Exploring the Logistic Map Through Python

Discrete Dynamical Systems and Chaos Theory

Zubair Ali

Understanding Discrete-Time Dynamical Systems

A Dynamical System is a mathematical model describing how points in a space evolve over time. In Discrete-Time Systems, updates occur at distinct time intervals, unlike continuous changes over time

Basic Concepts

- State: Represents the system's initial conditions or current status.
- Parameters: Fixed values that influence the system's evolution.
- Different values of the growth rate parameter r in the logistic map can lead to a wide array of behaviors, ranging from steady state and periodic cycles to complex, chaotic dynamics, showcasing the sensitivity of the system to this crucial parameter.

Real-World Examples

- Population Models: These models use discrete steps to predict changes in population over time, considering factors like birth rate, death rate, and carrying capacity.
- Economic Models: Such models simulate investment growth or market changes over discrete periods, helping in financial forecasting and policy-making.

Basics of the Logistic Map

Logistic Map equation : $x_{n+1} = r \cdot x_n \cdot (1 - x_n)$

This simple equation models complex behavior in systems over time.

Variables and Parameters

- x_n : Represents the state of the system at time *n*
- r : The rate parameter, controlling the system's growth or decline.
- The system's evolution is governed by iterative applications of a specific function or set of functions.

The logistic map effectively models population dynamics by accounting for the limited availability of resources, balancing reproductive growth against environmental constraints. It demonstrates the interaction between reproduction rates and resource availability, where excessive growth is naturally curbed by environmental limits.

As the parameter r varies, the logistic map showcases a transition from predictable behavior to chaos, serving as a classic example of how simple deterministic rules can lead to complex and unpredictable patterns.

Python’s Role in Analyzing Dynamical Systems

Python's ease of use and powerful libraries, such as NumPy for numerical computations and Matplotlib for plotting, make it an ideal tool for scientific computing. It has the ability to handle large datasets and complex calculations.

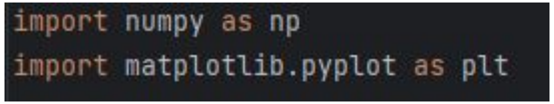
In the realm of discrete-time dynamical systems, Python excels in simulating their behavior over time. Python enables us to iterate these systems efficiently and visualize their evolution, providing insights into their underlying dynamics.

The use of computational methods in Python extends our capability to understand and analyze complex systems. By simulating the behavior of these systems, Python helps in unveiling patterns and behaviors that are otherwise difficult to predict or understand.

Key Python Libraries

NumPy

NumPy, short for Numerical Python, is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a vast collection of mathematical functions to operate on these arrays. In our logistic map implementation, NumPy's array structures allow us to efficiently store and manipulate large sets of numerical data, essential for calculating the iterations over a range of *r* values.



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

Matplotlib

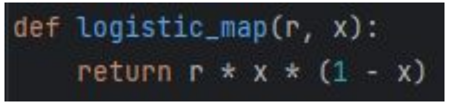
Matplotlib is a plotting library in Python that enables us to create a wide range of static, animated, and interactive visualizations. It's known for its flexibility and ease of use in generating graphs and charts.
For the logistic map, Matplotlib helps us in visually representing the data, allowing us to plot the bifurcation diagram and observe the transition from order to chaos as *r* varies.

‘logistic_map’ function

Function Overview:
The logistic_map function is the heart of our Python implementation. It defines the logistic map equation $x_{n+1} = r \cdot x_n \cdot (1 - x_n)$, which we use to simulate the behavior of the logistic map over time.

Parameters:
r (float): This parameter represents the rate of growth or reproduction. It's a key factor that influences how the population (or the value of *x*) changes with each iteration.
x (float): This is the current value or state of our system, representing, for example, the current population ratio.

