Save the figure

- In Python, save the figure is very simple

- savefig(fname, dpi=None, facecolor='w', edgecolor='w',
  orientation='portrait', papertype=None, format=None,
  transparent=False, bbox_inches=None, pad_inches=0.1,
  frameon=None, metadata=None)

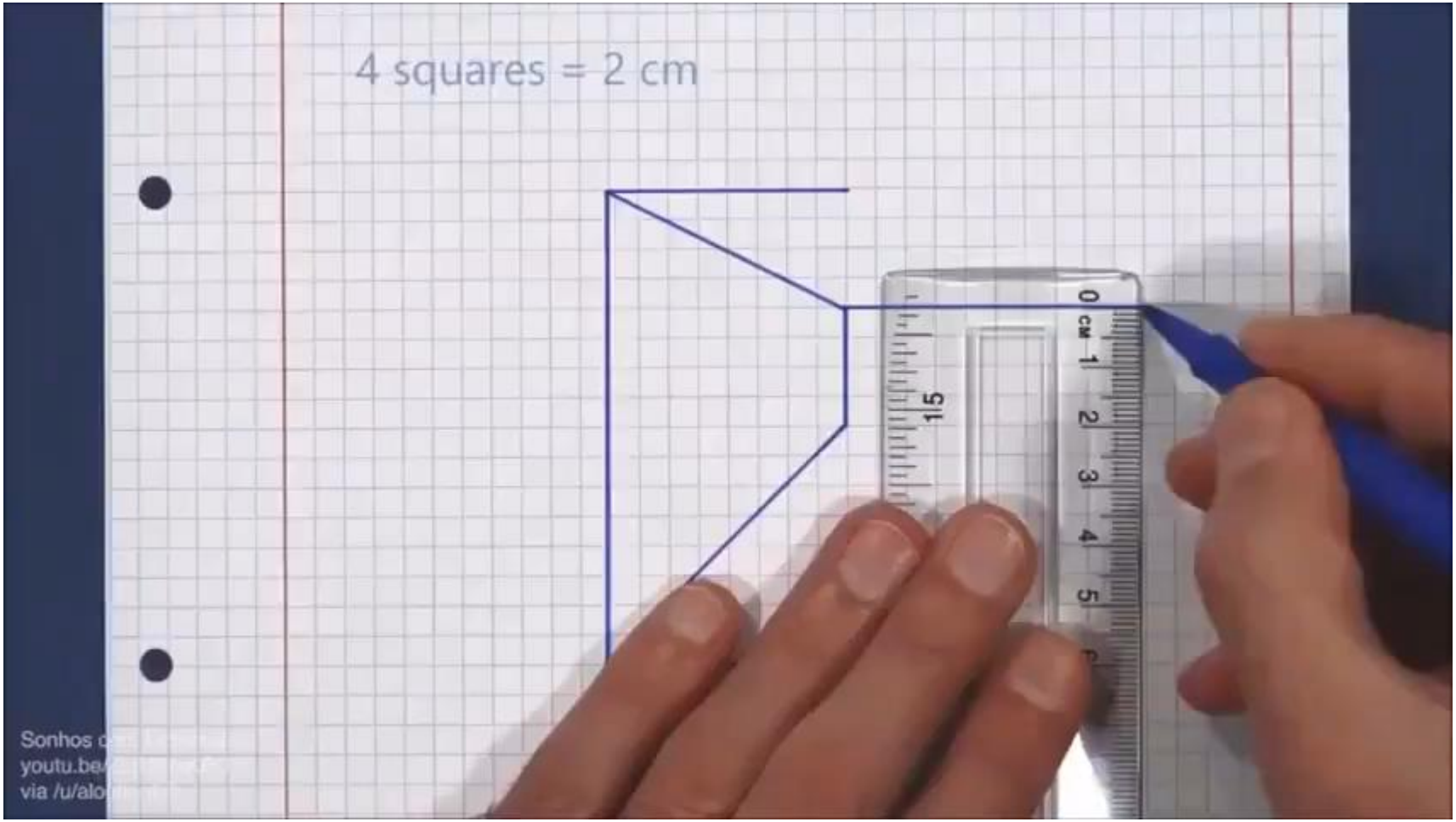
https://matplotlib.org/api/_as_gen/matplotlib.pyplot.savefig.html

- For exmpale:

savefig('foo.png') # save in png format, Raster graphics
savefig('foo.pdf') # save in pdf format, vector graphics

