EE2703 - Week 5

Anshul Vaddiraju EE21B151

March 8, 2023

1 Libraries

[ ]:
```
# Magic command below to enable interactivity in the JupyterLab interface
%matplotlib ipympl
# Some basic imports that are useful
import math
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.animation import FuncAnimation
```

[ ]:
```
a = int(input("Maximum Size: "))
```

2 Functions

2.1 Regular Polygon

[ ]:
```
def regular_polygon(n):
    r = 1
    k = []
    for i in range(n):
        x = r*math.cos(2*math.pi*i/n)
        y = r*math.sin(2*math.pi*i/n)
        k.append((x,y))
    return k
```

The function `regular_polygon` returns the vertices of a regular polygon whose centroid is fixed at origin and another point (1,0) is fixed

2.2 Final Polygons

[ ]:
```
def fin_pol_A(j,n):
    arr = regular_polygon(j)
    x,y = [],[]
    for i in range(len(arr)):
        x1,y1 = arr[i][0],arr[i][1]
        k = i+1
        if k == len(arr):
            k = 0
```

```
        x2,y2 = arr[k][0],arr[k][1]
        x = x+ list(x1*np.ones(n)) + list(np.linspace(x1,x2,n))
        y = y + list(y1 * np.ones(n)) + list(np.linspace(y1, y2, n))
    return np.array(x), np.array(y)
```

The function `fin_pol_A` generates n evenly spaced points along the side using the np.linspace() function and the vertices of polygon itself n times each. It then adds the coordinates of these points to two lists, `x` and `y`. These lists are returned

[ ]:
```
def fin_pol_B(j,n):
    arr = regular_polygon(j)
    x, y = list(np.linspace(arr[0][0], arr[1][0], n)), list(np.
↪linspace(arr[0][1], arr[1][1], n))
    for i in range(1,len(arr)-1):
        x1, y1 = arr[i][0], arr[i][1]
        k = i + 1
        if k == len(arr):
            k = 0
        x2, y2 = arr[k][0], arr[k][1]
        x = x + list(x1 * np.ones(n)) + list(np.linspace(x1, x2, n))
        y = y + list(y1 * np.ones(n)) + list(np.linspace(y1, y2, n))
    x = x+ list(np.linspace(arr[-1][0], arr[0][0], n))
    y = y+ list(np.linspace(arr[-1][1], arr[0][1], n))
    return np.array(x), np.array(y)
```

The function `fin_pol_B` generates n evenly spaced points along the side using the np.linspace() function and the vertices of polygon itself n times each(except the first and the last vertex. This is done to emulate the animation of a ponit moving on all the sides of the figure except the last side. It then adds the coordinates of these points to two lists, `x` and `y`. These lists are returned

2.3 Init

[ ]:
```
def init():
    ax.set_xlim(-1.2, 1.2)
    ax.set_ylim(-1.2, 1.2)
    return ln,
```

The function `init` is used to set the limits of the axis of the plot which is output.

2.4 Update

[ ]:
```
def update(frame):
    # xdata.append(frame)
    # ydata.append(np.sin(frame))
    m = int(frame)
    if m < a - 3:
        xc, yc = fin_pol_A(m+3, 100)
        xs, ys = fin_pol_B(m+4, 100)
```

```
    else:
        xc, yc = fin_pol_B(2*a-m-3, 100)
        xs, ys = fin_pol_A(2*a-m-4, 100)
    xdata, ydata = morph(xs, ys, xc, yc, frame)
    ln.set_data(xdata, ydata)
    return ln,
```

The function `update` takes frame as input and calls another function `morph` which alters the shape of the output between fig_A and fig_B based on the value of frame.

2.5 Morph

[ ]:
```
def morph(x1, y1, x2, y2, alpha):
    alpha = alpha - int(alpha)
    xm = alpha * x1 + (1-alpha) * x2
    ym = alpha * y1 + (1-alpha) * y2
    return xm, ym
```

[ ]:
```
fig, ax = plt.subplots()
xdata, ydata = [], []
ln, = ax.plot([], [], 'r')

ani = FuncAnimation(fig, update, frames=np.linspace(0, 2*a-6, 500),
init_func=init, blit=False, interval=10, repeat=True)
plt.show()
```