ATRACTOR LOGÍSTICO – GEOMETRÍA COMPUTACIONAL – 2023

PABLO FERNÁNDEZ DEL AMO

1. INTRODUCCIÓN:

En esta práctica se muestran los resultados del estudio de la convergencia del sistema dinámico no lineal definido por la sucesión de la función logística, i.e., la sucesión $x_{n+1} = f(x_n)$, con $f(x) = rx(1-x)$. Este ejemplo ofrece una visión fascinante de la complejidad inherente en dichos sistemas.

2. MATERIAL USADO:

Para resolver el problema se utilizó el algoritmo visto en clase para el cálculo de conjuntos atractores. Primero calculamos la órbita, siendo el número de elementos pertenecientes a esta órbita delimitado por la amplitud del nuevo conjunto de elementos, ya que cuando esta sea menor que el épsilon o tolerancia escogida (en este caso 0,0005), pararemos. Con esta órbita hallaremos su periodo para calcular el valor aproximado de los elementos del conjunto límite junto a sus errores. Para decidir si el conjunto seleccionado es atractor, se estudiará su estabilidad.
Aplicando la teoría vista en los apuntes de clase, y utilizando Python como lenguaje junto a sus librerías asociadas de cálculo científico como Numpy y de visualización como Matplotlib.plot, implementaremos este algoritmo para obtener los siguientes resultados.
Finalmente, para el apartado 2, se ha utilizado la librería Random con el fin de obtener el x_0 a partir de una distribución uniforme sobre el intervalo (0,1)

3. RESULTADOS:

i) Encuentra dos conjuntos atractores diferentes para $r \in (3, 3.544)$ con $x \in (0, 1)$. Estima los valores de sus elementos con el correspondiente intervalo de error.

Hemos probado de manera aleatoria distintos valores de r y x_0, obteniendo los dos siguientes conjuntos atractores, con sus elementos acotados por un intervalo de error:

- Con $r = 3.35$ y $x_0 = 0.79$ se tiene que el conjunto atractor cuenta con únicamente los siguientes dos elementos: *(Ilustración 1)*

$$0{,}4650901092632805 \pm 4{,}659161945141932 \cdot 10^{-12}$$
$$0{,}8334173534213137 \pm 1{,}0896838986695911 \cdot 10^{-12}$$

- Con $r = 3.1$ y $x_0 = 0.9$ se tiene que el conjunto atractor cuenta con únicamente los siguientes dos elementos: *(Ilustración 2)*

$$0{,}5580141275575399 \pm 1{,}6364167798599283 \cdot 10^{-09}$$
$$0{,}7645665185229874 \pm 9{,}976250936460929 \cdot 10^{-10}$$

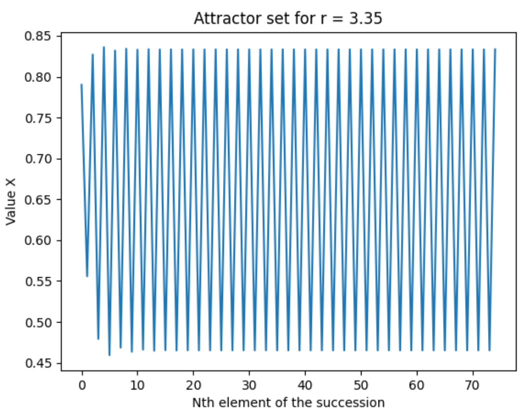

Ilustración 1

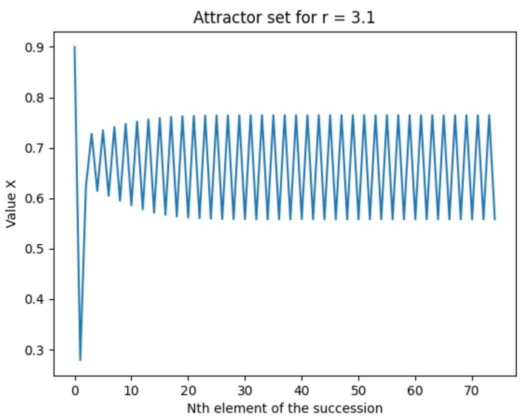

Ilustración 2

ii) Estima los valores de r ∈ (3.544, 4), junto con su intervalo de error, para los cuales el conjunto atractor tiene 8 elementos.
Obtén algún ejemplo concreto de conjunto atractor final
Los valores pedidos de r, según nuestra tolerancia definida previamente (0,0005), un épsilon que estima su intervalo de error (0,001) y un valor inicial a x_0=0,592 son los siguientes:

