Práctica 1: Atractor logístico

Guillermo Martín Sánchez

22/02/20

1 Introducción

El objetivo de esta práctica es el estudio del conjunto atractor $V_\infty = \lim_{n\to\infty} orb_n(x_0, f)$ del sistema dinámico discreto $x_{n+1} = f(x_n)$ con la aplicación logística $f(x) = rx(1-x)$ con $r \in R$ y $x \in D = [0,1]$.

2 Material usado

Para el cálculo del conjunto atractor hemos usado una simplificación del *Algoritmo 1.2.1. (Búsqueda de atractores)*:

Find V_0:

- **Input**: Una función f , $x_0 \in [0,1]$, $\epsilon > 0$ y $n \in \mathbb{N}$
- **Output**: $orb_n(x_0, f)$, $V_0 \subset D$ una estimación del conjunto atractor y $k = |V_0|$ el periodo de la órbita
- <u>Paso 1</u>: Cálculo de $orb_n(x_0, f)$. Para ello calculamos con $i \in \{0, ..., n-1\}$ $x_{i+1} = f(x_i)$
- <u>Paso 2</u>: Búsqueda del periodo k. Para ello comprobamos con $k \in \{1, ..., n\}$ cuál es el primero que cumple $|orb_n(x_0, f)[n-k] - orb_n(x_0, f)[n]| < \epsilon$
- <u>Paso 3</u>: Según si encontramos el periodo k:
  - <u>Paso 3a</u>: Si hemos encontrado tal k, devolver k y $V_0 = \{orb_n(x_0, f)[i] \quad | \quad i \in \{n-k, n-k+1, ..., n\}\}$
  - <u>Paso 3b</u>: Si no hemos encontrado tal k, devolver $k = -1$ y $V_0 = \{-1\}$

2.1 Ejercicio 1

Encuentra dos conjuntos atractores diferentes para $r \in (3, 3.5)$ con $x \in [0,1]$.

Para ello hemos ejecutado el algoritmo para $n = 1000$, $\epsilon = 10^{-5}$, $r = \{3, 3.11, 3.23, 3.34, 3.45\}$ y $x_0 = \{0.1, ..., 0.9\}$ y visto cuáles han sido los resultados. Hemos probado con varios $x_0 \in (0, 1)$ para comprobar que efectivamente V_0 no depende del valor de x_0. Los casos $x_0 = 0$ y $x_0 = 1$ se consideran por separado pues estos sí varían el valor de V_0 ($f(0) = 0$ y $f(1) = 1$).

2.2 Ejercicio 2

Estima el valor de $r \in (3, 4)$ para el cual el conjunto atractor tiene 8 elementos.

Para ello hemos ejecutado el Algoritmo para $n = 1000$, $\epsilon = 10^{-5}$, $r = \{3.50, 3.53, 3.56, 3.59, 3.61, 3.64, 3.67, 3.7, 3.8, 3.81, 3.83, 3.84, 3.86, 3.87, 3.89, 3.9\}$ y $x_0 = 0.5$. No hemos variado x_0 puesto que consideramos que el anterior ejercicio demuestra la invariancia de V_0 siempre que $x_0 \in (0, 1)$. Además hemos empezado con $r = 3.5$ ya que en el ejercicio anterior ya estudiamos el atractor para $r \in (3, 3.5)$

2.3 Error

Hemos añadido una función que nos aproxima el valor del conjunto atractor V_∞ con una precisión mayor y explícita. Para ello elegimos el percentil 90 de las diferencias $\Delta V = |V_i - V_{i+1}|$ con $i \in 0, .., 9$. En este caso, como el número de datos es 10 el percentil 90 equivale al valor que ocupa el noveno lugar una vez hemos ordenado ΔV de menor a mayor

Error:

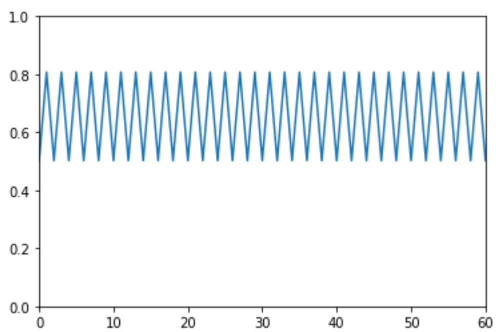
- **Input**: Una función f y la estimación inicial V_0 de su conjunto atractor V_∞
- **Output**: Una mejor estimación V_{10} y un error err tal que se espera que $V_\infty \in (V_{10} - err, V_{10} + err)$ con alta confianza.
- Paso 1: Cálculo de V_i con $i \in 1, .., 10$. Para ello calculamos con $i \in \{1, .., 10\}$ $V_i = f(V_{i-1})$ y las ordenamos de menor a mayor.
- Paso 2: Cálculo de las diferencias $\Delta V[i] = \max_{j=0,\dots,k} |V_i[j] - V_{i+1}[j]|$ con $i \in 0, .., 9$.
- Paso 3: Cálculo del error err. Para ello ordenamos ΔV de menor a mayor y asignamos $err = \Delta V[8]$
- Paso 4: Devolver V_{10} y err

Hemos probado a calcular el error err para $n = \{100, 1000, 10000\}$, $\epsilon = 10^{-5}$, $r = \{3.569\}$ y $x_0 = 0.5$. Hemos elegido este r porque, como se puede ver en la Figura 1, se encuentra justo en el límite donde el sistema dinámico empieza a comportarse de forma caótica y nos parecía interesante ver como se comporta el error cuando k es alto.

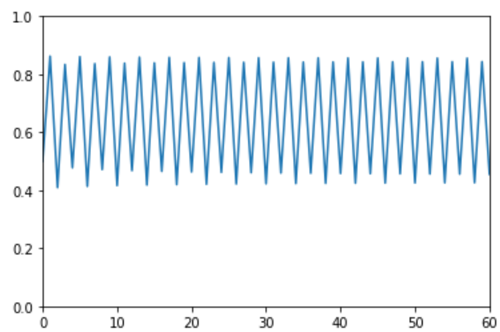
3 Resultados y discusión

3.1 Ejercicio 1

Ejecutando la correspondiente celda, observamos que para diferentes valores de x_0 no varía el valor de V_0 para un r dado. Además, respondiendo a la pregunta del ejercicio observamos que para diferentes valores de r obtenemos distintos V_0 con cardinalidad $k = 2$. De hecho, para $r = 3.45$ observamos que V_0 tiene cardinalidad $k = 4$. Bastaría coger cualquier par de valores de r para encontrar dos conjuntos atractores distintos. Por ejemplo, para $r = 3.23$ y $r = 3.45$ obtenemos $V_{10} = \{0.504, 0.806\} \pm 0$ y $V_{10} = \{0.434, 0.446, 0.847, 0.853\} \pm 3 \cdot 10^{-6}$



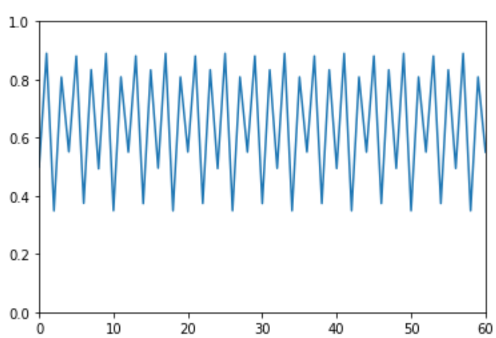
Con r = 3.23 obtenemos k = 2



Con r = 3.45 obtenemos k = 4

3.2 Ejercicio 2

Ejecutando la correspondiente celda, vemos que una posible respuesta al ejercicio es el valor $r = 3.54$ ya que tiene un periodo de $k = 8$. Es además curioso ver como después el conjunto se comporta de manera caótica salvo para $r = 3.83, 3.84, 3.87$ en el que encuentra un periodo de $k = 3, 6, 28$ respectivamente. Esto se debe a que, como muestra la Figura 1 , dentro de la zona caótica ($r > 3.56995$) de los valores de r, hay ciertos intervalos en los que vuelve a existir un periodo finito.



