Mattia Filipponi
Filippo Zaccari

Project Overview

The goal is to **generate and measure the fractal dimension** of several types of fractal using the methods described in the paper **“Estimating fractal dimension” by James Theiler (1989).**
The fractals are generated and visualized using two different approaches in the Python language.

Background

Fractals

In mathematics, a **fractal** is a geometric shape containing detailed structure at arbitrarily small scales, usually having a **fractal dimension** strictly exceeding the topological dimension. Many fractals appear similar at various scales. This exhibition of similar patterns at increasingly smaller scales is called self-similarity.

Background

Fractal Dimension

“One of the exceptional characteristics of fractals is that they can be described by a noninteger dimension. The geometry of fractals and the mathematics of fractal dimension have provided useful tools for a variety of scientific disciplines, among which is chaos. Chaotic dynamical systems exhibit trajectories in their phase space that converge to a strange attractor. The fractal dimension of this attractor counts the effective number of degrees of freedom in the dynamical system and thus quantifies its complexity.” - **James Theiler**

Background

Fractal Dimension

Fractal dimension is a matematical concept that describes the complexity and form factor of a fractal object. Unlike traditional geometric shapes, **fractals can have non-integer dimension** that evaluates their capability to fill the space.

For example:

- The costline of Great Britain: the closer you look at it, the more details you find.
- Tree branch or a snowflake: structure repeat itself in different scales.

These objects can have a non-integer dimension like 1.3 or 1.7.

Background

Box-counting

The idea is to **evaluate the complexity** of an object on a plane by covering it with a square grid: We count the number of squares that contain at least a point of the object. Then, We make the squares of the grid smaller and then count again. The **speed** with which the number of occupied cells grows gives a measure of how complex the object is.

$$D_0 \approx \frac{\log[1 - n(r)]}{\log r}, \quad r = size \\ n(r) = \#boxes$$

Background

Correlation dimension

It's a way to measure **how densely distributed points are on an attractor.**

Instead of covering the space with boxes we look at the distances between pairs of points.

For each distance r we count how many pairs of points are included into r, in this way we calculate the **correlation function** $C(r)$

If we have a fractal system, the function grows according to the power of r: $C(r) \propto r^{\nu}$

The correlation dimension is then the slope of the logarithm:

$$C(r) = \frac{1}{N(N-1)} \sum_{i \neq j} \Theta(r - \|x_i - x_j\|), \; \nu = \lim_{r \to 0} \frac{\log C(r)}{\log r}$$

Computational approches

To generate and visualize fractals and estimate their fractal dimension two Python frameworks/libraries were used:

- **Mesa**: framework that brings NetLogo approach to Python but keeping the language flexibility

- **Plotly**: library to plot complex graphs.


MESA

Mesa

Framework

Framework that aims to offer the possibility to quickly create an interactive program to simulate an agent-based model.
The root logic components are the classes “Agent” and “Model” over which the desired behaviour can be developed.
The visualization side is managed using Solara with a library of available components.
Each complete application can be made in a single file (or divided in several python files for organization).

Mesa Process

1. Define agent(s) and model(s) via OOP
2. Define global parameters and UI elements in a parameter format
3. Execute the code via Solara
4. Open the web page and interact with a NetLogo-like interface

Mesa

Advantages

Capabilities of the Python language. Out-of-the-box NetLogo-like feel. Availability of models library. Possibility to implement additional features and changes.

Disadvantages

Being a relatively new framework there is not much literature/examples about it. No 3D capabilities.

Mesa

Let's see it in action!


PLOTLY
and Tkinter...

Tkinter

Graphical User Interface

Standard Python library used to create basic graphical user interfaces (GUIs) in a **lightweight** and accessible way.
It provides a set of ready-to-use components such as buttons, sliders, labels, and input fields, which can be arranged using layout managers.
The logic is structured around widgets and callback functions, allowing the interface to respond to user actions.

Plotly

Interactive Visualization

Framework designed to easily **create interactive visualizations in Python**. It supports a wide range of chart types, from simple scatter plots to complex 3D surfaces and geographical maps.

The core component is the Figure object, which can be built using high-level commands (via Plotly Express) or customized in detail using Graph Objects.

It integrates seamlessly in **Jupyter** notebooks and allows interactive elements such as zoom, hover and selection.

It is particularly suitable for rendering mathematically-generated structures such as fractals, thanks to its support for high-resolution plotting and **real-time interactivity**.

Plotly

Advantages

- Allows fast creation of interactive plots with minimal code.
- Supports a wide variety of 2D and 3D chart types, suitable for both scientific data and complex mathematical visualizations like fractals.
- Integrates well with Jupyter Notebooks.

Provides hover tooltips, zooming, and dynamic updates out of the box.

Disadvantages

- Slower performance with very large datasets or highly iterative animations compared to low-level libraries.
- Can be less flexible than libraries like Matplotlib for low-level control over certain plot details.

Plotly

Let's see it in action!


THANK
YOU