# Course Project: Scientific Computing using Python

IKHTIYOR NEMATOV

## 1 PROBLEM STATEMENT

This project relates to the Lorenz attractor. In his 1963 paper, Edward Lorenz presented a simple mathematical model for atmospheric convection, which was based on three coupled first-order Ordinary Differential Equations (ODEs) as:

$$\frac{dx}{dt} = \sigma(y - x) \tag{1}$$

$$\frac{dy}{dt} = x(\rho - z) - y \tag{2}$$

$$\frac{dz}{dt} = xy - \beta z \tag{3}$$

Where: $x, y, z$ are positions in space
$\sigma, \rho, \beta$ are model parameters (Provided 5 different test cases)
$t$ is time dependency, such that $t \leftarrow nt_\delta$ having $n \in \{0, 1, ..., N\}$ assuming uniform sampling

Given the following ordinary differential equations (ODEs), create an ODE solver in Python that can compute the solution for $(x, y, z)$ in discrete time, given initial values $(x[0], y[0], z[0])$, and simulate $(x[n], y[n], z[n])$ for $n = 0, 1, \ldots, N-1$, where $N$ and $t\delta$ are provided:

$$\frac{dx}{dt} = f_x(x, y, z)$$
$$\frac{dy}{dt} = f_y(x, y, z)$$
$$\frac{dz}{dt} = f_z(x, y, z)$$

Overall task is consists of the following :

- Experiment with various values of $t\delta$ and $N$ to find their accurate/appropriate values.
- Propose some possible initial conditions for $(x[0], y[0], z[0])$ and study their effect on the results.
- Plot 3D and 2D probability density function (pdf) plots for $(x, y, z)$, $(x, y)$, $(x, z)$, and $(y, z)$ respectively.
- Use colors in the plots to indicate the intensity of the Euclidean distance from one time step $n$ to the next $n + 1$.
- Store the configuration and data in a cross-platform format.
- Follow the principles of reproducible research.

Author's address: Ikhtiyor Nematov, ikne@cs.aau.dk.

## 2 SOLUTION

### 2.1 Euler Approximation

We utilize the Euler general approximation method, as suggested in the project description, to derive equations 6, 8, and 10 using Euler approximation. The general Euler approximation, with a variable named $v$, is represented as shown in 4.

$$\frac{dv}{dt}\left((n+1)t_\delta\right) \approx \frac{v((n+1)t_\delta) - v(nt_\delta)}{t_\delta} \tag{4}$$

By denoting the general variable as $v(nt_\delta) = v[n]$, the specific differential equation 1 can be expressed as equation 5:

$$\frac{x[n+1] - x[n]}{t_\delta} \approx \sigma \cdot (y[n] - x[n]) \tag{5}$$

We obtain equation 6 by performing algebraic operations on equation 5 as follows:

$$\begin{aligned} t_\delta \cdot \frac{x[n+1] - x[n]}{t_\delta} &\approx \sigma \cdot (y[n] - x[n]) \cdot t_\delta \\ x[n+1] - x[n] &\approx \sigma \cdot (y[n] - x[n]) \cdot t_\delta \\ x[n+1] - x[n] + x[n] &\approx \sigma \cdot (y[n] - x[n]) \cdot t_\delta + x[n] \end{aligned}$$

$$x[n+1] \approx \sigma \cdot (y[n] - x[n]) \cdot t_\delta + x[n] \tag{6}$$

Similarly, by denoting the general variable as $v(nt_\delta) = v[n]$, the specific differential equation 2 can be expressed as equation 7:

$$\frac{y[n+1] - y[n]}{t_\delta} \approx x[n] \cdot (\rho - z[n]) - y[n] \tag{7}$$

We derive equation 8 by performing algebraic operations on equation 7 as follows:

$$\begin{aligned} t_\delta \cdot \frac{y[n+1] - y[n]}{t_\delta} &\approx (x[n] \cdot (\rho - z[n]) - y[n]) \cdot t_\delta \\ y[n+1] - y[n] &\approx x[n] \cdot (\rho - z[n]) \cdot t_\delta - y[n] \cdot t_\delta \\ y[n+1] - y[n] + y[n] &\approx x[n] \cdot (\rho - z[n]) \cdot t_\delta - y[n] \cdot t_\delta + y[n] \end{aligned}$$

$$y[n+1] \approx x[n] \cdot (\rho - z[n]) \cdot t_\delta - y[n] \cdot t_\delta + y[n] \tag{8}$$

