

Práctica 1 - GCOMP

Celia Rubio Madrigal

9 de febrero de 2022

Índice

1. Introducción	2
2. Material usado y metodología	2
2.1. Apartado <i>i</i>)	2
2.2. Apartado <i>ii</i>)	2
3. Resultados y conclusiones	3
3.1. Apartado <i>i</i>)	3
3.2. Apartado <i>ii</i>)	3
4. Código	4

1. Introducción

En esta práctica analizamos la convergencia de un sistema dinámico no lineal. En este caso, la transformación asociada viene dada por la función logística: $f_r(x) = r \cdot x \cdot (1 - x)$. Se usa para representar el crecimiento de poblaciones cuando hay competición entre varias especies.

Esta función es dependiente de un parámetro $r \in \mathbb{R}$, por lo que el sistema cambia de comportamiento al modificar este parámetro. Veremos cómo son las cuencas de atracción para algunos valores de r , y cuáles tienen un tamaño determinado.

2. Material usado y metodología

Vamos a tomar un margen de error de $\varepsilon = 0,001$ para todos los apartados. Además, en general, tomaremos el estado inicial $x_0 = 0,200$. Tomamos $N_0 = 200$ como nuestra máxima capacidad de cómputo; supondremos que más allá perdemos precisión por errores de la computadora. También consideramos periodos de, como máximo, $N = 20$.

Los cálculos se realizarán en un *script* de Python, adjuntado en la sección 4.

2.1. Apartado i)

En primer lugar, tomaremos dos valores del parámetro r en $(3, 3,544)$. Para cada uno, calculamos la órbita de ese sistema (hasta el término N_0), aplicando la función logística iterativamente.

Después, calculamos hasta qué paso de iteración —o tiempo transcurrido— la órbita empieza a mantenerse estable. Nos quedaremos con la subórbita de tamaño N a partir de $m \in \mathbb{N}$ tal que su diámetro y el de la subórbita siguiente no difieren de más de ε . Es decir, si $D(S) = \max(S) - \min(S)$:

$$|D(\{f_r^i(x_0)\}_{i=m-N}^m) - D(\{f_r^i(x_0)\}_{i=m}^{m+N})| < \varepsilon$$

Calculamos su conjunto atractor. Buscamos el periodo de la órbita: el primer $1 < p \in \mathbb{N}$ tal que:

$$|f_r^{m-1}(x_0) - f_r^{m-p}(x_0)| < \varepsilon$$

El conjunto atractor serán los p últimos valores de la subórbita calculada hasta el paso m .

Calcularemos el intervalo de error del valor de la r tomando valores cercanos a los escogidos y asegurándonos de que el conjunto atractor permanece estable. Es decir, si el conjunto atractor es $V_r(x_0)$, encontraremos el $\delta > 0$ tal que, para $0 < d < \delta$:

$$|V_r(x_0) - V_{r \pm d}(x_0)|_\infty < \varepsilon$$

Además veremos si cambiando el estado inicial x_0 también permanece estable. Encontraremos el $\delta > 0$ para el que, si $0 < d < \delta$:

$$|V_r(x_0) - V_r(x_0 \pm d)|_\infty < \varepsilon$$

2.2. Apartado ii)

En segundo lugar, buscamos los valores de r tales que su cuenca de atracción es igual a 8.

Para ello, calculamos para cada valor $r \in (3,544, 4)$ —tomándolos de ε en ε — su conjunto de atracción, y comprobamos si su tamaño es 8.

Si lo es, comprobamos que al cambiar el estado inicial $x_0 \pm \varepsilon$ el conjunto atractor sigue siendo el mismo.

3. Resultados y conclusiones

3.1. Apartado *i)*

Tomaremos $r = 3,140$ y $r = 3,500$.

Para $r = 3,140 \pm 0,003$, el conjunto atractor calculado es $[0.538, 0.780] \pm \varepsilon$. Hemos calculado que, para $\delta = 0,003$, $r + \delta$ da un conjunto atractor de $[0.537, 0.782] \pm \varepsilon$.

