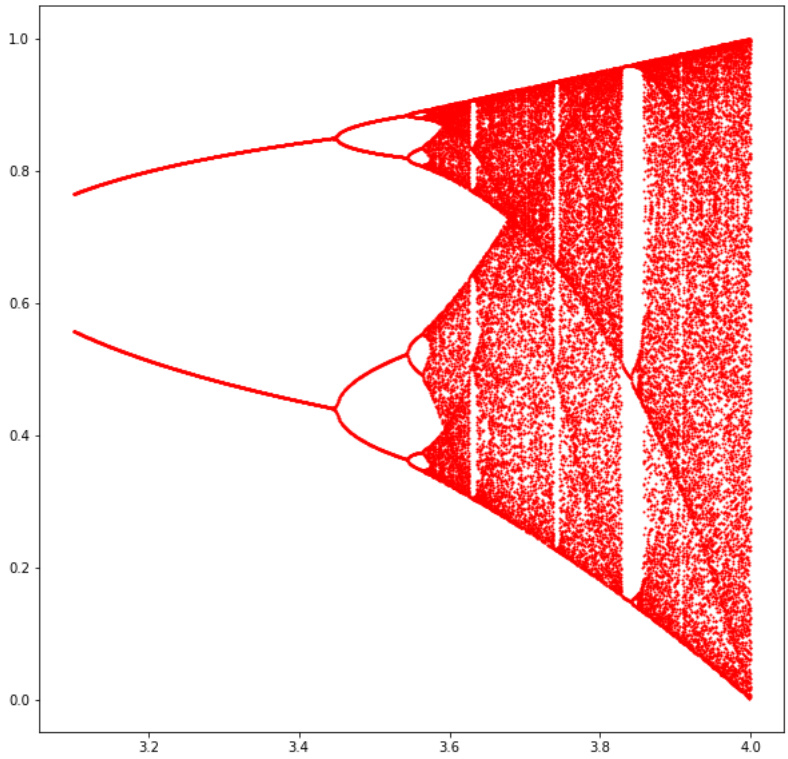
PRÁCTICA 1: Atractor Logístico

Geometría Computacional

Belén Sánchez Centeno
belsan05@ucm.es

16 de febrero de 2022



Date Performed: 26 de enero y 2 de febrero de 2022
Code Partner: Martín Fernández de Diego

1. Introducción

Sea un sistema dinámico discreto $x_{n+1} = f(x_n)$ dominado por la función logística $f(x) = rx(1-x)$:

Primer Objetivo
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.

Segundo Objetivo
Estima los valores de $r \in (3{,}544, 4)$, junto con su intervalo de error, para los cuales el conjunto atractor tiene 8 elementos. Obtén algún ejemplo concreto de conjunto atractor final.

2. Material utilizado

Se resuelve el problema con un script de Python, en el que se usan funciones de implementación propia, de la plantilla proporcionada y de las siguientes bibliotecas: `numpy` y `random`. Quedan detalladas a continuación:

- `uniform`, de `random`, que devuelve un número aleatorio en punto flotante en el intervalo que se le indique.
- `órbita`, que, dado un punto x_0, una r, una función `f` dependiente de la r y un entero N, devuelve una lista con los elementos de la sucesión $f^i(x_0)$ con $0 \leq i < N$. Se ayuda de la función `fn`, con los mismos parámetros excepto porque se le pasa el entero i en vez de N, que devuelve el cálculo de aplicar recursivamente `f` a x_0 i veces con la r dada.
- `periodo`, que empieza recorriendo desde el final los elementos de una subórbita dada y devuelve la distancia mínima que hay entre el último y uno que se le repita. Se entiende por elementos repetidos cuando hay entre ellos menos de una diferencia $\mathcal{E} = 0{,}001$, lo que se comprueba con la función `igual`. En el caso de que no haya ningún punto lo suficientemente parecido, `periodo` devuelve -1.
- `atractor`, que recibe las variables x_0 y r, una función `f` y dos enteros N y N_{cola}. En primer lugar se calcula la órbita (con la función `órbita`) de longitud N, se sustraen los N_{cola} últimos elementos, que suponemos estables, y se pasan a la función `periodo`, que devuelve el tamaño de los ciclos en la subórbita final. Se devuelve una lista ordenada (con la función `sort` de `numpy`) de los elementos (tantos como período haya salido) que forman los ciclos en la subórbita, es decir, el conjunto atractor.
- `tiempo_transitorio`, que, dada una función `f`, las variables x_0 y r y un entero N, devuelve el mínimo m que cumpla la condición de recubrimiento del algoritmo. Esta función prueba con $m = N$ y con $2 * m$ en sucesivas iteraciones, hasta que encuentra una válida. Es decir, una m que, siendo $|U(S)|$ el tamaño del entorno mínimo que envuelve un conjunto S, verifica que $|U(\{x_n\}_{n=4m}^{16m})| \leq |U(\{x_n\}_{n=2m}^{4m})| \leq |U(\{x_n\}_{n=m}^{2m})|$. Se requiere otra función, `subórbita`, igual a `órbita` pero con un parámetro adicional, el entero M, que indica la iteración a partir de la cual se calcula la subórbita, quedando truncado el inicio; y de las funciones `max` y `min` de `numpy`, utilizadas para calcular el tamaño del recubrimiento.
- `error_x`, que devuelve una variable *delta* válida que delimite un entorno del valor inicial x_0 en el que el conjunto atractor dado a la función no varía, es decir, de puntos estables. Para ello, se prueba con $delta = 0{,}5$ (o la delta que devuelva la función `ajustar`, que evita que sature en los extremos del intervalo dado, en este caso el $[0,1]$) y con $delta/2$ en sucesivas iteraciones. Es necesario calcular `tiempo_transitorio` y `atractor` en $x_0 \pm$delta y comprobar si son puntos estables, viendo si tienen el mismo número de elementos y si entre ellos hay menos de una diferencia $\mathcal{E} = 0{,}001$, de nuevo con la función `igual`.

Adicionalmemte, se utiliza la librería `matplotlib.pyplot` para generar las gráficas que ilustran esta memoria.

3. Resultados

3.1. Primer objetivo

En primer lugar, la función `uniform` permite tomar un x_0 en el intervalo $[0, 1]$ y dos r en el intervalo $(3, 3{,}544)$. La órbita final es fuertemente dependiente del valor de la constante r, por lo que para un mismo x_0 y dos r se van a obtener dos conjuntos atractores diferentes. Para cada r se calcula el tiempo transitorio N, pasándole a la función `tiempo_transitorio` las variables x_0 y r elegidas previamente al azar, la función `logística` dada por el enunciado y un entero $N_0 = 100$. Dicha N, que puede seguir siendo 100 o ser un múltiplo mayor, se utiliza a continuación en la función `atractor` para obtener el conjunto que se estaba buscando. Se calcula también el intervalo de error en torno a x_0 con la función `error_x`.

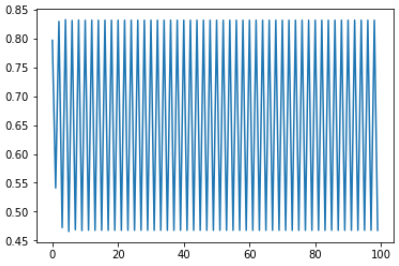


Figura 1: Cuenca de atracción de x0 = 0.7968166367329519 +- 0.2021833632670481 en r = 3.3410798197497327: [**0.46755217 0.83175226**]

Para evitar que ambos conjuntos conjuntos coincidan, se guarda el primero en una variable V que luego se compara con el segundo, para ver que no tienen la misma longitud o que se diferencian punto a punto en al menos $\mathcal{E} = 0, 001$, con la función `igual`.

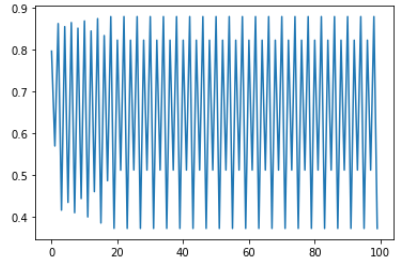


Figura 2: Cuenca de atracción de x0 = 0.7968166367329519 +- 0.2021833632670481 en r = 3.520678951076925: [**0.37278363 0.51242618 0.82319105 0.87962611**]

3.2. Segundo objetivo

Para encontrar los valores de r para los que el conjunto atractor tiene ocho elementos, se itera una variable entre 3,544 y 4 con valores espaciados en 0,001, con ayuda de la función `arange`, de `numpy`. La solución se expresa en intervalos de valores de r que cumplen la premisa.