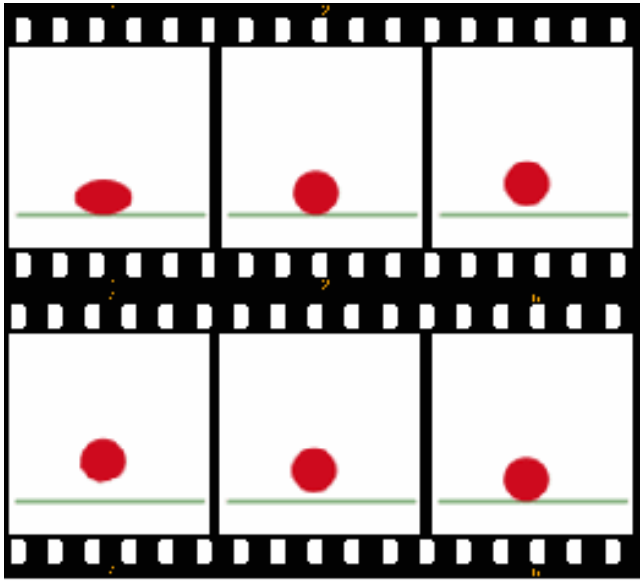
- One can also directly save the matrix into a figure as
  plt.imsave(‘foo.png’)
  plt.imsave(‘foo.pdf’)

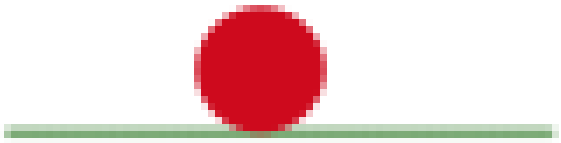
Illusion due to visualization



Animation

- What is animation?
- Animation is just a dynamic medium in which images are manipulated to appear as moving images. NOT real movements!!!
- You can easily make an animation by representing the images one by one with some pauses in certain intervals.





An old example

Frame Rate

- An animation or a video is a series of still images that, when viewed in order at a certain speed, give the appearance of motion. Each of those images is called a “frame.”

- Frame rate, then, is the speed at which those images are shown, or how fast you “flip” through the book. It’s usually expressed as “frames per second,” or FPS. So if a video is captured and played back at 24fps, that means each second of video shows 24 distinct still images.

- The speed at which they’re shown tricks your brain into perceiving smooth motion.

Typical Frame Rate

- 24fps –This is the standard for movies and TV shows, and it is the minimum speed to capture video while still maintaining realistic motion. Even if a film is shot at a higher frame rate, it's often produced and displayed at 24fps. Most feature films and TV shows are shot and viewed at 24 fps.

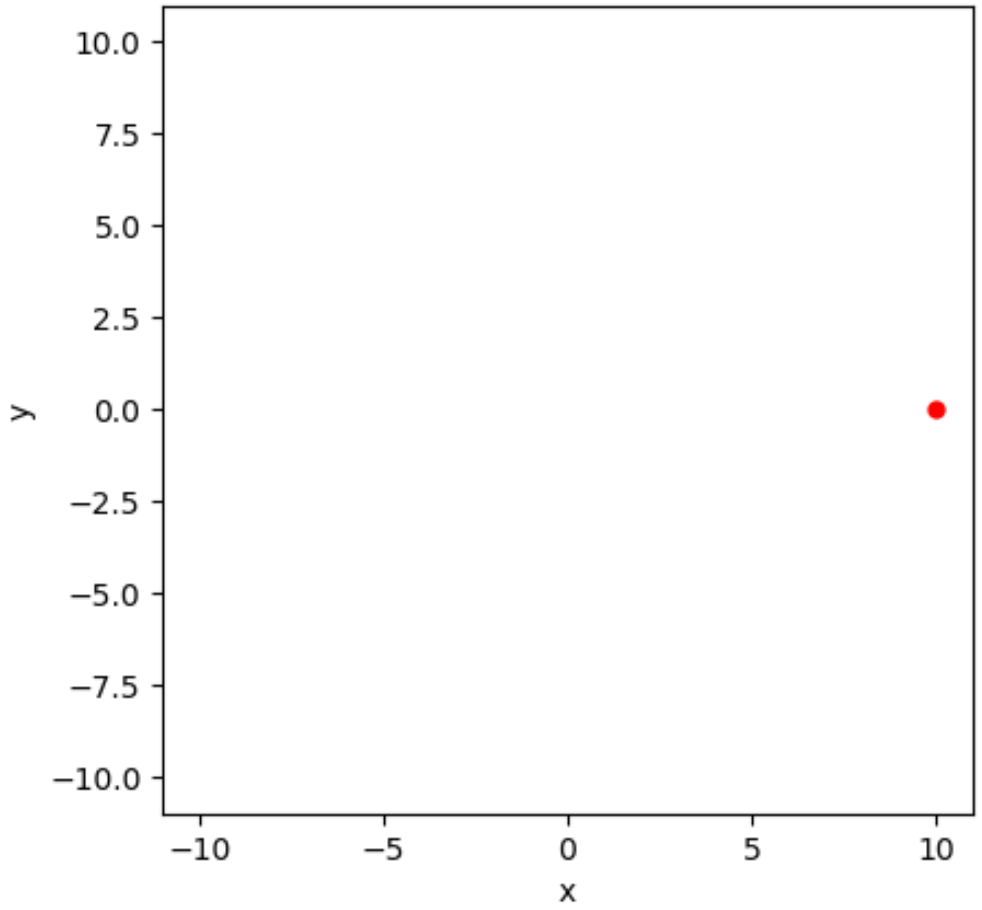
- 30fps –This has been the standard for television since the early days, and is still widely used. Videos with a lot of motion, such as sports, will often benefit from the extra frames per second. The reason for using 30fps is complicated and it mainly has to do with television and electricity standards set a long time ago.

