PRÁCTICA 1: ATRACTOR LOGÍSTICO

Autor *Martín Fernández de Diego*
Compañera de código *Belén Sánchez Centeno*

1. INTRODUCCIÓN

A través del análisis del sistema dinámico discreto $x_{n+1} = f(x_n)$ dominado por la función logística $f(x) = rx(1-x)$, se pretenden resolver las dos siguientes cuestiones:

Apartado 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.**

Apartado ii) **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 USADO

A continuación, se desarrollan las funciones utilizadas.

2.1. Funciones principales

`logistica(x,r)` Dado un par x, r, devuelve la función logística.

`fn(x,r,f,n)` Dado un par x, r, una función f y un entero n, devuelve $f^n(x)$.

`orbita(x,r,f,N)` Dado un par x, r, una función f y un entero N, devuelve un vector con $f^i(x)$ para cada $0 \leq i \leq N$.

`suborbita(x,r,f,M,N)` Dado un par x, r, una función f y dos enteros M y N con $M \leq N$, devuelve un vector con $f^i(x)$ en cada posición $0 \leq i \leq N - M$.

`periodo(suborbita)` Dada una suborbita, devuelve el tamaño de los ciclos, si existen.

El algoritmo toma el último elemento de la cola y cuenta el número de iteraciones que tarda en encontrar uno igual.

`atractor(x,r,f,N,N_cola)` Dado un par x, r, una función f y dos enteros N y N_cola que indicará el tamaño de la órbita deseada y el número de elementos que debemos analizar respectivamente, devolverá un conjunto ordenado V con los valores de x que forman la cuenca de atracción.

El algoritmo calcula una órbita de N elementos con la función `orbita`, toma sus últimos N_cola elementos para asegurar cierta estabilidad en sus valores, halla el periodo a través de la función `periodo` y toma ese mismo número de elementos del final del vector de la órbita.

`tiempo_transitorio(x,r,f,N)` Dado un par x, r, una función f y un entero N que definirá el punto de partida, devuelve el mínimo entero m que cumpla $|U(\{x_n\}_{n=4m}^{16m})| \leq |U(\{x_n\}_{n=2m}^{4m})| \leq |U(\{x_n\}_{n=m}^{2m})|$ donde $|U(S)|$ es la medida extensiva del entorno mínimo que recubre a S.

El algoritmo toma inicialmente $m = N$ y calcula los conjuntos de la forma $U(\{x_n\}_{n=2^i m}^{2^{2^i} m})$ para $i = 0, 1, 2$ a través de la función `suborbita`. A continuación, calculamos la medida extensiva como la distancia euclídia entre máximo y mínimo del conjunto. Si no se cumple la condición descrita y, además, la diferencia entre estas medidas es superior a cierto ϵ, consideraremos que el conjunto no ha sido recubierto y volveremos a iterar el algoritmo con una m mayor.

`error_x(x,r,f,N,N_cola,V)` Dado un par x, r, una función f, dos enteros N y N_cola y un conjunto atractor V, devuelve un valor δ tal que $C(x) = C(x \pm \delta)$ donde $C(x)$ es la cuenca de atracción de x para cierta función.

El algoritmo, tras calcular los tiempos transitorios adecuados para $x \pm \delta$, halla los atractores en dichos puntos y comprueba si coinciden con el atractor inicial V. Si no coinciden, significará que en un entorno de radio δ de x la órbita no es estable y volvería a ejecutar el algoritmo con δ menor.

Hay que tener en cuenta que, para que $x \pm \delta$ pertenezcan a $[0,1]$, debemos truncar los valores de δ acorde al valor de x. Esto lo hacemos en la función auxiliar `ajustar`.

2.2. Funciones auxiliares

`ajustar(x,delta,a,b,epsilon)` Dado un par x, *delta*, un intervalo $[a,b]$ y *epsilon*, devuelve un *delta'* de forma que si $x\pm$*delta* no está contenido en el intervalo $[a,b]$, toma la menor distancia entre x y los extremos del intervalo.

Si un valor de $x\pm$*delta* coincide con los extremos del intervalo resultarán puntos con poco interés para el problema actual. Por ello, restamos *epsilon* al valor devuelto.

`igual(x,y,epsilon)` Dado un par x, y y cierto *epsilon*, devuelve si x es igual a y excepto por un valor *epsilon*.

2.3. Apartado i)

Decidimos los valores iniciales $N0 = 100$, $N\_cola = 50$ y cierta $x0 \in [0, 1]$. Un $N0$ lo suficientemente grande para estabilizar la órbita y que permita la extracción de al menos N_cola elementos.

Buscamos cada cuenca de atracción de manera que sean diferentes: Tomamos una $r \in (3, 3{,}544)$ independiente, calculamos el `tiempo_transitorio`, el conjunto `atractor` para ese tiempo y buscamos el intervalo de `error_x`.

2.4. Apartado ii)

Decidimos los valores iniciales $N0 = 100$, $N\_cola = 50$ y la misma $x_0 \in [0, 1]$.

Buscamos los subintervalos de $r \in (3{,}544, 4)$ donde se cumpla que el conjunto atractor posee exactamente 8 elementos: Recorremos el intervalo $[3{,}544, 4)$ con una precisión de $0{,}001$, calculamos el conjunto `atractor` (sin antes haber descubierto el `tiempo_transitorio` puesto que el perjuicio computacional en tiempo es superior al beneficio del resultado) y comprobamos si estamos en un punto de bifurcación acumulando los extremos de los intervalos en dos vectores. El punto de bifurcación se da cuando el número de elementos de la cuenca atractora cambia entre dos valores de r consecutivos.