Similarly, by denoting the general variable as $v(nt_\delta) = v[n]$, the specific differential equation 3 can be expressed as equation 9:

$$\frac{z[n+1] - z[n]}{t_\delta} \approx x[n] \cdot y[n] - \beta \cdot z[n] \tag{9}$$

We derive equation 10 by performing algebraic operations on equation 9 as follows:

$$\begin{aligned} t_\delta \cdot \frac{z[n+1] - z[n]}{t_\delta} &\approx (x[n] \cdot y[n] - \beta \cdot z[n]) \cdot t_\delta \\ z[n+1] - z[n] &\approx x[n] \cdot y[n] \cdot t_\delta - \beta \cdot z[n] \cdot t_\delta \\ z[n+1] - z[n] + z[n] &\approx x[n] \cdot y[n] \cdot t_\delta - \beta \cdot z[n] \cdot t_\delta + z[n] \end{aligned}$$

$$z[n+1] \approx x[n] \cdot y[n] \cdot t_\delta - \beta \cdot z[n] \cdot t_\delta + z[n] \tag{10}$$

## 2.2 Simulations

In the preceding section, we derived equations 6, 8, and 10 using Euler approximation. Now, let's discuss the simulation or computation process for obtaining the values of $(x, y, z)$.

To compute the values of $(x, y, z)$, we propose an iterative process where the values are updated at each step until a desired number of steps, denoted by $N$, is reached. The iterative process involves the following steps:

1. Initialize the initial values of $(x[0], y[0], z[0])$. 2. For each step $n$ from 0 to $N-1$, compute the values of $(x[n+1], y[n+1], z[n+1])$ using the Euler approximation equations 6, 8, and 10, where the values of $(x[n], y[n], z[n])$ are known. 3. Update the values of $(x[n], y[n], z[n])$ with $(x[n+1], y[n+1], z[n+1])$ for the next iteration. 4. Repeat steps 2 and 3 until the desired number of steps $N$ is reached.

By iterating through these steps, you can compute or simulate the values of $(x, y, z)$ over the specified number of steps. The initial values and the choice of $N$ will determine the accuracy and resolution of the computed values.

*2.2.1 Selection of Values.* In our simulation or computation process, we make the following choices for the initial values $x$, $y$, and $z$, as well as the parameters $N$ and $t_\delta$:

$$x = y = z = 1, \quad N = 10,000, \quad t_\delta = 0.02$$

These choices of values have been found to work well for all the given cases. By setting $x$, $y$, and $z$ to 1 initially, we establish a starting point for the iterative process. The value of $N = 10,000$ determines the number of steps or iterations in the process, indicating the resolution and accuracy of the computed values. The choice of $t_\delta = 0.02$ represents the time step or interval used in the computations.

It is worth noting that these values strike a balance between accuracy and computational efficiency. While higher values of $N$, such as 50,000, may provide increased accuracy, they also require more computational resources and time to plot the resulting graph. Therefore, the chosen values ensure a reasonable compromise between accuracy and efficiency, allowing for faster visualization of the graph while still producing satisfactory results for the given cases.

*2.2.2 Simulation Algorithm.* The algorithm for computing the values of $(x, y, z)$ is described in the pseudocode shown in Algorithm 1. The input to the algorithm includes the initial values of $(x, y, z)$, and the output is the set of resultant values of $(x, y, z)$ for a specified number of simulations, denoted by $N$.

The algorithm consists of a function called $simulator$, which takes three parameters representing the initial values of $(x, y, z)$. Within the function, new values for $(x, y, z)$ are computed using equations 6, 8, and 10 for $N$ iterations using a for loop (lines 4-7). The updated values of $(x, y, z)$ are added to the set of resultant values $R$ (line 9), and the current values of $(x, y, z)$ for the ongoing iteration are updated with the new values (line 10). Finally, the function returns the set of resultant values for $(x, y, z)$ computed over $N$ iterations (line 11).