- 60+fps – Anything higher than 30fps is mainly used to create slow-motion video or to record video game footage. Additionally, as technology continues to evolve, many smartphones are now capable of recording at 60 fps as well.

Difference between video games and movies

Simple Animation

- In Python, we can easily to realize animation by using **pyplot.pause(interval)**

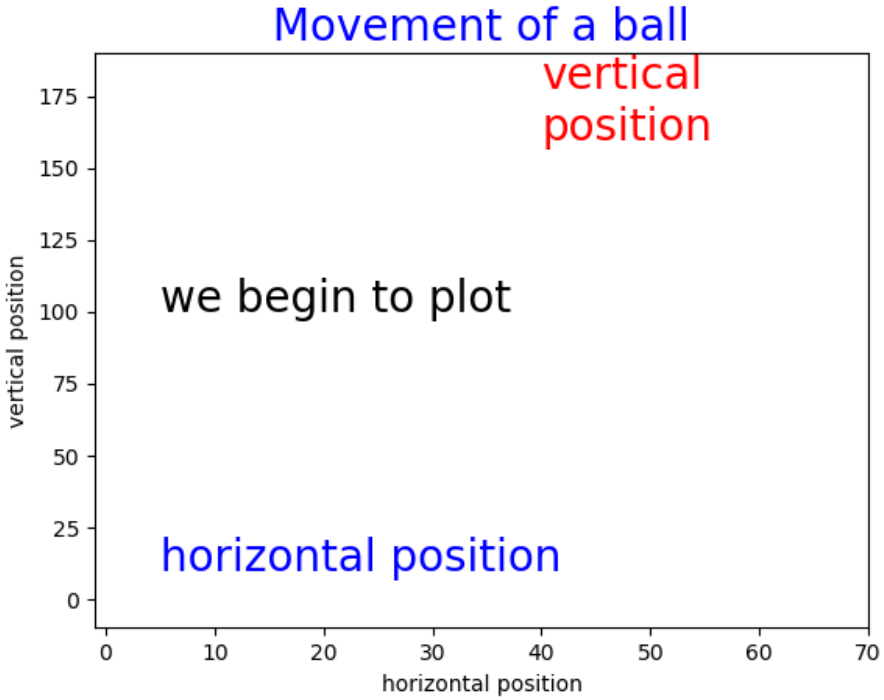


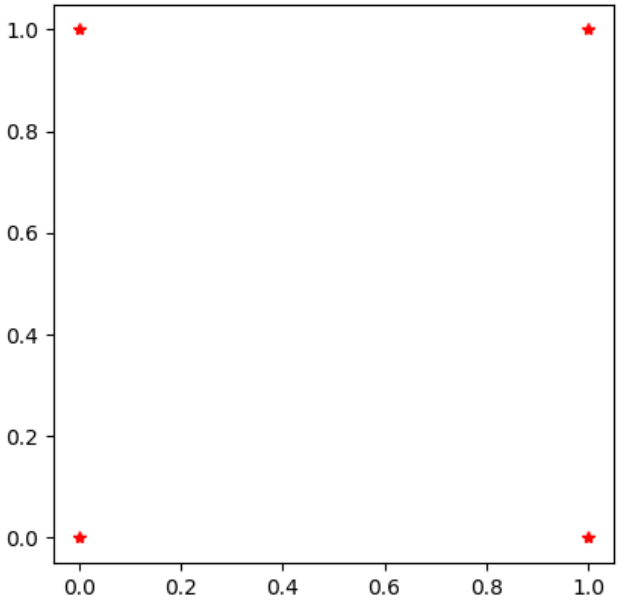
Save gif figure

- We can generate many figures and save them in gif format to represent animation by using the following the codes.

```
import imageio

def create_gif(image_list, gif_name, dura):
    # Save them as frames into a gif
    all_images = []
    for image_name in image_list:
        all_images.append(imageio.imread(image_name))
        imageio.mimsave(gif_name, all_images, 'GIF', duration = dura)
    return
```

Practice- moving ball





True animation

- In the previous example, it is not a real animation because there will be many balls eventually in a single shot. However, in real animation, there should be one ball moving around.
- We can use ***FuncAnimation*** and ***ArtistAnimation*** to make true animations in Python.
- **FuncAnimation** Makes an animation by repeatedly calling a function func.
- **ArtistAnimation** Animation using a fixed set of Artist objects.
- It is critical to keep a reference to the instance object. The animation is advanced by a timer which the Animation object holds the only reference to. If you do not hold a reference to the Animation object, it (and hence the timers), will be garbage collected which will stop the animation.
- To save an animation to disk use **Animation.save** or **Animation.to_html5_video**

matplotlib.animation.FuncAnimation()