3. RESULTADOS

Mostramos el resultado gráfico arrojado por cada apartado. También se mostrará en el anexo la salida de una ejecución.

3.1. Apartado i)

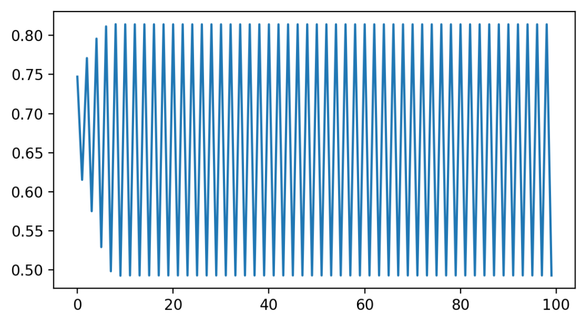

Fig. 1. Cuenca de atracción de periodo 2
$x_0 = 0{,}7471248146931895 \pm 0{,}2518751853068105$
$r = 3{,}256708942895666$
$[\mathbf{0{,}49303902, 0{,}81401943}]$

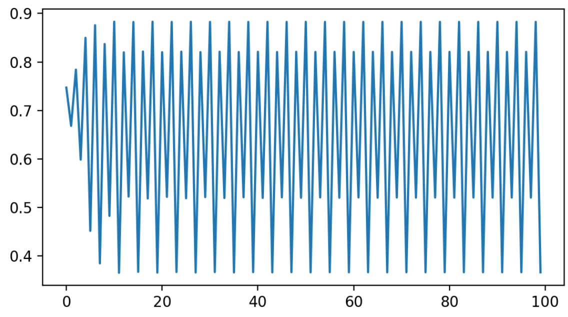

Fig. 2. Cuenca de atracción de periodo 4
$x_0 = 0{,}7471248146931895 \pm 0{,}2518751853068105$
$r = 3{,}5366375599918096$
$[\mathbf{0{,}36612864, 0{,}52016586, 0{,}8208127, 0{,}88272117}]$

3.2. Apartado ii)

En $x_0 = 0{,}7471248146931895$, con sensibilidad 0.001, hay 5 intervalos de cuencas de atracción de 8 elementos

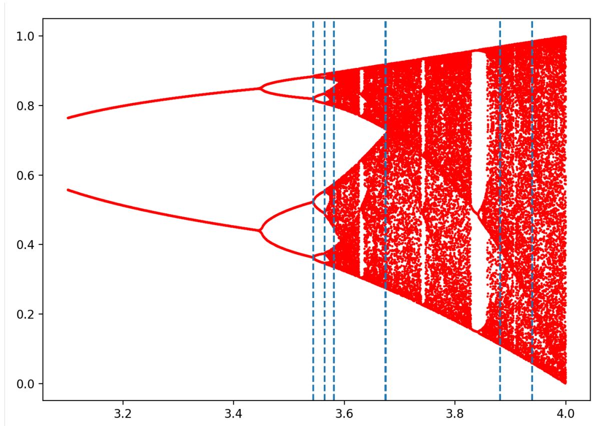

Fig. 3. Marcas de los puntos de bifurcación

4. CONCLUSIÓN

A lo largo del desarrollo, hemos utilizado el término *igual* sin darle importancia al *épsilon*. Este valor es muy importante: si es lo suficientemente grueso, puede detectar el periodo con menor número de iteraciones. Así que, encontrar una buena proporción iteraciones-*épsilon* mejoraría los resultados.

Por último, véase que a pesar de la clara existencia de más intervalos con atractores de 8 elementos, solo hemos conseguido detectar 5 con sensibilidad 0.001. Tan impreciso es este resultado que, si reducimos la sensibilidad a 0.0001, el número de intervalos es 28 y, si la volvemos a reducir a 0.00001, el número es 244.

5. ANEXO CON EL SCRIPT Y CÓDIGO UTILIZADO

5.1. Código

```
"""
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)
```

```python

"""
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
```

```python
        else :
            estable = True
    return delta



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

# CONSTANTES
N0, N_cola = 100, 50

x0 = 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
```

5.2. Ejecución

Apartado i)

```
1 - Cuenca de atracción de x0 = 0.7471248146931895 +- 0.2518751853068105 en r =
    3.256708942895666
  > [0.49303902 0.81401943]
2 - Cuenca de atracción de x0 = 0.7471248146931895 +- 0.2518751853068105 en r =
    3.5366375599918096
  > [0.36612864 0.52016586 0.8208127 0.88272117]
```

Apartado ii)

```
  - 5 intervalos de cuencas de atracción de 8 elementos con una sensibilidad de 0.001 en
     x0 = 0.7471248146931895
  > [ 3.544 , 3.5639999999999998 ][ 3.5809999999999996 , 3.5809999999999996 ][
    3.6739999999999857 , 3.6749999999999856 ][ 3.880999999999963 , 3.880999999999963 ][
     3.9389999999999565 , 3.9389999999999565 ]
Ej: Cuenca de atracción de 8 elementos de x0 = 0.7471248146931895 +- 0.2518751853068105
    en r = 3.5589999999999984
  > [0.34932508 0.37356269 0.49544104 0.55004222 0.80895039 0.83285442 0.88083747
    0.88967603]
```