Se han encontrado 5 intervalos en los que las cuencas de atracción son de 8 elementos, con una sensibilidad de 0.001, en x0 = 0.7471248146931895: [**3.544, 3.563999999999998**], [**3.580999999999996, 3.580999999999996**], [**3.6739999999999857, 3.6749999999999856**], [**3.880999999999963, 3.880999999999963**] y [**3.9389999999999565, 3.9389999999999565**].

Finalmente, se muestra una cuenca de atracción aleatoria que tenga ocho elementos, cogiendo la primera r para la que esto se verifique de entre una permutación aleatoria (con la función `permutation` de `random`) de sus valores posibles.

Ejemplo de cuenca de atracción de 8 elementos, de x0 = 0.7471248146931895 +- 0.2518751853068105, en r = 3.5469999999999997: [**0.35747664 0.36836523 0.51143244 0.5354508 0.81470815 0.8252886 0.88229228 0.8862864**]

4. Conclusión

Puede observarse que depende de r que se encuentre o no (si se acerca demasiado a una bifurcación) el conjunto atractor, y que el número de elementos que lo forman también aumenta a la vez que ese valor. Por otro lado, cuanto más aumentamos la sensibilidad $\mathcal{E} = 0, 001$, más intervalos diferentes de cuencas de atracción de 8 elementos obtenemos, pero más tarda el algoritmo en ejecutarse.