- **FuncAnimation**(fig, func=update, init_func=None, frames=None, fargs=None, save_count=None, *, cache_frame_data=True, **kwargs)

fig: the figure object used to get needed events

func: The function to call at each frame. The first argument will be the next value in frames. Any additional positional arguments can be supplied via the fargs parameter.

init_func: A function used to draw a clear frame. If not given, the results of drawing from the first item in the frames sequence will be used. This function will be called once before the first frame.
If **blit** == True, init_func must return an iterable of artists to be re-drawn. This information is used by the blitting algorithm to determine which parts of the figure have to be updated. The return value is unused if **blit** == False and may be omitted in that case.

Parameter: frames and fargs

frames*: iterable, int, generator function, or None, optional*
Source of data to pass func and each frame of the animation

- If an iterable, then simply use the values provided
- If the iterable has a length, it will override the save_count kwarg.
- If an integer, then equivalent to passing ***range(frames)***
- If a generator function, then must have the signature: def gen_function() -> obj
- If None, then equivalent to passing itertools.count.

In all of these cases, the values in frames is simply passed through to the user-supplied func and thus can be of any type.

fargs: Additional arguments to pass to each call to func.

interval: int, default: 200. Delay between frames in milliseconds.

Other parameters

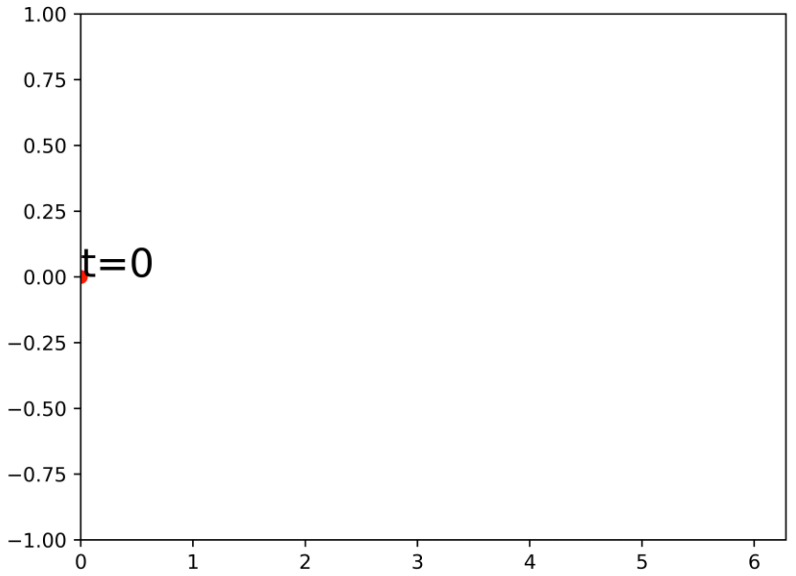
repeat_delay: int, default: 0
The delay in milliseconds between consecutive animation runs, if repeat is True.

repeat: bool, default: True
Whether the animation repeats when the sequence of frames is completed.