What the Function Does:
Each time the function is called, it calculates the next value in the sequence based on the current value and the growth rate. This iterative process is what allows us to observe the behaviors of the logistic map, from stable states to chaotic fluctuations.



```python
def logistic_map(r, x):
    return r * x * (1 - x)
```

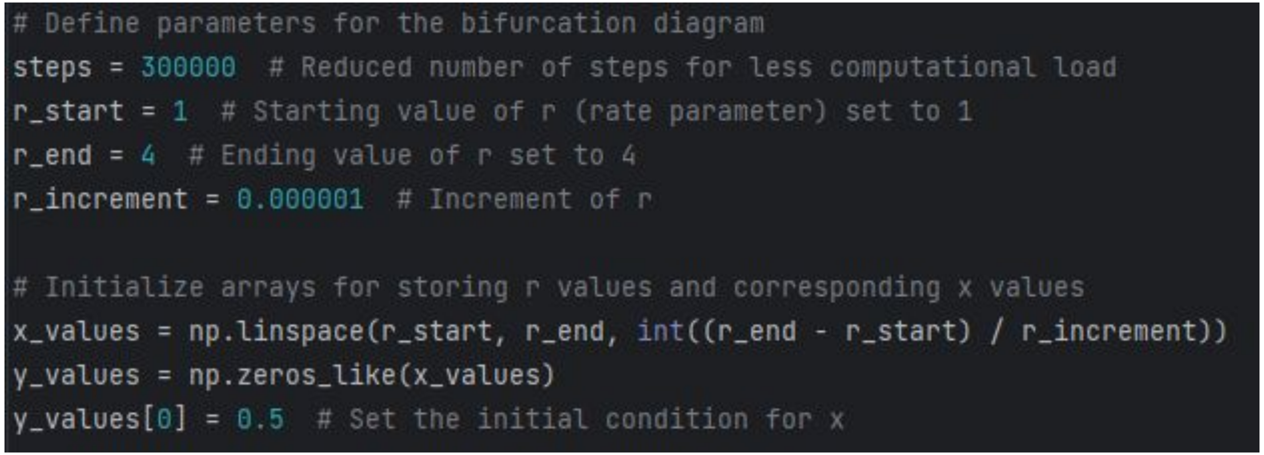
Configuring Parameters for the Bifurcation Diagram

Defining the Parameters:
The first step in our Python code is to define key parameters for the bifurcation diagram:

- steps: Number of iterations to perform, set to 300,000 for detailed analysis with manageable computational load.
- r_start and r_end: Define the range of the rate parameter *r*, starting from 1 and ending at 4, focusing on a specific segment of the logistic map.
- r_increment: The step size for incrementing *r*, chosen to be small (0.000001) for high resolution in the diagram.

Initializing arrays for r and x values:

- We use NumPy's linspace function to create an array of *r* values from r_start to r_end. This array represents the different growth rates we will explore in the logistic map.
- An array y_values is initialized to store the corresponding *x* values for each *r*. It's set to the same size as x_values and starts with an initial condition of 0.5, representing our starting point for the population ratio or the state of the system.


```python
# Define parameters for the bifurcation diagram
steps = 300000  # Reduced number of steps for less computational load
r_start = 1  # Starting value of r (rate parameter) set to 1
r_end = 4  # Ending value of r set to 4
r_increment = 0.000001  # Increment of r

# Initialize arrays for storing r values and corresponding x values
x_values = np.linspace(r_start, r_end, int((r_end - r_start) / r_increment))
y_values = np.zeros_like(x_values)
y_values[0] = 0.5  # Set the initial condition for x
```

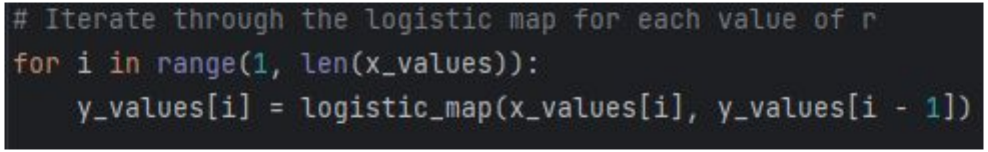
Simulating the Logistic Map

The iteration process:
This segment of the code is where we simulate the behavior of the logistic map for each value of the growth rate *r*. Using a for loop, we iterate through each *r* value stored in x_values.

In each iteration, the logistic_map function is called to calculate the next *x* value, given the current *x* value and the *r* value. This iterative process allows us to explore how the system evolves over time for different growth rates.

Building the Bifurcation Diagram:
As we iterate through each *r*, the corresponding *x* values are stored in y_values. This array accumulates the final state of the system after numerous iterations for each *r*, which is crucial for plotting the bifurcation diagram.