Para $r = 3,500 \pm 0,002$, el conjunto atractor calculado es $[0.383, 0.501, 0.827, 0.875] \pm \varepsilon$. Hemos calculado que, para $\delta = 0,002$, $r + \delta$ da un conjunto atractor de $[0.382, 0.502, 0.827, 0.875] \pm \varepsilon$.

En ambos casos, la elección de estado inicial $x_0 \in (0, 1)$ es irrelevante, ya que es suficientemente estable. También hemos comprobado que el mayor r tiene un intervalo de error $\pm \delta$ menor. Por tanto, el conjunto atractor se ve modificado más rápidamente.

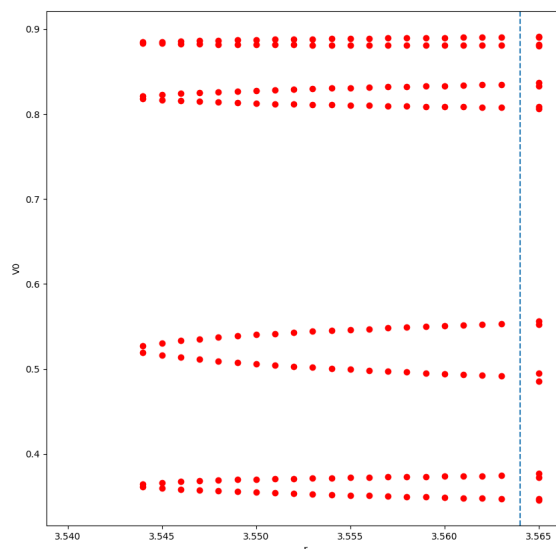
Entre ambos puntos escogidos hay al menos un punto de bifurcación, ya que el tamaño de la primera cuenca es 2 y la segunda es de 4.

3.2. Apartado *ii)*

Hemos calculado, para $r \in (3,544, 3,563) \pm \varepsilon$, que todas sus cuencas de atracción tienen 8 elementos. Además, hemos comprobado que con $x_0 = 0,200 \pm \varepsilon$ sus cuencas de atracción se mantienen estables.

Por ejemplo, para $r = 3,550 \pm \varepsilon$ se tiene el conjunto $[0.355, 0.370, 0.506, 0.540, 0.813, 0.828, 0.882, 0.887] \pm \varepsilon$. A la derecha se encuentra la gráfica de cada conjunto para cada r calculado.

A partir de $r = 3,565 \pm \varepsilon$ el conjunto de atracción ya no tiene 8 elementos sino 16, por lo que ha habido un punto de bifurcación. Después, no volvemos a encontrar valores de r con cuencas de atracción de tamaño 8.



Con este experimento hemos analizado cómo se comporta el sistema dinámico logístico para algunos valores de su parámetro r . Hemos visto algunas de sus características en los intervalos de $(3,1, 3,544)$ y $(3,544, 4)$, calculando sus conjuntos atractores y midiendo las cotas de error de nuestros cálculos.