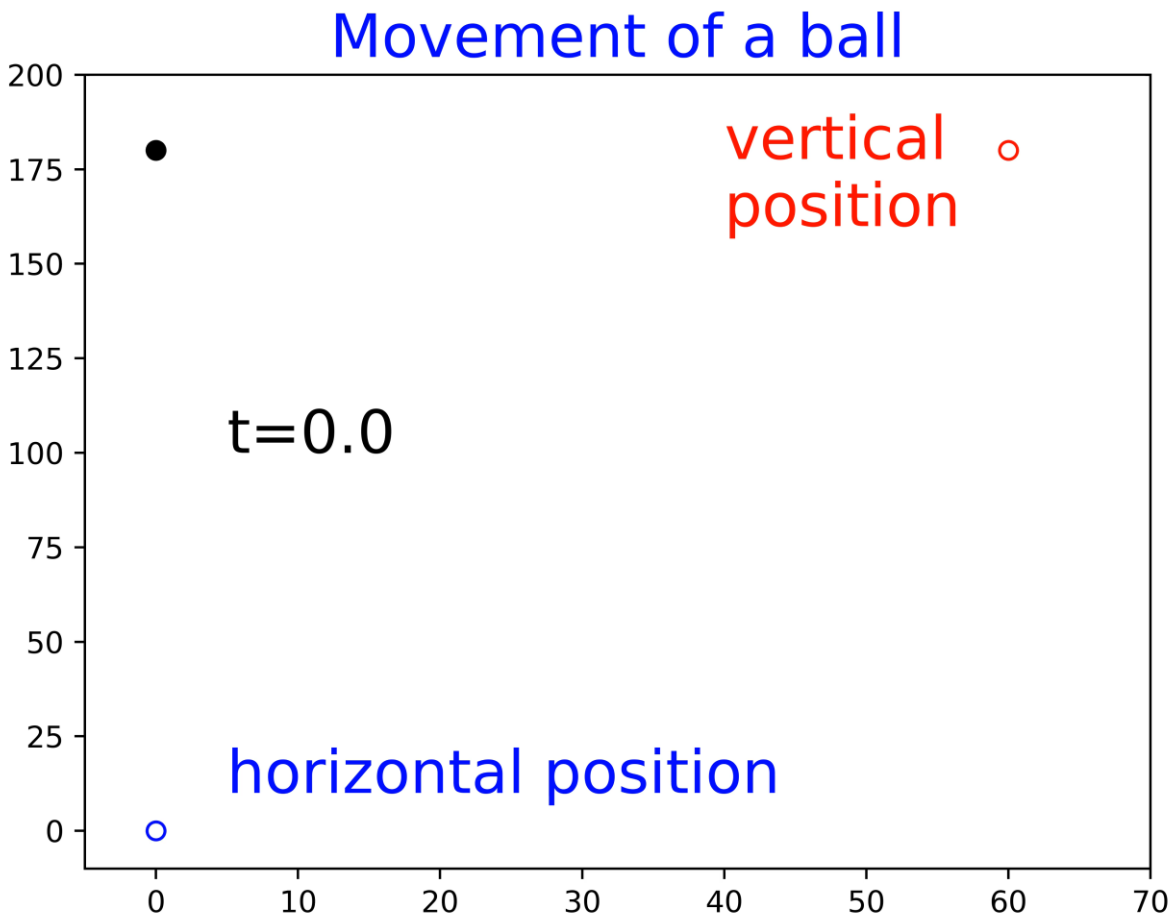
blit: bool, default: False
Whether blitting is used to optimize drawing. Note: when using blitting, any animated artists will be drawn according to their zorder; however, they will be drawn on top of any previous artists, regardless of their zorder.