**Algorithm 1** Simulation Algorithm using Euler's Approximation

**Require:** Initial values $x_i, y_i, z_i$
**Ensure:** Set of resultant values $R$

**function** *simulate*$(x_i, y_i, z_i)$
  $x_c \leftarrow x_i, y_c \leftarrow y_i, z_c \leftarrow z_i$ ▷ Current values
  $R \leftarrow$ ▷ Resultant set of values
  **for** $n \leftarrow 0$ to $N - 1$ **do**
    $x_n \leftarrow \sigma \cdot (y_c - x_c) \cdot t_\delta + x_c$ ▷ Update $x$
    $y_n \leftarrow t_\delta \cdot (x_c \cdot (\rho - z_c) - y_c) + y_c$ ▷ Update $y$
    $z_n \leftarrow x_c \cdot y_c \cdot t_\delta - \beta \cdot z_c \cdot t_\delta + z_c$ ▷ Update $z$
    $R \leftarrow R \cup \{(x_n, y_n, z_n)\}$ ▷ Add to resultant set
    $x_c \leftarrow x_n, y_c \leftarrow y_n, z_c \leftarrow z_n$ ▷ Update current values
  **return** $R$

*2.2.3 Visualization.* The visualization of the simulated values of $(x, y, z)$ for the five parameter cases is presented using 3D plots. The intensity of the color in these plots represents the Euclidean distance between two consecutive time steps, with colors computed based on distance-based RGB values. The 2D plots are also provided, showing the relationships between pairs of variables, such as $(x, y)$, $(x, z)$, and $(y, z)$. These plots include both colored and non-colored versions.

To view the complete set of 2D plots and access the resources, including the 3D plots, you can visit the GitHub repository[1] associated with this project.

As an example, Figure **??** displays the three-dimensional plots for each parameter case. Note: The figures and additional plots can be found in the GitHub repository mentioned above.

*2.2.4 Data Storage.* The configuration for each case is stored in a text file, and the resulting values (x, y, z) are stored in a CSV format.

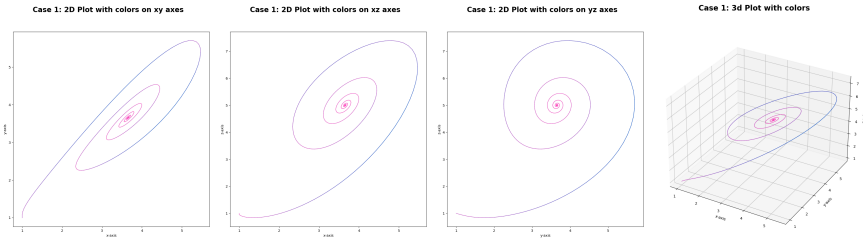
### 2.3 Code Structure

The project's source code is available on GitHub, organized into two modules: *Core* and *Test Cases*. The *Result* directory stores the simulation results. The code is well-documented, ensuring clarity and facilitating future maintenance.

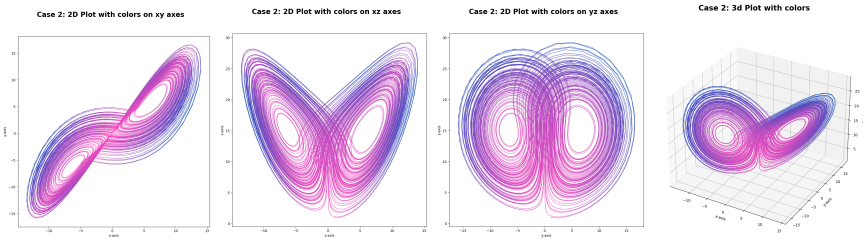
*2.3.1 Code Modules.* The project's source code consists of the following modules:

(1) *lorenz*: This module contains five files: *init*, *filehandling*, *solver*, *plot*, and *testing*. The *solver* file implements Algorithm 1 and provides getter and setter methods for the required parameters. It also includes methods to compute the values of $x$, $y$, and $z$ using the given equations, as well as a method to retrieve the class configuration as a string. The *plot* file includes methods for plotting 2D and 3D plots in color format. The *filehandling* file contains methods for writing the configuration and the $(x, y, z)$ values to files. The *testing* file includes test methods to verify the functionality of the *solver* class using the *Unittest* library.
(2) *cases*: This module includes separate files for each test case. These files execute the ODE solver with different configurations, plot the results, and save the configurations.
(3) *result* directory: This directory contains folders to store the resultant files for each test case.

