ISSN 2181-3187

ANIMATING PLOTS A GUIDE TO PYTHON'S POWER

Qurbonova Dilnora Nuriddinovna
Student at Bukhara State University
dilnoraqurbonova3@gmail.com.

Annotatsiya: Ushbu maqolada biz python dasturida qanday animatsiya dasturlarini tuzish nafaqat 2D fazoda balki 3D fazodagi animatsiyalarni tayyorlashni ko'rsatib o'tdik. Biz har bir codelarimizni soddalik bilan tushuntirib o'tganmiz.

Kalit so'zlar: numpy, matplotlib, animatsiya, python, vektor, grafika, funksiya, algoritm

Аннотация: В этой статье мы показали, как создавать анимационные программы на Python не только в 2D-пространстве, но и в 3D-пространстве. Мы объяснили каждый из наших кодов простым способом.

Ключевые слова: numpy, matplotlib, анимация, python, вектор, графика, функция, алгоритм.

Abstract: In this paper, we demonstrate how to create animations not only in 2D but also in 3D dimensions using Python. We provide detailed explanations for every line of code.

Key words: numpy, matplotlib, animation, python, vector, figure, function, algorithm

Introduction

Animation serves as both entertainment and an effective means of visualizing changes or evolution. Its popularity in movies, cartoons, and video games attests to its appeal. Therefore, animating our plots (or data visualizations, in general) can enhance their attractiveness and present information more effectively. This example provides a brief insight into the power of animation.

In this article, I aim to introduce how to animate plots using matplotlib.animation. First section, I will guide you through a simple example of animation in 2D. Following that, I will present more complex examples in 3D. Finally, I will conclude by highlighting the key aspects of creating animated plots. You can also download our code from our GitHub repository. The link can be found in the references section.

Creating 2D Animations with Python

Python is a powerful programming language with specialized tools for creating animations. In this article, we will explain our code line by line to ensure clarity for our readers. We will begin by writing and running our animation code.

To begin, we must import the necessary packages to implement our animations.

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

ISSN 2181-3187

```
import matplotlib.animation as animation
```

1-algorithm

As shown in Algorithm 1, we have imported all the necessary packages to proceed with our code.

```
x = np.linspace(0, 10)
y = np.sin(x)
```

2-algorithm

In Algorithm 2, we initialize the function that we are visualizing. In this example, I have chosen the sine function (sin). It's important to note that 'x' and 'y' are lists representing vectors with the same shape. When implementing functions in Python, it's crucial to ensure that 'x' and 'y' have matching shapes to avoid errors.

```
fig, ax = plt.subplots()
line, = ax.plot(x, y)
```

3-algorithm

Next, we apply matplotlib in our code. We initialize the figure objects 'fig' and 'ax' and plot a 'backbone'. It's important to note that we need to obtain the plotted object 'line' here for further updates.

```
def update(num, x, y, line):
    line.set_data(x[:num], y[:num])
    return line,
```

4-algorithm

We also need to write the update function. This function is an essential part of our animation. Animation is an “illusion”: in each frame, the plotted object is updated using the update function; all frames are saved as a file (either a gif file or an mp4 file) at the end.

```
ani = animation.FuncAnimation(fig, update, len(x), interval=100,
                              fargs=[x, y, line], blit=True)
```

5-algorithm

Next, it comes to the core part of this example. We now construct an FuncAnimation object to really perform the animation.

The first two arguments are simply fig (the figure object) and update (the function for updating the plotted object in each frame). The third argument passed into the constructor is len(x), which indicates the total number of data points (recall that x and y are lists). This is exactly the num parameter passed to update! In each frame, a value ranging from 1 to len(x) is given to the update function for generating different plots. These plots form the “illusion” of animation. The argument interval is the delay between frames in milliseconds, which is set to be 100ms in this case. The argument fargs is a tuple containing other arguments (besides num) passed to the update function, which is a tuple of x, y and line in the same order. The last argument blit is set to “True”

