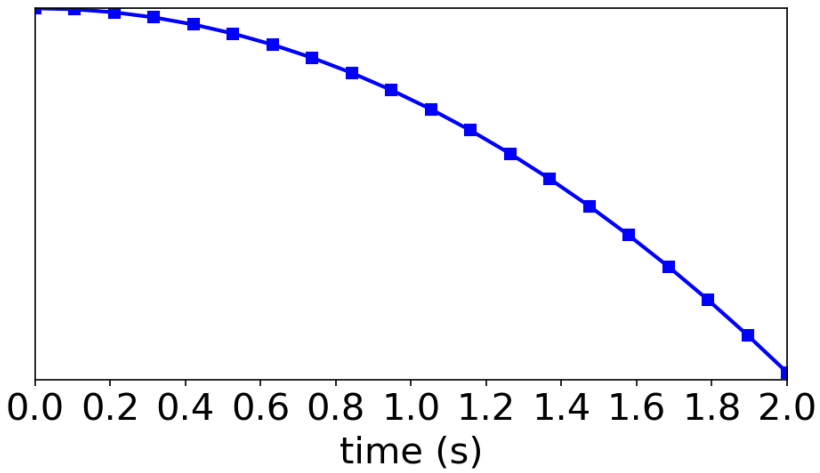





```
# the second subplot
ax2=plt.subplot(222, sharey=ax1)

plt.plot(t, h, color="blue", linewidth=2.0, linestyle="-",marker='s')

#set x axis
plt.xlabel('time (s)',fontsize=20)
plt.xlim(0,2)
plt.xticks(np.linspace(0,2,11,endpoint=True),fontsize=20)

#set y axis
plt.tick_params(
    axis='y',          # changes apply to the x-axis
    which='both',      # both major and minor ticks are affected
    left=False,      # ticks along the bottom edge are off
    right=False,         # ticks along the top edge are off
    labelleft=False) # labels along the bottom edge are off
```

