Assignment 5

Varsha S P, EE21B154 <ee21b154@smail.iitm.ac.in>

March 8, 2023

1 Assignment 5

The given animation is a morphing of multiple polygons. We realise that to morphe any two diagrams we need to have equal number of points by which the polygon is plotted.

[1]:
```
%matplotlib ipympl
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from functools import partial
```

1.1 Defining a function for number of points

We realise that the number of points need to be constant for morphing any two polygons. So we can have number of points as the `lcm` of all the polygons expected to be morphed.

[2]:
```
def number_ofpoints(n_start,n_end):
    return np.lcm.reduce(np.arange(n_start,n_end+1,1))
```

1.2 Defining a function `pol`

This function is used to generate all the points that are used to plot any range of polygon.`apex` is used to determine number of points on each side of a polygon. We use `np.linspace` to generate points for every side that is (np.cos(k*angle), np.cos((k+1)*angle) gives two end points of a side, and concatenate these points for every polygon.

[3]:
```
def pol(n_start,n_end):
    all_points=[]
    for i in range(n_start,n_end+1):
        angle=2*np.pi/i
        apex=number_ofpoints(n_start,n_end)/i
        x = np.concatenate([np.linspace(np.cos(k*angle), np.cos((k+1)*angle),␣
 ↪num=int(apex)) for k in range(i)])
        y = np.concatenate([np.linspace(np.sin(k*angle), np.sin((k+1)*angle),␣
 ↪num=int(apex)) for k in range(i)])
        points=np.column_stack((x,y))
        all_points.append(points)
```

```
    return all_points
```

1.3 Using FuncAnimation

Now, since `all_points` returned from pol contains (x,y) of all the polygons we need to have a list as list of list for each polygon. And we append a list in reverse order as required by the animation. The `update` function inputs frame, which is equals to equally spaced points from (0,11.99) and the loop runs for each frame. f is defined as the int(frame), which is used as index for the xdata,ydata to get all the values for a particular polygon. These points are inputs to morph function. Morph function plots the transition between 2 polygons as alpha tends to 1. We use `FuncAnimation` to get the final animation

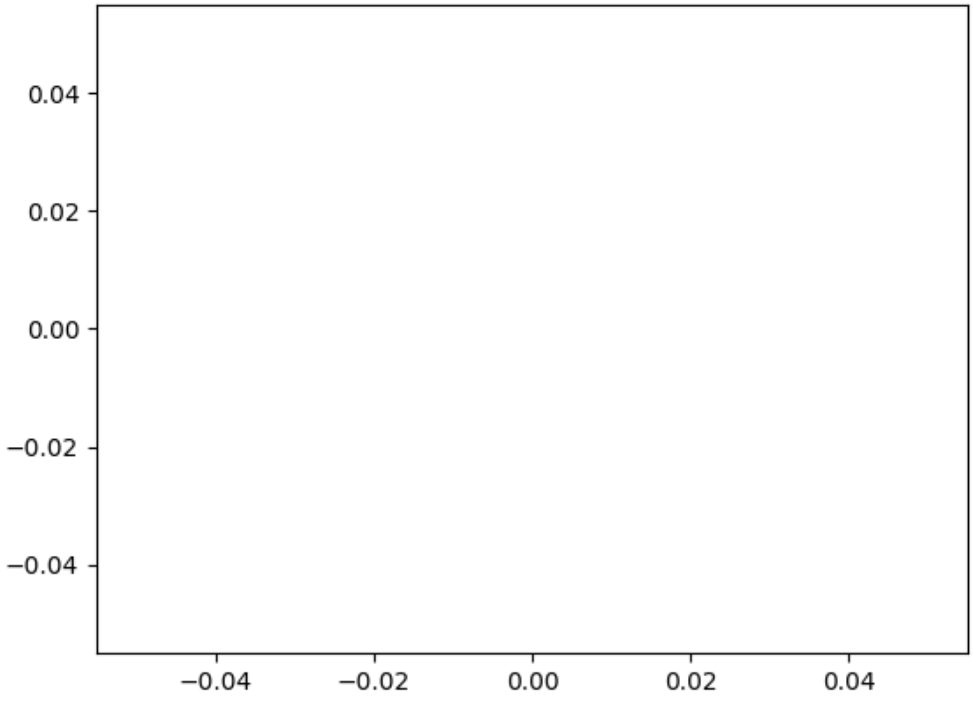
[4]:
```
fig, ax = plt.subplots()
xdata, ydata = [], []
n_start=3
n_end=8
ln, = ax.plot([], [], 'r')
def init():
    ax.set_xlim(-1.2, 1.2)
    ax.set_ylim(-1.2, 1.2)
    return ln,
point=int((number_ofpoints(n_start,n_end)/np.abs(n_end-n_start+1))) # number
arr=pol(n_start,n_end)
xdata=[arr[j][i][0] for j in range(n_end-n_start+1) for i in␣
 ↪range(number_ofpoints(n_start,n_end)) ]
xdata=[xdata[i:i + number_ofpoints(n_start,n_end)] for i in range(0,␣
 ↪len(xdata), number_ofpoints(n_start,n_end))]
xdata=xdata+xdata[::-1]
ydata=[arr[j][i][1] for j in range(n_end-n_start+1) for i in␣
 ↪range(number_ofpoints(n_start,n_end)) ]
ydata=[ydata[i:i + number_ofpoints(n_start,n_end)] for i in range(0,␣
 ↪len(ydata), number_ofpoints(n_start,n_end))]
ydata=ydata+ydata[::-1]
def update(frame):
    f=int(frame)

    x_i=np.array(xdata[f])
    x_f=np.array(xdata[f+1])
    y_i=np.array(ydata[f])
    y_f=np.array(ydata[f+1])
    xfdata, yfdata = morph(x_f, y_f, x_i, y_i, frame-f)
    ln.set_data(xfdata, yfdata)
    return ln,


def morph(x1, y1, x2, y2, alpha):
    xm = alpha * x1 + (1-alpha) * x2
```

```
    ym = alpha * y1 + (1-alpha) * y2
    return xm, ym

ani = FuncAnimation(fig, update, frames=np.linspace(0,11.
 ↪99,500),init_func=init, blit=True, interval=30, repeat=True)
plt.show()
```



1.4 Explanation

The reason behind why our animation is diffrent from the one given in the assignment is because we are not splitting the vertices and mapping them to the new vertices of the polygon. This can be done by storing every point of the polygon for each frame and using these points as the transition polygons' vertices. By doing this and altering the interval and frames we can achieve the exact same animation as the one given in the assignment.