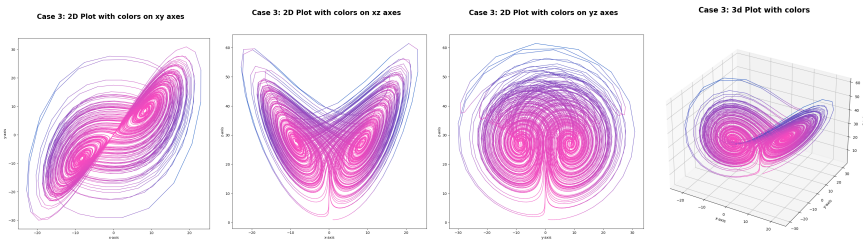
[1] https://github.com/iNema9590/PhD_course_PythonForScientificCompyting



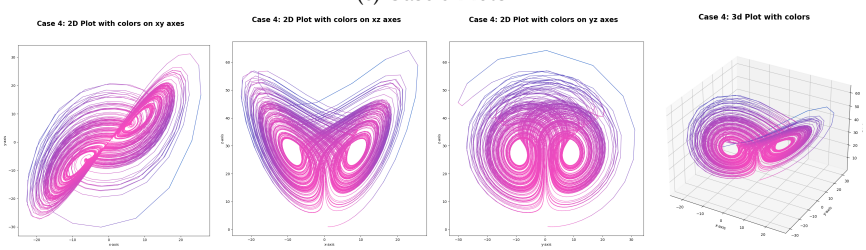

(a) Case 1 Plots



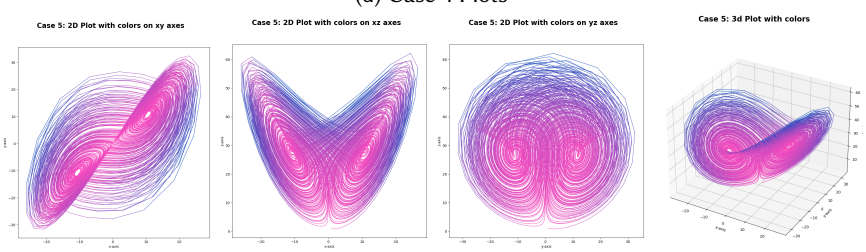

(b) Case 2 Plots




(c) Case 3 Plots




(d) Case 4 Plots




(e) Case 5 Plots

### 2.4 Design Decisions

Throughout the development of the project, several design decisions were made to ensure the code meets the requirements and provides the desired functionality. Some of the key design considerations include:

- Determining appropriate values for $t_\delta$ and $N$ by experimenting with different values and selecting the ones that work well for all the test cases (as described in Section 2.2.1).

- Choosing suitable initial conditions for $(x[0], y[0], z[0])$ based on the proposed values mentioned in Section 2.2.1.

- Creating 3D and 2D PDF plots for $(x, y, z)$ and $(x, y)$, $(x, z)$, $(y, z)$ respectively, as shown in Figure ??. Additional plots are available in the submitted resources.

- Incorporating colors in the plots to represent the intensity of the Euclidean distance between consecutive time steps.

- Storing the configuration and data in a cross-platform compatible format, such as text files for configuration and CSV files for $(x, y, z)$ values.

- Following the principles of reproducible research to ensure that the project's results can be easily reproduced and validated.

### 2.5 Testing

To ensure the correctness and reliability of the ODE Solver functionality, comprehensive testing was performed. The testing process involved creating test cases for each method of the *ODE Solver* class using the *Arrange*, *Act*, and *Assert* approach commonly used in unit testing. The test cases were designed to cover different scenarios and expected behaviors of the methods. While the *ODE Solver* class was thoroughly tested, the File Handler and Plotter classes were not extensively tested as they mainly deal with input/output and graphical operations.

The testing approach and specific test cases are well-documented within the code, ensuring that the tests can be easily understood and repeated in the future.