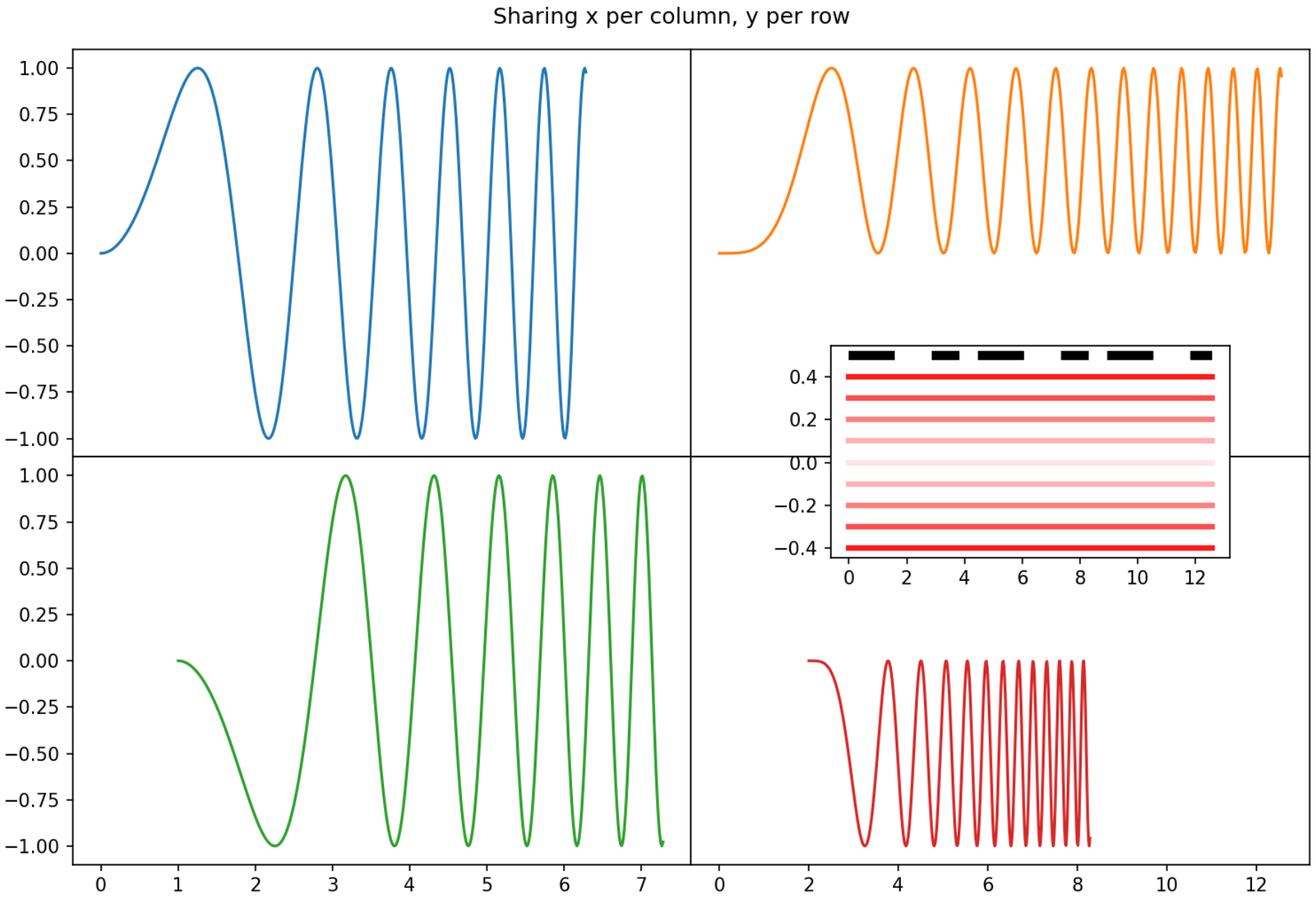
Add and control the axes by hands






```
# the fourth subplot

ax4=fig.add_axes([0.18,0.32,0.12,0.12])
ax4.plot(s,t,color='green')
plt.xlabel('distance (m)',fontsize=10)
plt.ylabel('time (s)',fontsize=10)
```