The logic y_values[i] = logistic_map(x_values[i], y_values[i - 1]) ensures that each new *x* value is calculated based on the most recent *x* value, maintaining the continuity and dependency of the system's state over time.

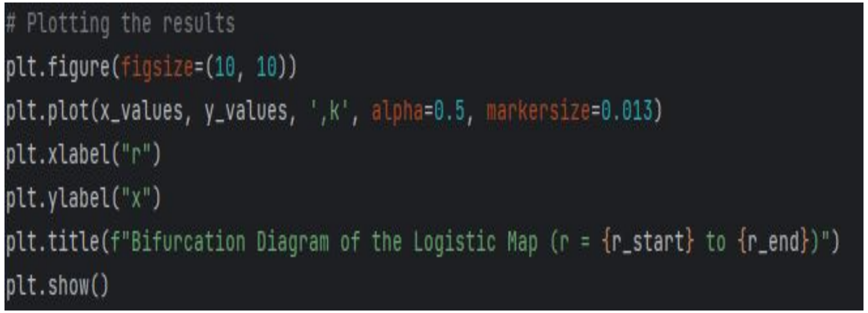


```
# Iterate through the logistic map for each value of r
for i in range(1, len(x_values)):
    y_values[i] = logistic_map(x_values[i], y_values[i - 1])
```

Creating the Bifurcation Diagram with Matplotlib

Setting up the plot:
We begin by creating a figure with plt.figure(figsize=(10, 10)), specifying the size of the plot.

The plt.plot function is then used to draw the diagram. Here, x_values (representing different *r* values) and y_values (corresponding system states) are plotted.



```
# Plotting the results
plt.figure(figsize=(10, 10))
plt.plot(x_values, y_values, ',k', alpha=0.5, markersize=0.013)
plt.xlabel("r")
plt.ylabel("x")
plt.title(f"Bifurcation Diagram of the Logistic Map (r = {r_start} to {r_end})")
plt.show()
```

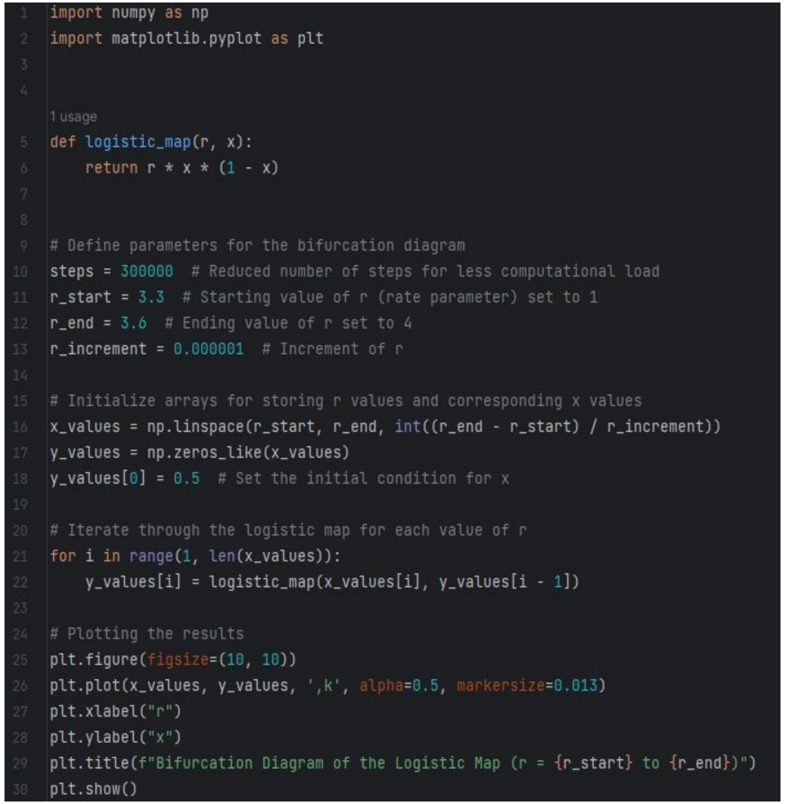
Customizing the plot:
The arguments ',k', alpha=0.5, and markersize=0.013 in plt.plot are used to define the appearance of the plot points. ',k' sets the color to black, alpha adjusts the transparency, and markersize controls the size of each point.

Labels for the x-axis and y-axis are set with plt.xlabel("r") and plt.ylabel("x").