Example by using FuncAnimation

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

fig_example, ax = plt.subplots()
#prepare something you want to update
ln, = plt.plot([], [], 'ro')
text_step=plt.text(0, 0, 't',color='black',fontsize=20)
#the initializaion function
def init():
    ax.set_xlim(0, 2*np.pi);     ax.set_ylim(-1, 1)
    return ln,text_step
#The function to update what you want to plot
def update(t):
    ln.set_data(t*0.02, np.sin(t*0.02))
    text_step.set_text('t='+str(t))
    return ln,text_step
ani = FuncAnimation(fig=fig_example, #which figure you want to update
                    init_func=init,  #the inition function
                    func=update, #the update function
                    frames=314, #frames
                    interval=314, # interval in total
                    blit=True #only update the plot that changed
                    )
```

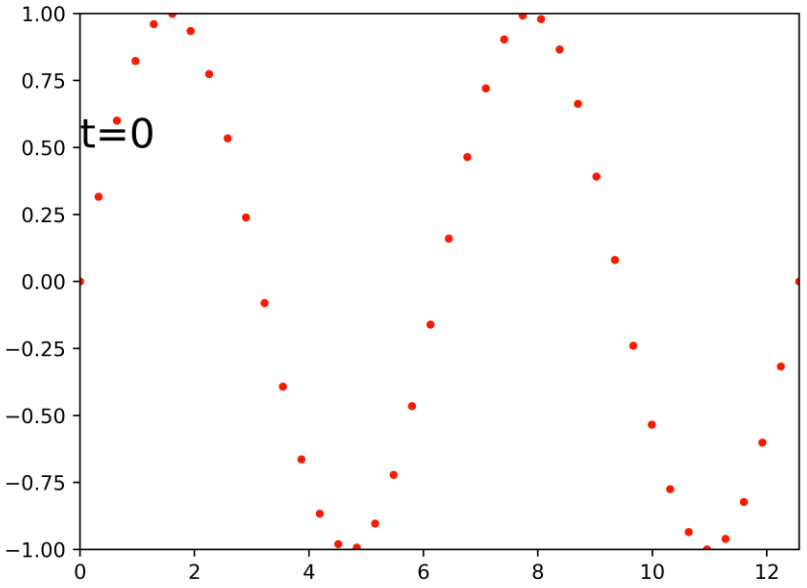


Practice- moving ball



Update the whole line

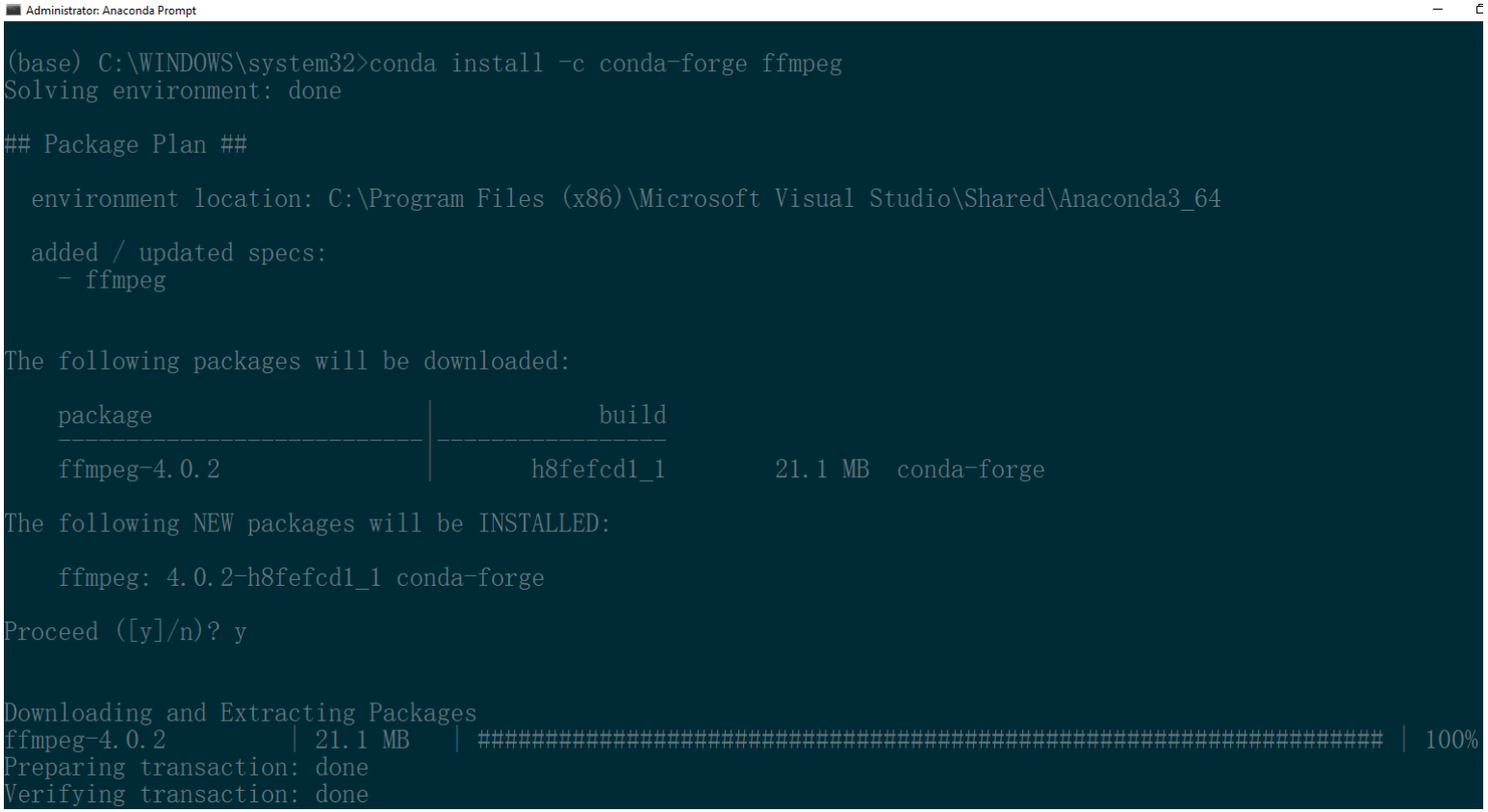
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

fig_example, ax = plt.subplots()
#prepare something you want to update
ln, = plt.plot([], [], 'r.')
text_step=plt.text(0, 0.5, 't',color='black',fontsize=20)
#the initializaion function
def init():
    ax.set_xlim(0, 4*np.pi);     ax.set_ylim(-1, 1)
    return ln,text_step
#The function to update what you want to plot
def update(t):
    x = np.linspace(0, np.pi*4, 40)
    y = np.sin(x - 0.01 * t*np.pi)
    ln.set_data(x, y)
    text_step.set_text('t='+str(t))
    return ln,text_step

ani = FuncAnimation(fig=fig_example,
                    init_func=init,
                    func=update,
                    frames=100,  interval=100, blit=True)
```