- Todos los valores pertenecientes al intervalo [3,5409999999999403, 3,564999999999938]
- 3,5699999999999372 ± 0,001
- 3,8019999999999117 ± 0,001
- 3,889999999999902 ± 0,001
- 3,960999999999894 ± 0,001

Visualizado en la *Ilustración 3*, donde los puntos verdes son los valores de conjuntos atractores finales para cada r, y los rojos aquellos pertenecientes a conjuntos atractores finales de 8 elementos.

A modo de ejemplo, el conjunto atractor final para r = 3.5419999999999403 y x_0= 0,592 es:

1. 0.36342416462116806 ± 6.724399447882723·10^{-05},
2. 0.36478816890888527 ± 7.627902586748725·10^{-05}
3. 0.5211099796598387 ± 0.00016598269973822077
4. 0.5242346499193682 ± 0.00016304005930301013
5. 0.8194312198970001 ± 6.507479709305652·10^{-05}
6. 0.8207443085162293 ± 7.304253968687391·10^{-05}
7. 0.8834197187347029 ± 2.8084587356036472·10^{-05}
8. 0.8839215741435172 ± 2.4723983777508174e·10^{-05}

Visualizado en la *Ilustración 4*.

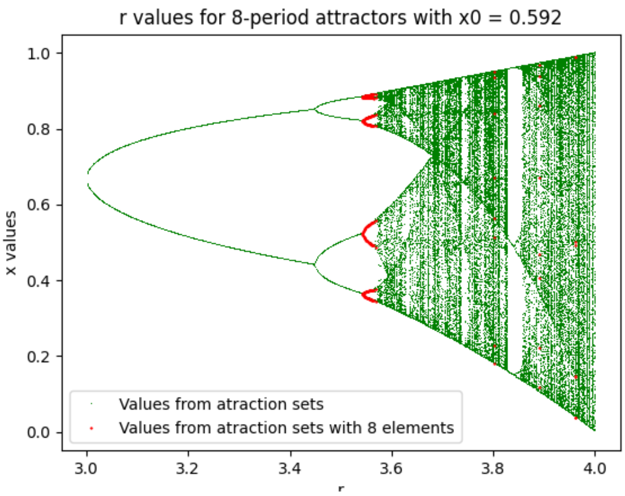

Ilustración 3

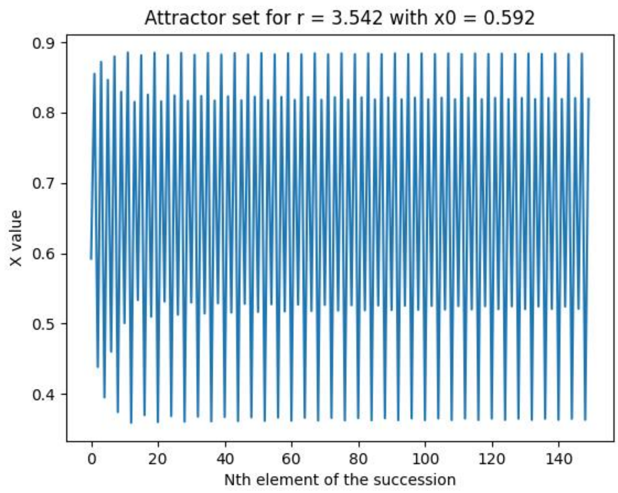

Ilustración 4

4. CONCLUSIÓN:

A través de esta práctica hemos estudiado la existencia y estabilidad de conjuntos atractores para distintos valores iniciales y del parámetro r. Además hemos encontrado aquellos valores de r para un valor inicial aleatoria con un conjunto atractor final de 8 elementos, aportando todos estos resultados con la máxima precisión posible junto a sus respectivos intervalos de error, respondiendo correctamente a las dos cuestiones que se nos planteaban.
Además, ejecutando el archivo .py asociado a esta práctica, hemos obtenido conjunto diferente para la primera pregunta.