The title of the plot is set with plt.title.

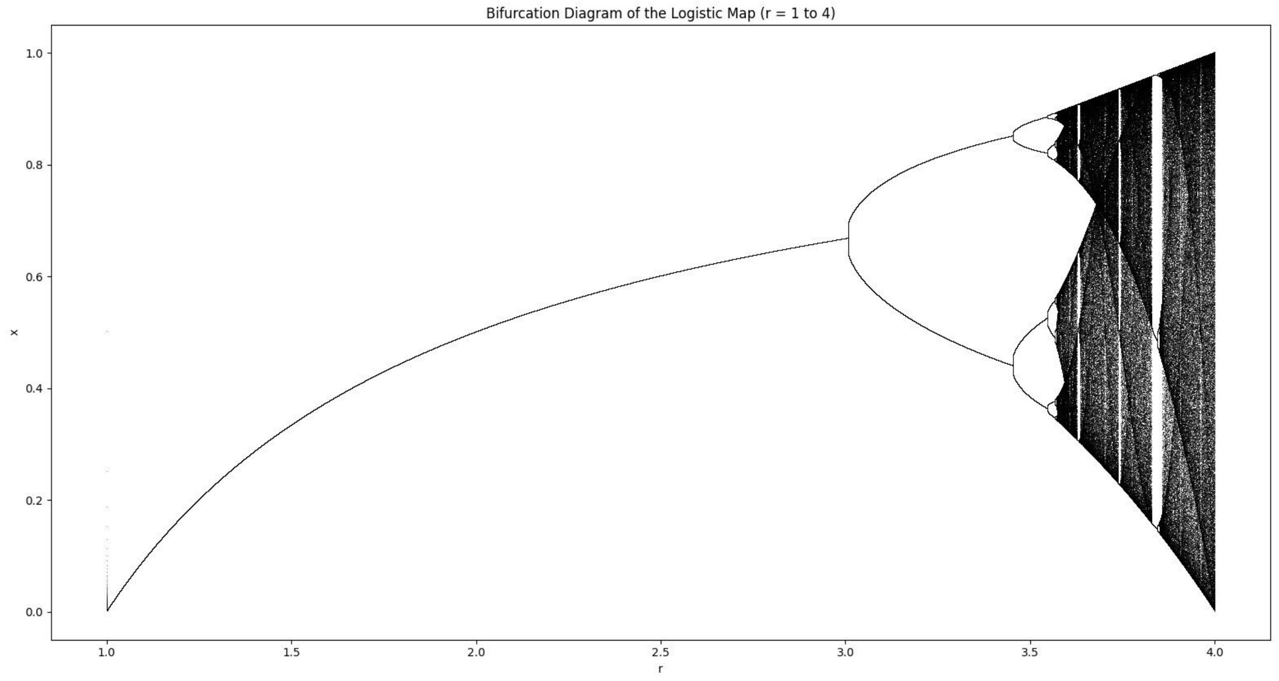
Finally, plt.show() is called to display the completed bifurcation diagram, showcasing the logistic map within the specified *r* interval.

Code



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


1 usage
def logistic_map(r, x):
    return r * x * (1 - x)


# Define parameters for the bifurcation diagram
steps = 300000  # Reduced number of steps for less computational load
r_start = 3.3  # Starting value of r (rate parameter) set to 1
r_end = 3.6  # Ending value of r set to 4
r_increment = 0.000001  # Increment of r

# Initialize arrays for storing r values and corresponding x values
x_values = np.linspace(r_start, r_end, int((r_end - r_start) / r_increment))
y_values = np.zeros_like(x_values)
y_values[0] = 0.5  # Set the initial condition for x

# Iterate through the logistic map for each value of r
for i in range(1, len(x_values)):
    y_values[i] = logistic_map(x_values[i], y_values[i - 1])

# Plotting the results
plt.figure(figsize=(10, 10))
plt.plot(x_values, y_values, ',k', alpha=0.5, markersize=0.013)
plt.xlabel("r")
plt.ylabel("x")
plt.title(f"Bifurcation Diagram of the Logistic Map (r = {r_start} to {r_end})")
plt.show()
```

Bifurcation Diagram Output


Bifurcation Diagram of the Logistic Map (r = 1 to 4)
x
r