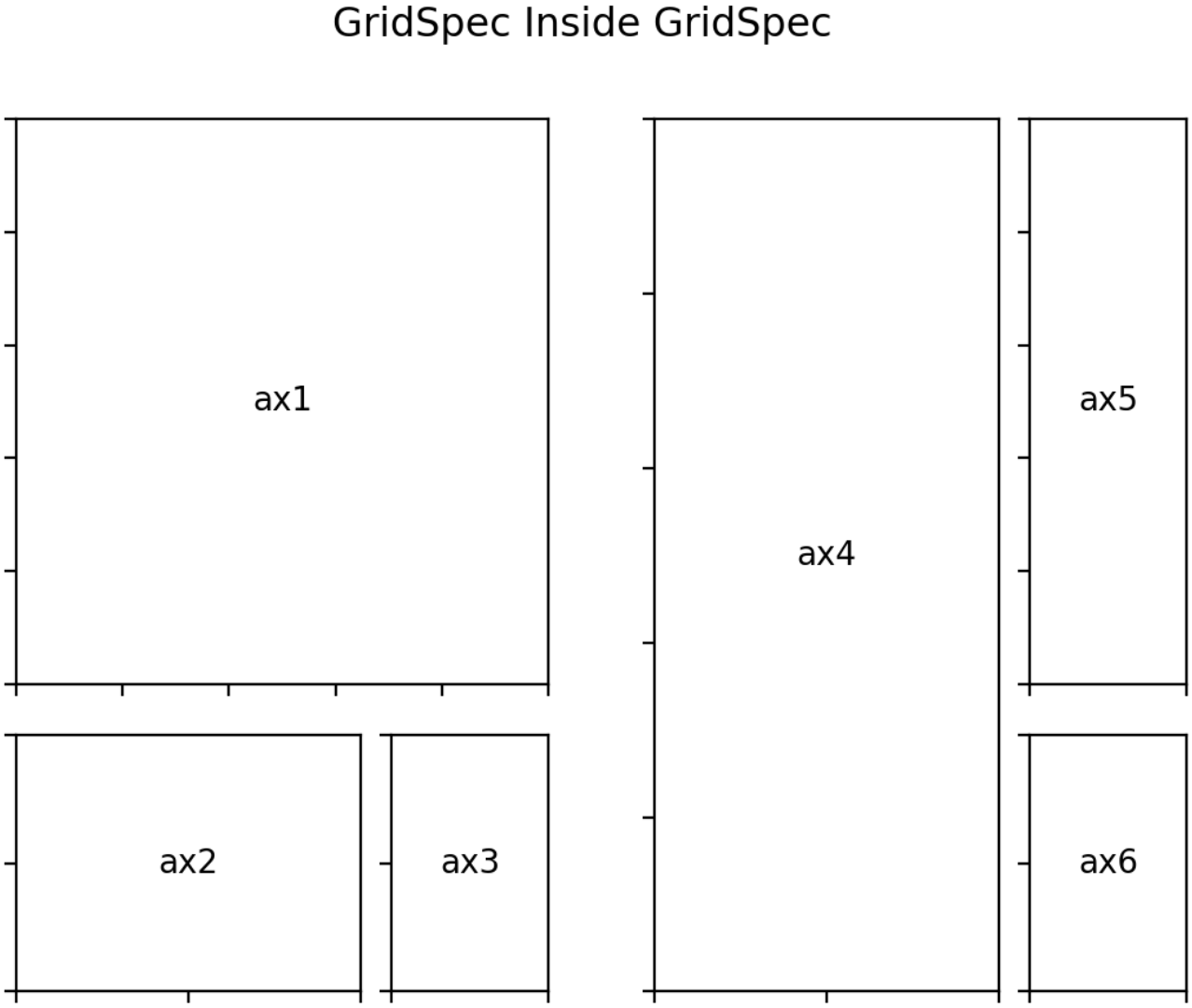
More advanced control using Gridspecs



The codes

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

# Some example data to display
x = np.linspace(0, 2 * np.pi, 400); y = np.sin(x ** 2)

fig = plt.figure(figsize=(12,8),dpi=150)
axs = fig.add_gridspec(2, 2, hspace=0, wspace=0)

ax1=plt.subplot(axs[0,0])
ax1.plot(x, y)

ax2=plt.subplot(axs[0,1],sharey=ax1)
ax2.plot(x*2,  y**2, 'tab:orange')    #Tableau Colors
plt.tick_params(axis='y', which='both',left=False,right=False, labelleft=False)

ax3=plt.subplot(axs[1,0],sharex=ax1)
ax3.plot(x+1,  -y, 'tab:green')

ax4=plt.subplot(axs[1,1],sharex=ax2,sharey=ax3)
ax4.plot(x + 2, -y**2, 'tab:red')
plt.tick_params(axis='y', which='both',left=False,right=False, labelleft=False)

ax5=fig.add_axes([0.6,0.4,0.25,0.2])

#the comma after ln is very important, please check the
#type of return values with and without comma
#https://stackoverflow.com/questions/16037494/x-is-this-trailing-comma-the-comma-operator
#ax.plot() returns a list with one element. By adding the comma to
#the assignment target list, you ask Python to unpack the return value
#and assign it to each variable named to the left in turn.
ln5,=ax5.plot(x*2, y*0+0.5, 'k')
ln5.set_linewidth(5)
ln5.set_dashes([5,4,3,2])

ln6=ax5.plot(x*2, y*0+0.4, 'r',lw=3)[0]  ## taking the first element of return of plot()
ln6.set_alpha(0.9) # set the transparency
for nshift in [-0.3, -0.2, -0.1, 0, 0.1,0.2,0.3, 0.4]:
    ax5.plot(x*2, y*0-nshift, 'r',lw=3,alpha=abs(nshift*2)+0.1)

