UNIVERSIDAD DE SONORA

DEPARTAMENTO DE FÍSICA

ACTIVIDAD 8

Alumno:
Luis Alfonso Torres Flores

Profesor
Carlos Lizárraga Celaya

26 de Abril de 2017

Resumen

Python puede usarse para realizar animaciones además de generar graficas como lo hemos hecho en anteriores trabajos, por lo que en este trabajo será dedicado a un gif del efecto mariposa.

Introducción

Anteriormente hemos utilizado Python para crear graficas utilizando los datos que se nos proporcionan, obtenemos o nosotros mismos generamos. En este caso estaremos generando un gif del llamado "Efecto Mariposa". Este efecto lo podemos describir de la siguiente manera, un resultado podría parecer tener un camino determinado, invariable, sin embargo pequeños cambios puedes alterar el resultado final. Si esto lo graficamos podremos verlo como una mariposa, más bien como sus alas. Observaríamos como da vueltas por un lado pero en algún momento cambiara a otro centro y comenzara a girar a su alrededor. Eventualmente se ira moviendo de un lado a otro y el resultado que veremos es la forma anterior descrita.

Aquí presentaremos el código utilizado para generar dicho gif y algunas imágenes del efecto mariposa una vez generado.

Codigo

%matplotlib inline import numpy as np, matplotlib.pyplot as plt, glob, os import IPython.display as IPdisplay, matplotlib.font_manager as fm from scipy.integrate import odeint from mpl_toolkits.mplot3d.axes3d import Axes3D from PIL import Image

define the fonts to use for plots family = 'Myriad Pro' title_font = fm.FontProperties(family=fa style='normal', size=20, weight='normal', stretch='normal')

save_folder = 'images/lorenz-animate' if not os.path.exists(save_folder): os.makedirs(save_folder

define the initial system state (aka x, y, z positions in space) initial_state = [0.1, 0, 0]

define the system parameters sigma, rho, and beta sigma = 10. rho = 28. beta = 8./3.

define the time points to solve for, evenly spaced between the start and end times start_time = 1 end_time = 60 interval = 100 time_points = np.linspace(start_time, end_time, end_time * interval)

define the lorenz system def lorenz_system(current_state, t): x, y, z = current_state dx_dt = sigma * (y - x) dy_dt = x * (rho - z) - y dz_dt = x * y - beta * z return [dx_dt, dy_dt, dz_dt]

plot the system in 3 dimensions def plot_lorenz(xyz, n): fig = plt.figure(figsize=(12, 9)) ax = fig.gca(projection='3d') ax.xaxis.set_pane_color((1,1,1,1)) ax.yaxis.set_pane_color((1,1,1, ax.zaxis.set_pane_color((1,1,1,1)) x = xyz[:, 0] y = xyz[:, 1] z = xyz[:, 2] ax.plot(x, y, z, color='g', alpha=0.7, linewidth=0.7) ax.set_xlim((-30,30)) ax.set_ylim((-30,30)) ax.set_zlim((0,50)) ax.set_title('Lorenz system attractor', fontproperties=title_font)

plt.savefig('/:03d.png'.format(save_folder, n), dpi=60, bbox_inches='tight', pad_inches=0.1) plt.close()

return a list in iteratively larger chunks def get_chunks(full_list, size): size = max(1, size) chunks = [full_list[0:i] for i in range(1, len(full_list) + 1, size)] return chunks

get incrementally larger chunks of the time points, to reveal the attractor one frame at a time chunks = get_chunks(time_points, size=20)

get the points to plot, one chunk of time steps at a time, by integrating the system of equations points = [odeint(lorenz_system, initial_state, chunk) for chunk in chunks]

plot each set of points, one at a time, saving each plot for n, point in enumerate(points): plot_lorenz(point, n)

create a tuple of display durations, one for each frame first_last = 100 #show the first and last frames for 100 ms standard_duration = 5 #show all other frames for 5 ms durations = tuple([first_last] + [standard_duration] * (len(points) - 2) + [first_last])