4. Código

Práctica 1 de Celia Rubio Madrigal

```
import numpy as np

""" Funciones """

# Función logística
# R es global
def logistica(x):
    return R * x * (1 - x)

# Calcula la subórbita de f desde x0 aplicada n veces
# Mejorada a O(n)
def orbita(x0, f, n):
    orb = np.empty([n])
    orb[0] = f(x0)
    for i in range(n - 1):
        orb[i + 1] = f(orb[i])
    return orb

# Calcula el m tal que el diámetro de la órbita no cambia más que epsilon
# desde m hasta m+N, donde N es una constante global predefinida
def tiempo_transitorio(orb, epsilon=0.001):
    n = len(orb)
    m = 0
    next_sup, next_inf = max(orb[N : 2 * N]), min(orb[N : 2 * N])
    while m + N <= n:
        sup, inf = max(orb[m : m + N]), min(orb[m : m + N])
        if abs((sup - inf) - (next_sup - next_inf)) < epsilon:
            return m + N
        m += N
        next_sup, next_inf = sup, inf
    return n

# Calcula el periodo de la subórbita, suponiendo que el tiempo transitorio ya es estable
def periodo(suborb, epsilon=0.001):
    n = len(suborb)
    for i in range(1, n):
        if abs(suborb[n - 1] - suborb[n - i - 1]) < epsilon:
            return i
    return n
```

```

# Calcula la cuenca de atracción de f desde x0
def atrac(x0, f, epsilon=0.001):
    orb = orbita(x0, f, NO)
    m = tiempo_transitorio(orb, epsilon)
    suborb = orb[-m:]
    p = periodo(suborb, epsilon)
    v0 = np.sort(suborb[-p:])
    return v0

eps = 0.001 # predefinido el error
NO = 200 # nuestra capacidad de cómputo máxima
N = 50 # sabemos de antemano que nuestro conjunto será menor
X0 = 0.2

""" Apartado i) """

# Comprueba si dos vectores son iguales con precisión de epsilon
def equals_vectors(v1, v2, epsilon=0.001):
    if len(v1) != len(v2):
        return False
    for i, j in zip(v1, v2):
        if abs(i - j) >= epsilon:
            return False
    return True

# Comprueba la estabilidad (moviendo x0) y las bifurcaciones (moviendo R)
def test_v0(v0, epsilon=0.001):
    global R
    r = R
    print(v0)

    # Error de x0: en estos casos, todos los valores de x0 dan el mismo
    # resultado
    for x0 in np.arange(epsilon, 1, epsilon):
        v1 = atrac(x0, logistica, epsilon)
        if not equals_vectors(v0, v1, epsilon):
            print("X0", x0, "no estable")

    # Error de R: mover R hasta que no dé el mismo resultado
    for delta in np.arange(epsilon, 1, epsilon):
        R = r + delta
        v1 = atrac(X0, logistica, epsilon)
        if not equals_vectors(v0, v1, epsilon):
            print("R", R, "es distinto (delta=", delta, ")")
            print("\t", v0)
            print("\t", v1)
            break

```

```

    R = r - delta
    v2 = atrac(X0, logistica, epsilon)
    if not equals_vectors(v0, v2, epsilon):
        print("R", R, "es distinto (delta=", delta, ")")
        print("\t", v0)
        print("\t", v2)
        break
    R = r

print("-" * 10)
print("Conjunto atractor a)")

R = 3.14
v0 = atrac(X0, logistica, eps)
test_v0(v0, eps)

print("-" * 10)
print("Conjunto atractor b)")

R = 3.5
v0 = atrac(X0, logistica, eps)
test_v0(v0, eps)

""" Apartado ii) """
print("-" * 10)

rss = []
vss = {}

first = True
for r in np.arange(3.544, 4, eps):
    R = r
    v0 = atrac(X0, logistica, eps)

    # Primero, calculo si la cuenca de atracción tiene 8 elementos
    if len(v0) == 8:
        # Y después, si al mover x0 con error epsilon se mantiene el resultado
        test = True
        v1 = atrac(X0 + eps, logistica, eps)
        if not equals_vectors(v0, v1, eps):
            test = False
        v1 = atrac(X0 - eps, logistica, eps)
        if equals_vectors(v0, v1, eps):
            rss += [r]
            vss[r] = v0
            print("R", R, "\t", v0)

```

```

elif first and len(v0) != 8:
    first = False
    rss += [r]
    vss[r] = v0
    print("NO R", R, "\t", v0)

""" Gráfica """

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))

for r in rss:
    n = len(vss[r])
    plt.scatter([r] * n, vss[r], color="red")
plt.xlabel("r")
plt.ylabel("V0")

plt.axvline(x=3.564, ls="--")
plt.savefig("grafica.png")

```