#Add a centered suptitle to the figure.
fig.suptitle('Sharing x per column, y per row',x=0.5,y=0.92)
```

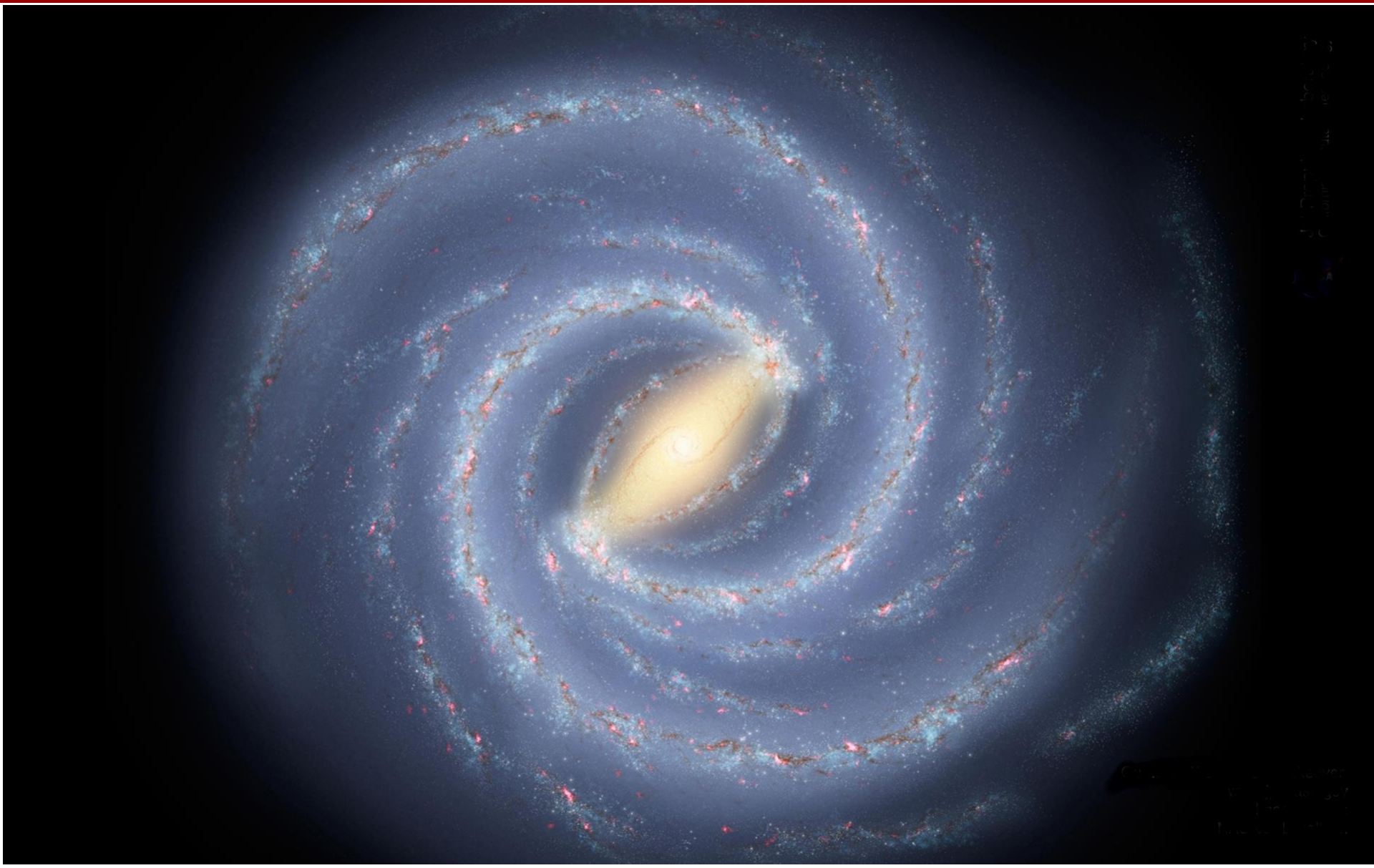
Nested Gridspecs



The codes

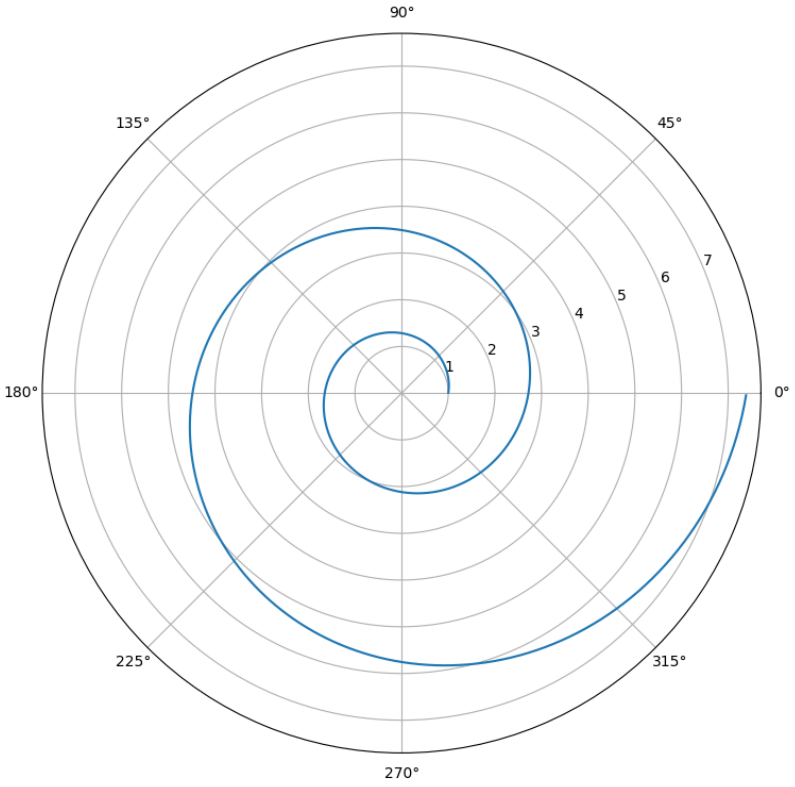
```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec


def format_axes(fig):
    for i, ax in enumerate(fig.axes):
        ax.text(0.5, 0.5, "ax%d" % (i+1), va="center", ha="center")
        ax.tick_params(labelbottom=False, labelleft=False)


# gridspec inside gridspec
fig = plt.figure()

gs0 = gridspec.GridSpec(1, 2, figure=fig)

gs00 = gridspec.GridSpecFromSubplotSpec(3, 3, subplot_spec=gs0[0])

ax1 = fig.add_subplot(gs00[:-1, :])
ax2 = fig.add_subplot(gs00[-1, :-1])
ax3 = fig.add_subplot(gs00[-1, -1])

# the following syntax does the same as the GridSpecFromSubplotSpec call above:
gs01 = gs0[1].subgridspec(3, 3)

ax4 = fig.add_subplot(gs01[:, :-1])
ax5 = fig.add_subplot(gs01[:-1, -1])
ax6 = fig.add_subplot(gs01[-1, -1])

plt.suptitle("GridSpec Inside GridSpec")
format_axes(fig)

plt.show()
```

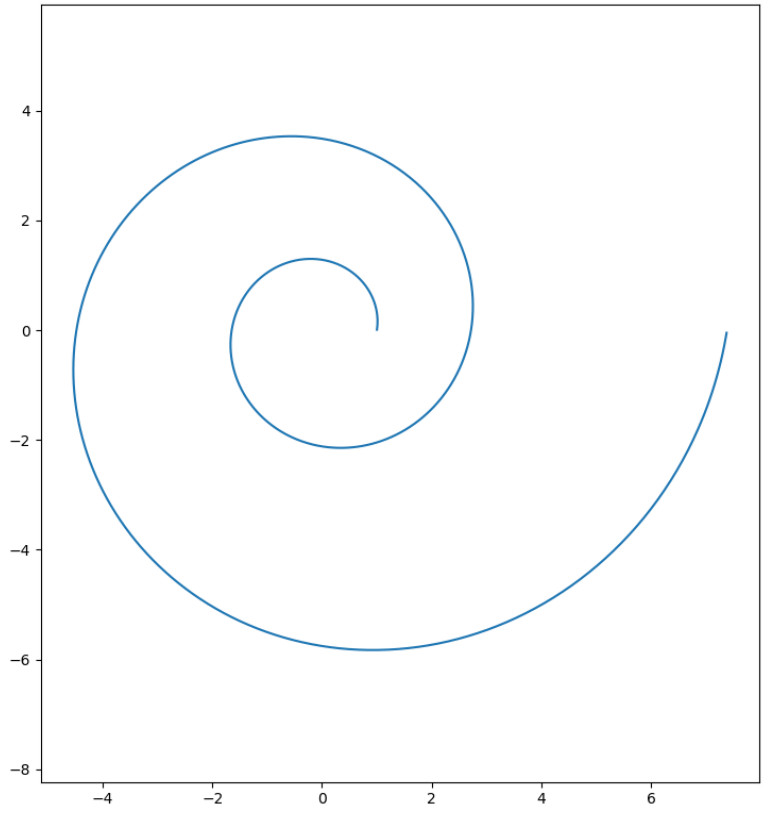
Galaxy



Polar coordinates

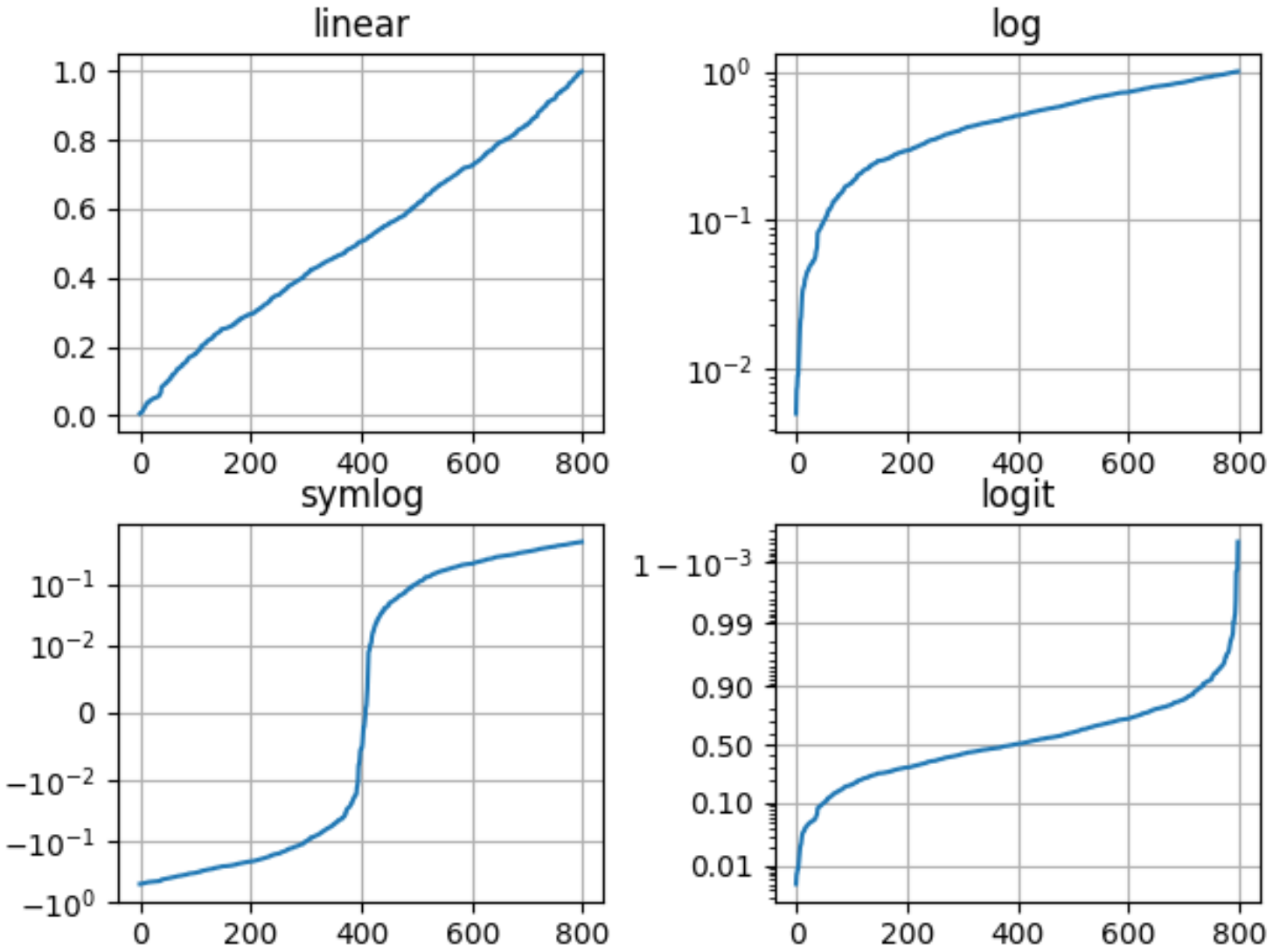