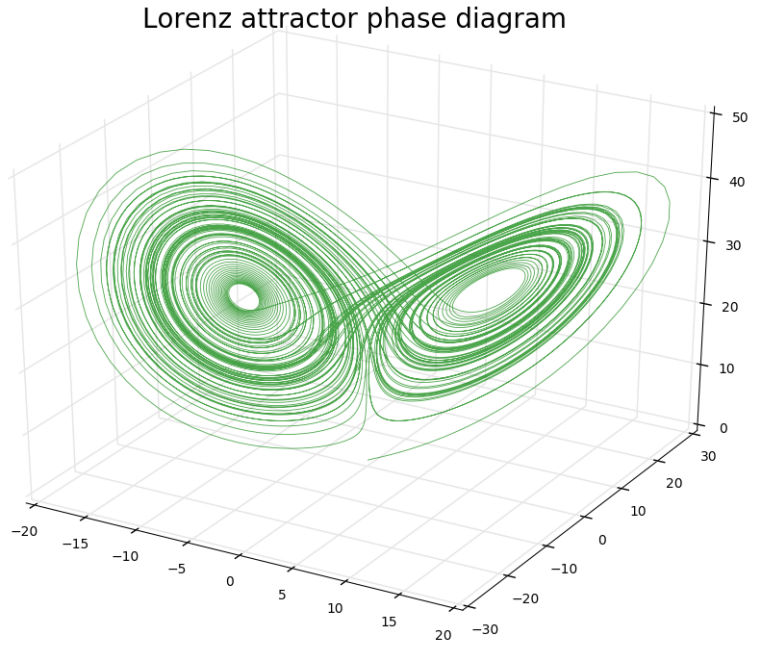
load all the static images into a list images = [Image.open(image) for image in glob.glob('/*.png'.format(save_folder))] gif_filepath = 'images/animated-lorenz-attractor.gif'

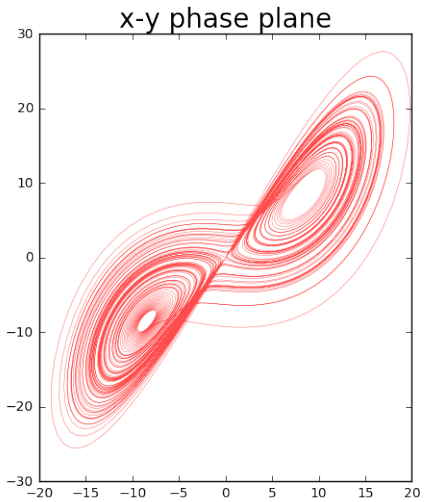
save as an animated gif gif = images[0] gif.info['duration'] = durations #ms per frame gif.info['loop'] = 0 #how many times to loop (0=infinite) gif.save(fp=gif_filepath, format='gif', save_all=True, append_images=images[1:])

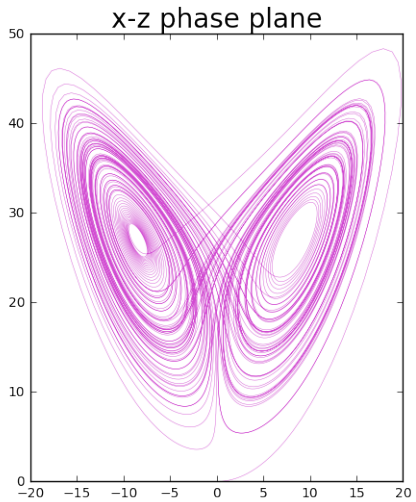
verify that the number of frames in the gif equals the number of image files and durations Image.open(gif_filepath).n_frames == len(images) == len(durations)

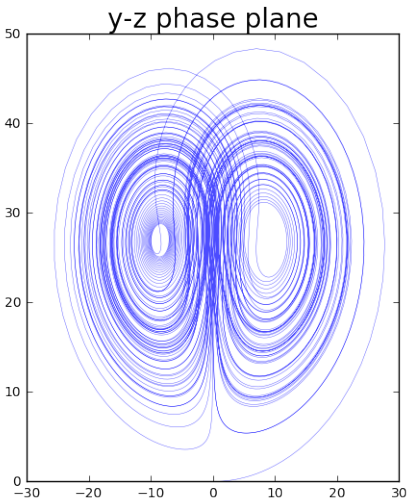
IPdisplay.Image(url=gif_filepath)

Imagenes


Lorenz attractor phase diagram
50
40
30
20
10
0
30
20
10
0
−10
−20
−30
−20
−15
−10
−5
0
5
10
15
20


x-y phase plane
30
20
10
0
−10
−20
−30
−20
−15
−10
−5
0
5
10
15
20


x-z phase plane
50
40
30
20
10
0
−20
−15
−10
−5
0
5
10
15
20


y-z phase plane
50
40
30
20
10
0
−30
−20
−10
0
10
20
30