ISSN 2181-3187

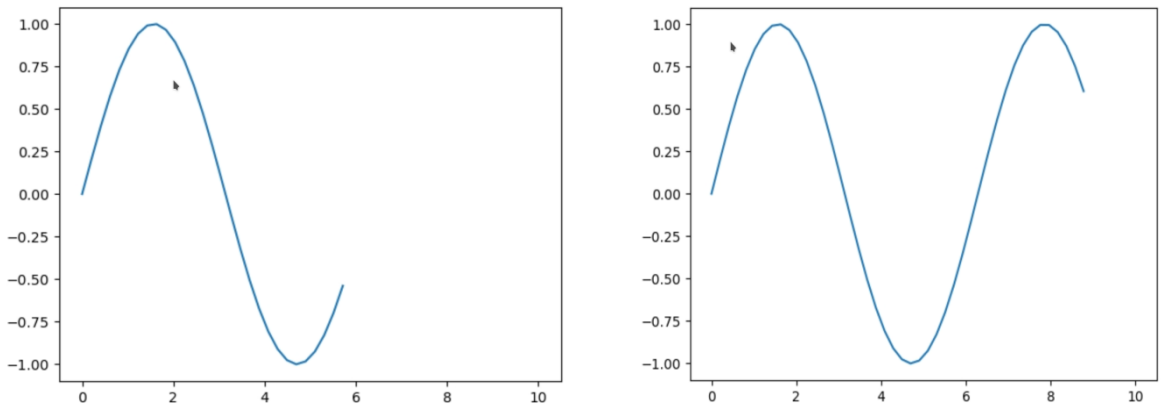
for optimizing the result. (There are many other parameters in the constructor of FuncAnimation.

```
ani.save('animation_drawing.gif', writer='imagemagick', fps=60)
```

6-algorithm

Finally, the save method of the FuncAnimation object is called to save the animation. Here, we save ani to a gif file using the gif generator imagemagick. Here, we set the frame rate fps to be 60.

As a result, we obtain the following results:



1-figure

Creating 3D Animations with Python

Now, let's delve into creating 3D animations in Python. The algorithm follows a similar structure to 2D animations. First, as in Algorithm 1, we import all the necessary packages for our 3D animations

```
def update(t):
    ax.cla()
    x = np.cos(t/10)
    y = np.sin(t/10)
    z = 5
    ax.scatter(x, y, z, s = 100, marker = 'o')
    ax.set_xlim(-2, 2)
    ax.set_ylim(-2, 2)
    ax.set_zlim(-1, 10)
```

7-algorithm

Afterward, we define an update function where the changes from one frame to the next are implemented. In our case, this involves updating the x, y, and z coordinates based on the parameter 't'. Subsequently, we clear the previous plot and generate a new

ISSN 2181-3187

one with the updated coordinates. Finally, it's advisable to set axis limits to maintain a fixed animation frame.

```
fig = plt.figure(dpi=100)
ax = fig.add_subplot(projection='3d')
ani = FuncAnimation(fig = fig, func = update, frames = 100, interval = 100)
plt.show()
```

8-algorithm

Next, you create a figure and a FuncAnimation instance. You pass the figure on which to play the animation, the update function, and other important parameters such as the number of frames and the interval between them. The complete code is as follows:

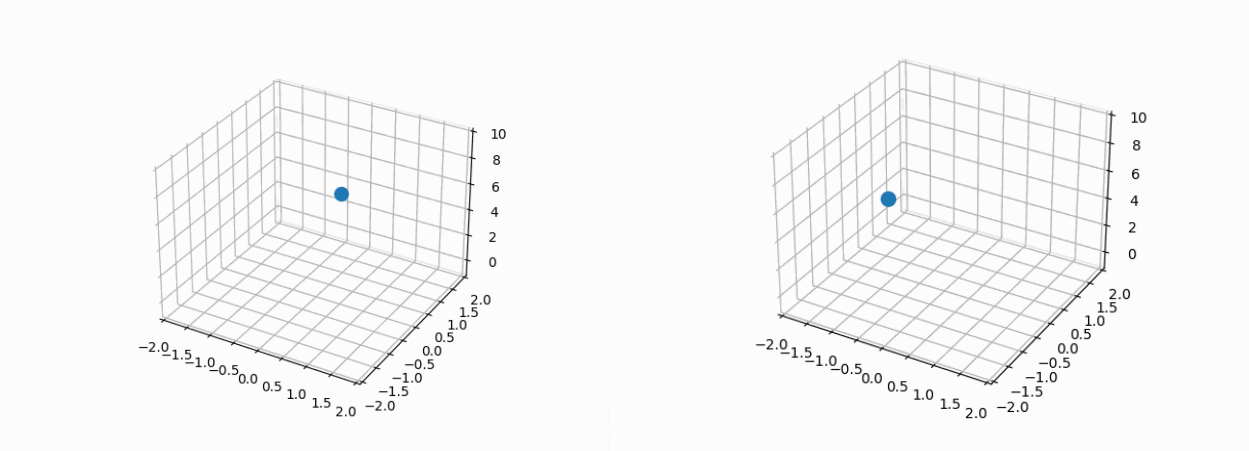
```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.animation import FuncAnimation

def update(t):
    ax.cla()
    x = np.cos(t/10)
    y = np.sin(t/10)
    z = 5
    ax.scatter(x, y, z, s = 100, marker = 'o')
    ax.set_xlim(-2, 2)
    ax.set_ylim(-2, 2)
    ax.set_zlim(-1, 10)
fig = plt.figure(dpi=100)
ax = fig.add_subplot(projection='3d')
ani = FuncAnimation(fig = fig, func = update, frames = 100, interval = 100)
plt.show()
```

9-algorithm

The results obtained show that a particle is moving from one location to another.