plt.subplot(121,projection='polar')
plt.subplot(121,polar=True)





Logarithmic and other nonlinear axes

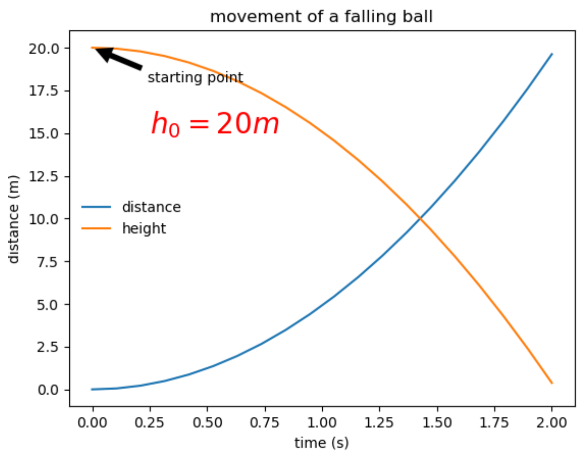
plt.yscale('linear'); plt.yscale('log'); plt.yscale('symlog'); plt.yscale('logit')

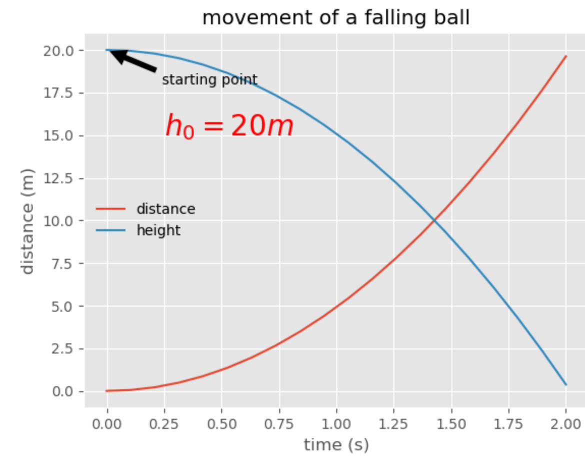


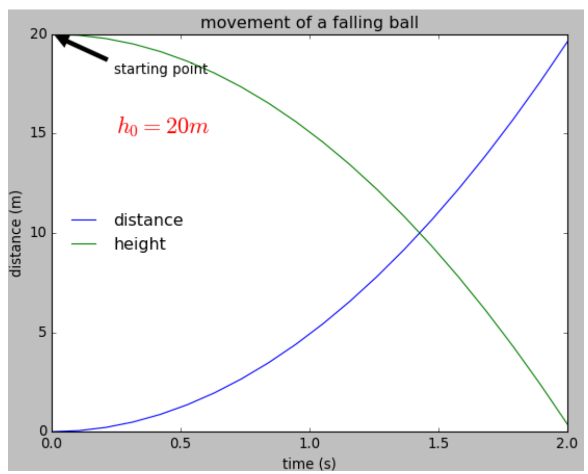
Styles of plots

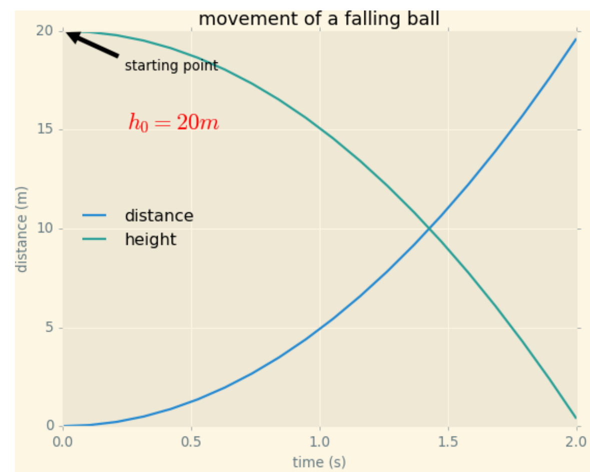
from matplotlib import style
style.use('ggplot')

