

Lorenz Attractor

Giovanni Crisalfi

November 1, 2022

Contents

1	Intro	1
2	Python environment	2
3	Math	3
4	Plot	4

1 Intro

The Lorenz system is a system of ordinary differential equations first studied by mathematician and meteorologist Edward Lorenz. It is notable for having chaotic solutions for certain parameter values and initial conditions. In particular, the Lorenz attractor is a set of chaotic solutions of the Lorenz system. In popular media the “butterfly effect” stems from the real-world implications of the Lorenz attractor, namely that in a chaotic physical system, in the absence of perfect knowledge of the initial conditions (even the minuscule disturbance of the air due to a butterfly flapping its wings), our ability to predict its future course will always fail. This underscores that physical systems can be completely deterministic and yet still be inherently unpredictable. The shape of the Lorenz attractor itself, when plotted in phase space, may also be seen to resemble a butterfly.¹

Let’s explore the shape of the Lorenz attractor with Python in Org mode/Emacs. Because why not.

¹Source: Wikipedia

2 Python environment

As we had chosen Python, we need to import some python libraries:

- Numpy
- Scipy
- Matplotlib

We could install them globally with pip, but I prefer setting up a virtual environment. It's up to you, this script works both ways, but the second one require a couple of additional steps. Basically you have to:

- Create the new venv
- Install the packages inside the venv
- Access the venv from inside Emacs

I assume you know how to do the first two steps. About accessing the virtual environment from Emacs, I suggest the use of the `pyvenv` package. On vanilla Emacs you can install it with `use-package`. If, like me, you use a different distribution of Emacs, you will proceed with the installation in accord to that.

Now, you should tell Emacs' `pyvenv` where is our venv. Let's say our venv is placed in `~/org-lorenz-attractor/venv`,

```
(pyvenv-activate "~/org-lorenz-attractor/venv")
```

```
echo $VIRTUAL_ENV  
which python
```

Check the version of matplotlib:

```
import matplotlib  
print(matplotlib.__version__)
```

I'm using 3.6.1 for this demo. You could try with other versions, but keep in mind that APIs may change. Now you can import the libraries and go to the interesting part of this file.

```

try:
    import numpy as np
    from scipy.integrate import solve_ivp
    import matplotlib.pyplot as plt
    from mpl_toolkits.mplot3d import Axes3D
    print("Success.")

except Exception as e:
    print(e)

```

3 Math

The Lorenz system of coupled, ordinary, first-order differential equations have chaotic solutions for certain parameter values σ , ρ and β and initial conditions, $u(0)$, $v(0)$ and $\omega(0)$.²

$$\frac{du}{dt} = \sigma(v - u)$$

$$\frac{dv}{dt} = \rho u - v - u\omega$$

$$\frac{d\omega}{dt} = uv - \beta\omega$$

Translating the math in code, you should:

- Declare the parameters
- Wrote off the equations as a function
- Integrate for the assigned parameters using `scipy`
- Interpolate the solution

```

# Lorenz paramters and initial conditions.
sigma, beta, rho = 10, 2.667, 28
u0, v0, w0 = 0, 1, 1.05

# Maximum time point and total number of time points.
tmax, n = 100, 10000

```

²Source: Scipython

```

def lorenz(t, X, sigma, beta, rho):
    """The Lorenz equations."""
    u, v, w = X
    up = -sigma*(u - v)
    vp = rho*u - v - u*w
    wp = -beta*w + u*v
    return up, vp, wp

# Integrate the Lorenz equations.
soln = solve_ivp(lorenz, (0, tmax), (u0, v0, w0), args=(sigma, beta, rho),
                 dense_output=True)

# Interpolate solution onto the time grid, t.
t = np.linspace(0, tmax, n)
x, y, z = soln.sol(t)

```

4 Plot

Now, with the already imported matplotlib, we have to create the actual image of the Lorenz attractor. Plot the points you gained from interpolation with Matplotlib.

N.B. 'k' is just a single character shorthand notation for the black color. Check the docs if you want to change it.

```

# We start with the constants that describe the image itself.
WIDTH, HEIGHT, DPI = 1000, 750, 100

# Plot the Lorenz attractor using a Matplotlib 3D projection.
try:
    fig = plt.figure(facecolor='k', figsize=(WIDTH/DPI, HEIGHT/DPI))
    ax = fig.add_subplot(projection='3d')
    ax.set_facecolor('k')
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1)

    # Make the line multi-coloured by plotting it in segments of length s which
    # change in colour across the whole time series.
    s = 10

```

```

# The 'winter' or 'cool' colormap are among the sequential ones.
# I picked 'copper' this time, but you can select your favorite.
# https://matplotlib.org/stable/tutorials/colors/colormaps.html
cmap = plt.cm.copper

for i in range(0,n-s,s):
    ax.plot(x[i:i+s+1], y[i:i+s+1], z[i:i+s+1], color=cmap(i/n), alpha=0.4)

# Remove all the axis clutter, leaving just the curve.
ax.set_axis_off()

plt.savefig('lorenz.png', dpi=DPI)
plt.show()
print("Success.")

except Exception as e:
    print(e)

```

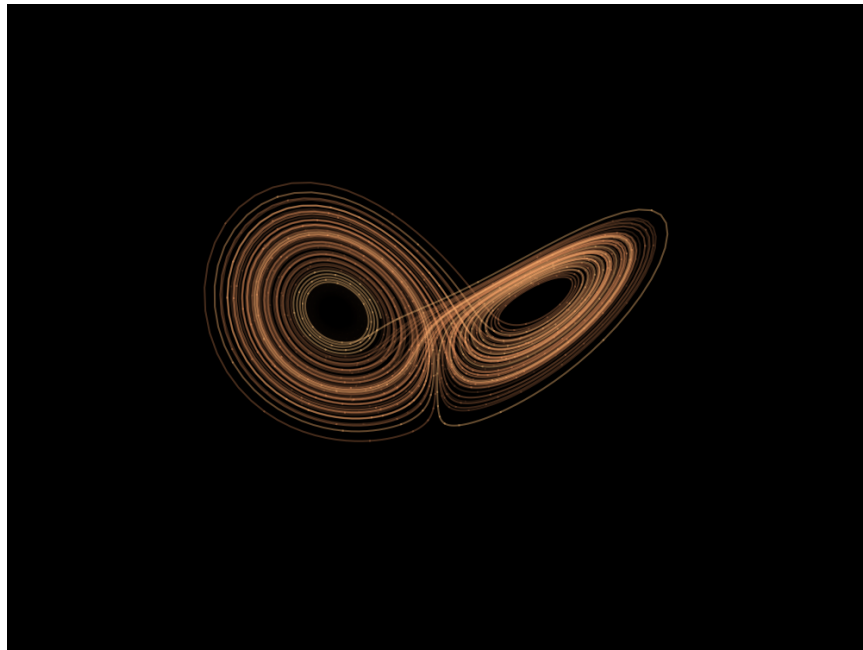


Figure 1: The output image