5. **ANEXO:**

Código de Python utilizado:

```python
"""
This program performs computations related to the Logistic Map.
Author: PABLO FERNÁNDEZ DEL AMO
UNIT: GEOMETRÍA COMPUTACIONAL

EXTRA PRACTICE 1
"""

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

# Tolerance for numerical comparison
tolerance = 0.0005

def logistic_function(x, r):
    """
    Logistic function parameterized by r
    Args:
    x: input value
    r: rate parameter
    Returns:
    result of logistic function
    """
    return r*x*(1-x)

def numerical_equals(x, y, epsilon=0.001):
    """
    Check if two numbers are approximately equal within a tolerance
    Args:
    x, y: numbers to compare
    epsilon: tolerance for comparison
    Returns:
    True if |x - y| < epsilon, else False
    """
    return abs(x - y) < epsilon

def compute_orbit(initial_point, r, f, initial_size):
    """
    Compute the orbit of a point under a given function.
    Args:
    initial_point: starting point
    r: parameter for the function
    f: function to apply
    initial_size: initial size of the orbit
    Returns:
    orbit: array of points in the orbit
    """
    amplitude = 0
    orbit = np.empty(initial_size)
    x = initial_point
```

```python
    for i in range(initial_size):
        orbit[i] = x
        x = f(x, r)

    prev_amplitude = amplitude
    amplitude = np.max(orbit) - np.min(orbit)

    while not numerical_equals(amplitude, prev_amplitude, tolerance):
        for i in range(initial_size):
            orbit = np.append(orbit, x)
            x = f(x, r)
        prev_amplitude = amplitude
        amplitude = np.max(orbit[-initial_size:]) - np.min(orbit[-initial_size:])

    return orbit

def compute_period(orbit, epsilon=0.001):
    """
    Compute the period of an orbit
    Args:
    orbit: array of points in the orbit
    epsilon: tolerance for comparing points
    Returns:
    period: number of iterations between repeated points
    """
    N = len(orbit)
    for i in np.arange(2, N-1, 1):
        if numerical_equals(orbit[N-1], orbit[N-i]) :
            break
    return i - 1

def check_stability(initial_point, r, period, limit_set, epsilon, initial_size):
    """
    Check if a limit set is stable
    Args:
    initial_point: starting point
    r: rate parameter
    period: period of the limit set
    limit_set: array of points in the limit set
    epsilon: tolerance for comparison
    initial_size: initial size of the orbit
    Returns:
    True if the limit set is stable, else False
    """
    for modified_initial_point in np.arange(-10*epsilon + initial_point, 10 * epsilon + 
initial_point, epsilon):
        modified_orbit = compute_orbit(modified_initial_point, r, logistic_function, 
initial_size)
        modified_period = compute_period(modified_orbit, epsilon)

        if period != modified_period: # not sharing the same period means not having the same 
long term behaviour, implying inestability
            return False

        modified_limit_set = np.sort(modified_orbit[modified_orbit.size-modified_period:])
```

```
        difference = np.max(np.absolute(limit_set - modified_limit_set))
        max_difference = np.max(difference)


        if (max_difference >= epsilon):  # if the difference in the limit set is bigger than
our tolerance threshold, it is not stable
            return False
    return True

def check_bifurcations(initial_point, r, delta):
    plt.figure(6, 6)
    for r in np.arange(-10 * delta, 10 * delta, delta):
        V0, _, _ = find_attractor_set(r, initial_point, tolerance)
        limit_set = [V0[i][0] for i in range(len(V0))]
        plt.plot([r] * len(V0), limit_set, 'ro', markersize=1)

    plt.xlabel = 'r'
    plt.ylabel = 'V_0'
    plt.axvline(x=r, ls="--")
    plt.show()