```
In [10]: plt.style.available
Out[10]:
['bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark-palette',
 'seaborn-dark',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
 'seaborn-notebook',
 'seaborn-paper',
 'seaborn-pastel',
 'seaborn-poster',
 'seaborn-talk',
 'seaborn-ticks',
 'seaborn-white',
 'seaborn-whitegrid',
 'seaborn',
 'Solarize_Light2',
 'tableau-colorblind10',
 '_classic_test']
```









Save the figure

plt.savefig("XXX.pdf");
fig1.savefig("XXX.pdf")

In addition to the basic functionality of saving the chart to a file, .savefig() also has a number of useful optional arguments.

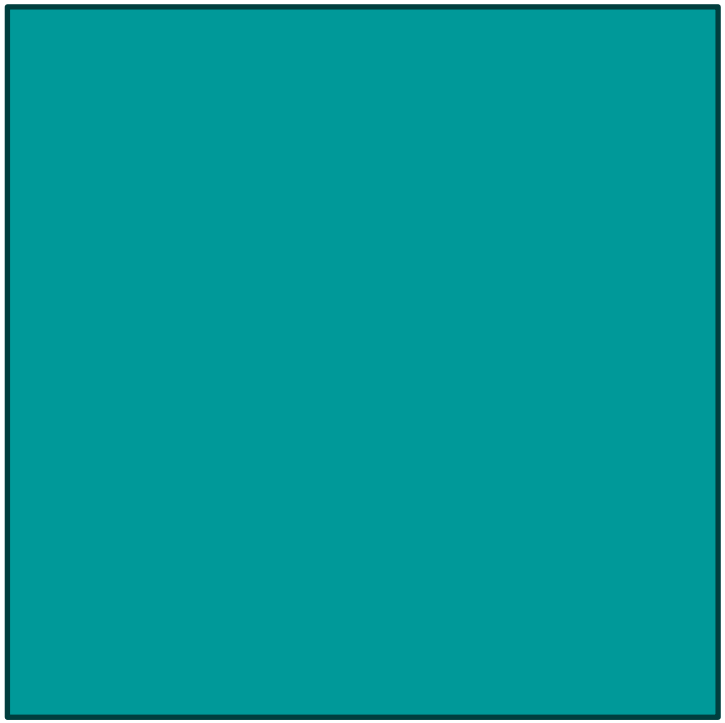
dpi can be used to set the resolution of the file to a numeric value.

transparent can be set to True, which causes the background of the chart to be transparent.

bbox_inches can be set to alter the size of the bounding box (whitespace) around the output image. In most cases, if no bounding box is desired, using **bbox_inches='tight'** is ideal.

If **bbox_inches** is set to **'tight'**, then the **pad_inches** option specifies the amount of padding around the image.

Prepare for next quiz



How to plot an exact square or circle?