Save animation

- To save animation to mp4 etc, you have to first install install the **ffmpeg** or other packages.
- Different systems have different ways to install them, in Anaconda, you can easily install it in Anaconad prompt

conda install -c conda-forge ffmpeg



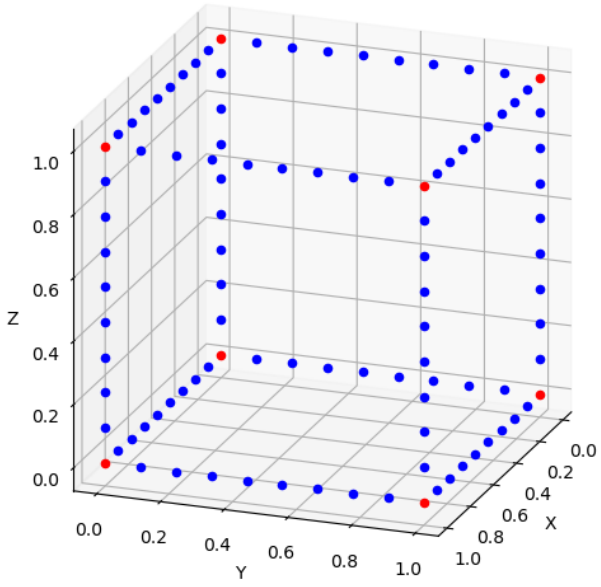
matplotlib.animation.Animation.save()