ISSN 2181-3187



2-figure

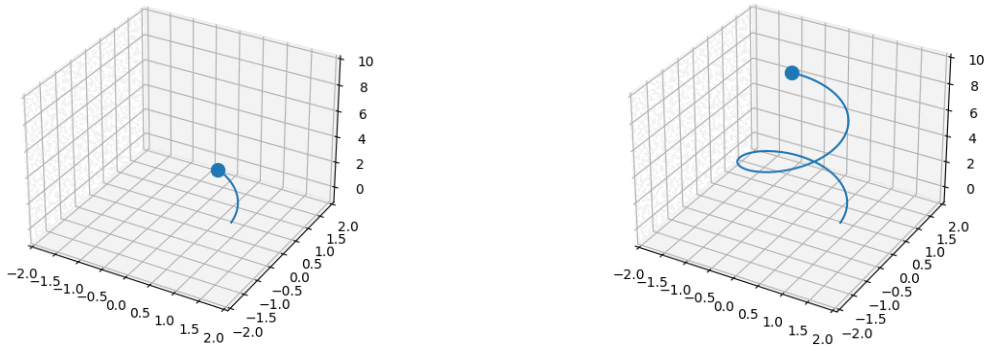
If you also wish to track the particle trajectory, you can add lists X, Y, and Z to save the particle coordinates.

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.animation import FuncAnimation
X = []
Y = []
Z = []
def update(t):
    ax.cla()
    x = np.cos(t/10)
    y = np.sin(t/10)
    z = t/10
    X.append(x)
    Y.append(y)
    Z.append(z)
    ax.scatter(x, y, z, s = 100, marker = 'o')
    ax.plot(X, Y, Z)
    ax.set_xlim(-2, 2)
    ax.set_ylim(-2, 2)
    ax.set_zlim(-1, 10)
fig = plt.figure(dpi=100)
ax = fig.add_subplot(projection='3d')
ani = FuncAnimation(fig = fig, func = update, frames = 100, interval = 100, repeat = False)
plt.show()
```

10-algorithm

ISSN 2181-3187

The resulting output is as follows, where we can also observe the trajectory of the particle.



Conclusion

In conclusion, there are four essential parts of code you should write when you animate plots with matplotlib.animation.

1. Initialize the figure objects fig and ax
2. Write the update function
3. Construct a FuncAnimation object to perform the animation
4. Call save method of the FuncAnimation object to save the animation

You also need to ensure that the environment is correctly set up before running your code.

And finally, there is more you can do with matplotlib.animation.

REFERENCES:

1. Sandro Tosi (2009). "Matplotlib for Python Developers". Packt Publishing Ltd.
2. Nicolas P. Rougier (November 2021). "Scientific Visualization: Python + Matplotlib". HAL Id: hal-03427242.
3. Dr. Pooja. "Data Visualization with Python: Exploring Matplotlib, Seaborn, and Bokeh for Interactive Visualizations". BPB Publications, 2023.
4. Jake VanderPlas. "Python Data Science Handbook: Essential Tools for Working with Data". "O'Reilly Media, Inc.", 2016
5. Утаева Феруза Холмаматовна Т.Ю. Х. и др. BUXORO TO'QIMACHILIK KOMBINATIDA MUTAXASSISLARNING O'RNI VA OILALARNING ISH BILAN TA'MINLANISHI: Utayeva Feruza Xolmamatovna, Buxoro davlat universiteti dotsenti, Teshayeva Yulduz Hakim qizi, Buxoro davlat universiteti 2-kurs magistr //Образование и инновационные исследования международный научно-методический журнал. – 2023. – №. 4. – С. 33-39.

Webpages:

1. https://numpy.org/
2. https://matplotlib.org/
3. **https://www.w3schools.com/python/matplotlib_pyplot.asp**
4. https://www.geeksforgeeks.org/using-matplotlib-for-animations/
5. https://github.com/Dilnora05/animation_python