def find_attractor_set(r, initial_point, epsilon):
    """
    Find an attractor set for a given r and initial point.
    Args:
    r: rate parameter
    initial_point: starting point
    epsilon: tolerance for comparison
    Returns:
    limit_set, orbit, is_stable: the limit set, the full orbit, and a boolean indicating
stability
    """
    initial_size = 25
    orbit = compute_orbit(initial_point, r, logistic_function, initial_size)
    period = compute_period(orbit, epsilon)

    limit_set = [(orbit[orbit.size-period+i],abs(orbit[orbit.size-period+i] -
orbit[orbit.size-2*period+i])) for i in range(period)]

    limit_set.sort()
    is_stable = check_stability(initial_point, r, period, [element[0] for element in
limit_set], epsilon, initial_size)

    return limit_set, orbit, is_stable

def apartadoI():
    """
    Compute and plot three attractor sets for given r and initial point values.
    """
    print("Section 1:")
    r_values = [3.4, 3.35, 3.1]
    initial_points = [0.5, 0.79, 0.9]

    for i in range(len(r_values)):
```

```
        print(f"Atractor set {i}: \nr = {r_values[i]}, initial point = {initial_points[i]}
\n")
        limit_set, orbit, is_stable = find_attractor_set(r_values[i], initial_points[i],
tolerance)

        if is_stable:
            for (x, epsilon) in limit_set:
                print(f"{x} ± {epsilon}")

            plt.title(f"Attractor set for r = {r_values[i]}")
            plt.plot(orbit)
            plt.xlabel("Nth element of the succession")
            plt.ylabel("Value X")
            plt.savefig(f"atractor_set{i}.png")
            plt.show()
        else:
            print("No stable set for r: {r_values[i]} and x_0: {initial_points[i]} found")


def apartadoII():
    """
    Estimate r values for which the attractor set has 8 elements.
    """
    print("Section 2:")

    initial_point = random.uniform(0, 1)
    r_error = 0.001
    counter = 0

    r_values = np.arange(3.001, 4, r_error)
    all_r_values = []
    f_r_values = []
    r_8elems = []
    atractor_set8 = []


    for r in r_values:
        limit_set0, orbit, _ = find_attractor_set(r, initial_point, tolerance)
        limit_set = [limit_set0[i][0] for i in range(len(limit_set0))]

        all_r_values.extend([r]* len(limit_set))
        f_r_values.extend(limit_set)


        if len(limit_set) == 8:

            for element in limit_set:
                r_8elems.append(r)
                atractor_set8.append(element)

            if counter == 0:
                example_values = limit_set0
                example_r = r
                example_orbit = orbit
                counter = 1
```

```
    plt.title(f"Attractor set for r = {round(example_r,4)} with x0 = {round(initial_point, 3)}")
    plt.plot(example_orbit)
    plt.xlabel("Nth element of the succession")
    plt.ylabel("X value")
    plt.savefig("attractor_set_8_elements.png")
    plt.show()

    plt.title(f"r values for 8-period attractors with x0 = {round(initial_point, 3)}")
    plt.plot(all_r_values, f_r_values, 'g,', markersize=0.01,label = 'Values from atraction sets')
    plt.plot(r_8elems, atractor_set8, 'r+', markersize=1.5, label = 'Values from atraction sets with 8 elements')
    plt.xlabel("r")
    plt.ylabel("x values")
    plt.legend()
    plt.savefig("AtractionSetsII.png")
    plt.show()

    print(f"The atractor set with x0 = {round(initial_point, 3)}  has 8 elements if r is:")
    r_8elems = list(set(r_8elems))
    r_8elems.sort()
    for r in r_8elems:
        print(f"{r} ± {r_error}")


    print(f"Example of an attractor set with x0 = {round(initial_point, 3)} and r = {example_r}:")
    for (x, epsilon) in example_values:
        print(f"{x} ± {epsilon}")


if __name__ == '__main__':
    apartadoI()
    apartadoII()
```