5. Anexo: Código utilizado

```
"""
PRÁCTICA 1: ATRACTOR LOGÍSTICO
Belén Sánchez Centeno
Martín Fernández de Diego
"""

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

"""
Dados una x, un cierto delta y un intervalo [a,b]
ajusta la eps para que x+eps y x-eps quepan en dicho intervalo
"""
# Restando epsilon evitamos que epsilon sature en los extremos
    indeseablemente
def ajustar(x,delta,a,b,epsilon=0.001):
    if x + delta > b:
        return b - x - epsilon
    if x - delta < a:
        return x - a - epsilon
    return delta

"""
Dados dos elementos
devuelve si son lo suficientemente parecidos
"""
def igual(x,y,epsilon=0.001):
    return np.max(abs(x - y)) < epsilon

"""
Dado un punto x y una r
devuelve f(x,r)
"""
def logistica(x,r):
    return r*x*(1-x)

"""
Dado un punto x0, una r, una función f y un entero n
devuelve f^n(x0,r)
"""
def fn(x0,r,f,n):
    x = x0
    for j in range(n):
        x = f(x,r)
    return(x)

"""
Dado un punto x0, una r, una función f y un entero N
devuelve una lista de f^i(x0) con 0 <= i < N
"""
def orbita(x0,r,f,N):
    orb = np.empty([N])
    for i in range(N):
        orb[i] = fn(x0,r,f,i)
    return(orb)
```

```
"""
Dado un punto x0, una r, una función f y dos enteros M y N con N > M
devuelve una lista de f^i(xm) con 0 <= i < N y xm = f^m(x0)
"""
def suborbita(x0,r,f,M,N):
    xm = fn(x0,r,f,M)
    orb = np.empty([N-M])
    for i in range(N-M):
        orb[i] = fn(xm,r,f,i)
    return(orb)

"""
Dado una subórbita
devuelve min(i) tal que |f^n(x0) - f^i(x0)| < epsilon
"""
# Encontramos el tamaño de los ciclos
def periodo(suborb):
    N = len(suborb)
    per = -1
    for i in np.arange(2,N-1,1):
        # Buscamos la distancia mínima a partir de la cual se vuelven a
            repetir puntos
        if igual(suborb[N-1],suborb[N-i]):
            per = i-1
            break
    return(per)

"""
Dada una función f y dos enteros N y N_cola
devuelve una lista ordenada con los per elementos de una subórbita
    final de N_cola respecto de una órbita de N elementos
"""
def atractor(x0,r,f,N,N_cola):
    orb = orbita(x0,r,f,N)
    ult = orb[-1*np.arange(N_cola,0,-1)]
    per = periodo(ult)
    V = np.sort([ult[N_cola-1-i] for i in range(per)])
    return V

"""
Dada una función f y un entero N
devuelve el mínimo m que cumpla la condición de recubrimiento del
    algoritmo
"""
# Encontramos un m a partir del que se puede intuir una cuenca de
    atracción
def tiempo_transitorio(x0,r,f,N):
    m = N
    recubierto = False
    while not recubierto:
        # Si no cumple la condición del algoritmo, no hay recubrimiento
            para ese tiempo transitorio
        for i in range(3):
            suborb = suborbita(x0,r,f,2**i*m,2**(2**i)*m) # m - 2m / 2m
                - 4m / 4m - 16m
            Dt_act = np.max(suborb) - np.min(suborb)
            # Condición del algoritmo
```

```
            if (i == 0) or (Dt_act <= Dt_ant and igual(Dt_act,Dt_ant)):
                # Guardamos el recubrimiento anterior
                Dt_ant = Dt_act
                recubierto = i == 2;
            else :
                # Aumentamos el tiempo transitorio
                m *= 2
                break
    return m

"""
Dada una función f, dos enteros N y N_cola que indican longitudes en la
     sucesión y el conjunto atractor
devuelve el mayor delta posible que define un entorno de atracción
    respecto al V
"""
def error_x(x0,r,f,N,N_cola,V,delta=0.5):
    delta = ajustar(x0,delta,0,1)
    estable = False
    while not estable:
        N_der = tiempo_transitorio(x0+delta,r,f,N)
        N_izq = tiempo_transitorio(x0-delta,r,f,N)
        V_der = atractor(x0+delta,r,f,N_der,N_cola)
        V_izq = atractor(x0-delta,r,f,N_izq,N_cola)
        # Comprobamos si es un punto estable
        if not ((len(V_der) == len(V) and len(V_izq) == len(V)) and (
            igual(V_der,V) and igual(V_izq,V))):
            delta /= 2
        else :
            estable = True
    return delta



# FORMATO
class Formato:
    BOLD = "\033[1m"
    RESET = "\033[0m"

# CONSTANTES
N0, N_cola = 100, 50

x0 = 0.7471248146931895
# rand.uniform(0,1)

# APARTADO i)
print("\n" + Formato.BOLD + "Apartado i)" + Formato.RESET)

i = 0
while i < 2:
    r = rand.uniform(3.000,3.544)
    # Calculamos un numero de iteraciones adecuado
    N = tiempo_transitorio(x0,r,logistica,N0)
    # Calculamos el conjunto atractor
    V = atractor(x0,r,logistica,N,N_cola)
    # Si los conjuntos obtenidos son distintos
    if i == 0 or not ((len(V) == len(V_ant)) and igual(V,V_ant)):
        # Calculamos el posible error
```

```
        err_x = error_x(x0,r,logistica,N,N_cola,V)
        # Mostramos
        print(i+1,"- Cuenca de atracción de x0 =",x0,"+-",err_x,"en r =
            ",r)
        print("  ->",V)
        V_ant = V
        i += 1

# APARTADO ii)
print("\n" + Formato.BOLD + "Apartado ii)" + Formato.RESET)

ext_izq, ext_der = [], []
err_r = 0.001
intervalo = False
for r in np.arange(3.544,4.000,err_r):
    # Calculamos el conjunto atractor
    V = atractor(x0,r,logistica,N0,N_cola)
    # Si no estamos en el intervalo y encontramos una cuenca de 8
        elementos
    if not intervalo and len(V) == 8:
        ext_izq.append(r)
        intervalo = True
    # Si estamos en el intervalo y encontramos una cuenca que no tiene
        8 elementos
    elif intervalo and len(V) != 8:
        ext_der.append(r-err_r)
        intervalo = False

print("  -",len(ext_izq),"intervalos de cuencas de atracción de 8
    elementos con una sensibilidad de",err_r,"en x0 =",x0)
print("  ->",end=' ')
for izq,der in zip(ext_izq,ext_der):
    print("[",izq,",",der,"]",end='')

# Tomamos un valor aleatorio del intervalo
for r in np.random.permutation(np.arange(3.544,4.000,0.001)):
    V = atractor(x0,r,logistica,N0,N_cola)
    if len(V) == 8:
        err_x = error_x(x0,r,logistica,N0,N_cola,V)
        print("\nEj: Cuenca de atracción de 8 elementos de x0 =",x0,"+-
            ",err_x,"en r =",r)
        print("  ->",V)
        break
```