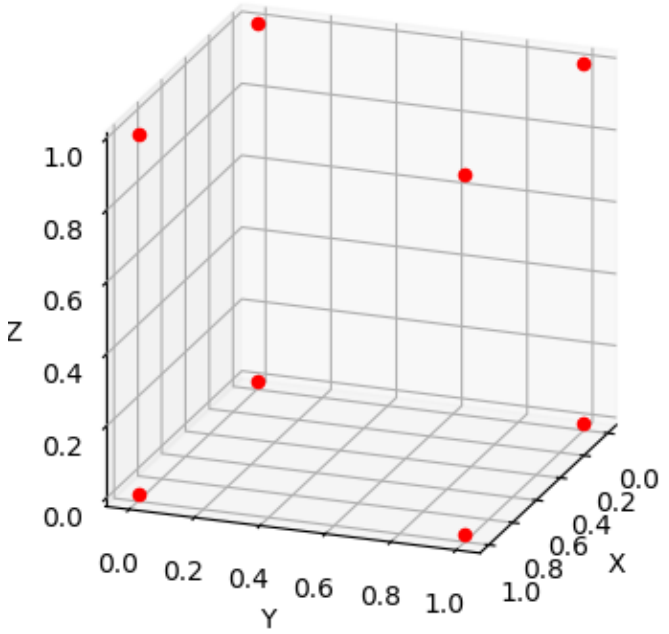
Animation.***save***(filename, writer=None, fps=None, dpi=None, codec=None, bitrate=None, extra_args=None, metadata=None, extra_anim=None, savefig_kwargs=None)

filename : str
The output filename, e.g., mymovie.mp4
writer : MovieWriter or str, optional
A MovieWriter instance to use or a key that identifies a class to use, such as ‘ffmpeg’ or ‘mencoder’. If None, defaults to rcParams['animation.writer']
fps : number, optional
frames per second in the movie. Defaults to None, which will use the animation’s specified interval to set the frames per second.
dpi : number, optional
Controls the dots per inch for the movie frames. This combined with the figure’s size in inches controls the size of the movie. If None, defaults to rcparam['savefig.dpi']
codec : str, optional
The video codec to be used. Not all codecs are supported by a given MovieWriter. If None, default to rcParams['animation.codec']

Tutorial and practice

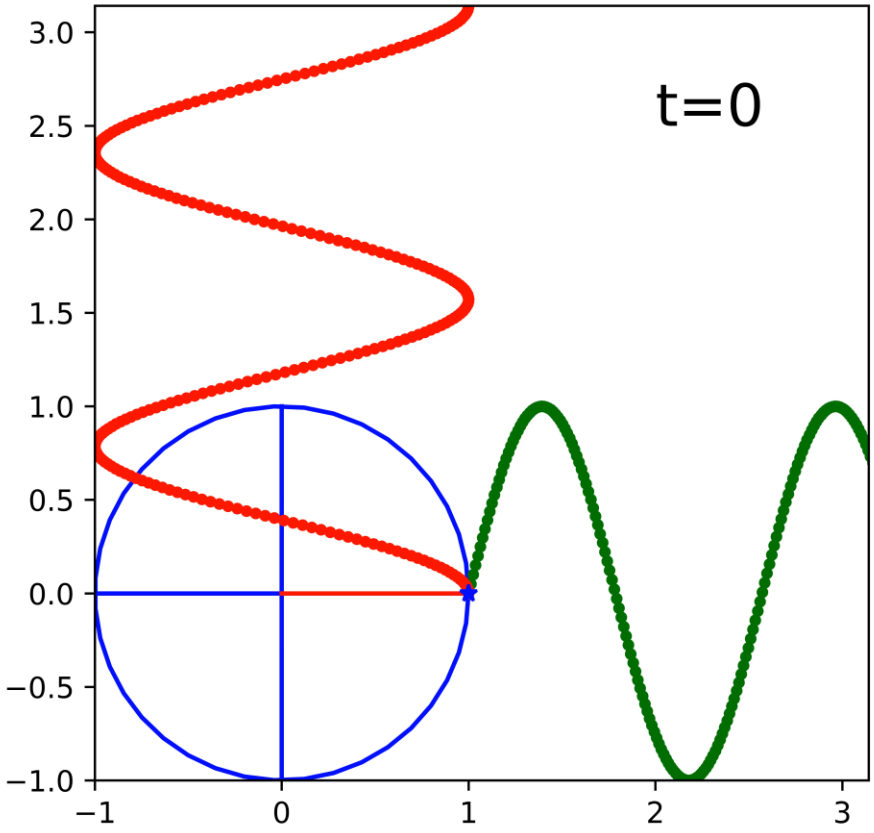
- Be able to plot the cube
- Show the animation of line plot
- Show the animation of rotating along theta and phi from 0 to 90





Assignment

Make the following video.



Animation of 3D Line and Scatter

Please learn how to make animations for 3D line and scatter plot. You can refer to the following links:

https://matplotlib.org/stable/gallery/animation/random_walk.html

https://pythonguides.com/matplotlib-3d-scatter/

https://medium.com/@pnpsegonne/animating-a-3d-scatterplot-with-matplotlib-ca4b676d4b55