Con r = 3.56 obtenemos k = 8

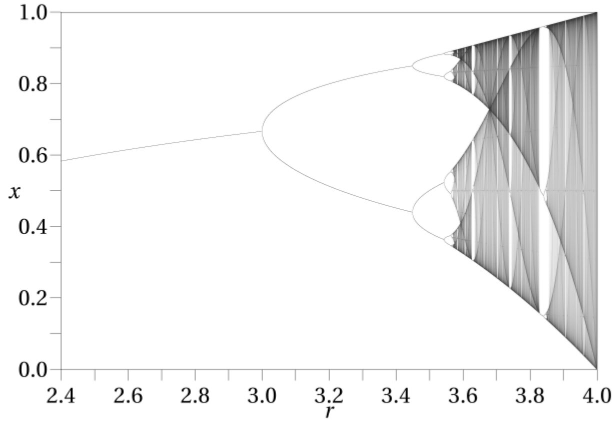


Figure 1: Gráfica de V_∞ para distintos valores de r

3.3 Error

Obtenemos los siguientes resultados:
- Para $n = 100$ obtenemos un error de $9 \cdot 10^{-6}$
- Para $n = 1000$ obtenemos un error de $6 \cdot 10^{-14}$
- Para $n = 10000$ obtenemos un error de 0

Como podemos observar, obtenemos el resultado previsto: a mayor n, menor error. De hecho, en este caso, para $n = 10000$ la estimación del error es tan pequeña, que *Python* lo redondea a 0.

4 Conclusión

Hemos visto que el cálculo de conjuntos atractores en algunos sistemas dinámicos (en concreto, en la función logística) es sencillo de hacer y nos da mucha información de como se comporta el sistema al cabo de cierto tiempo. También ha sido muy informativo ver cómo el comportamiento variaba con respecto a r de tener un sistema que convergía a un punto, a un sistema con un periodo finito, a incluso uno que se comporta de manera caótica.

5 Código

```
# -*- coding: utf-8 -*-
"""
Coursework 1: Logistic atractor
"""
import matplotlib.pyplot as plt
```

```python
import numpy as np

# Compute the orbit of n points of the function f from x0
def orbit(f,x0,n):
    orb = np.array([x0])
    for i in range(n):
        orb = np.append(orb,f(orb[i]))
    return orb;

# Compute the period of and orbit orb with difference epsilon
def period(orb,epsilon):
    n = len(orb)-1
    for k in range(1,n):
        if (abs(orb[n] - orb[n-k]) < epsilon):
            return k
    return -1

# Compute a better estimation (V10 +/- error) of the atractor set
# from an initial estimation V0 and the function f
def error(f,V0):
    Vant = V0
    Vant.sort()
    DeltaV = []
    for i in range(10):
        Vsig = f(Vant)
        Vsig.sort()
        DeltaV.append(max(abs(Vant - Vsig)))
        Vant = Vsig
    DeltaV.sort()
    return (Vant,DeltaV[8])

# Compute an estimation V0 of the atractor set of f
def find_V0(f,x0,n,epsilon):
    orb = orbit(f,x0,n)
    k = period(orb, epsilon)
    if (k == -1):
        return (orb,k,[-1])
    else:
        V0 = orb[-k:].copy()
        V0.sort()
        return (orb,k,V0)

#%%
"""
Set parameters manually and plot
"""
```

```
r = 3.56
x0 = 0.5

n = 100
epsilon = 0.001

logistic = lambda x: r * x * (1-x)

print("For r = :", r)
(orb,k,V0) = find_V0(logistic,x0,n,epsilon)
if (k == -1):
     print("- For x0 = {:.1f}".format(x0), "no period found smaller than",n)
else:
    (V10,err) = error(logistic,V0)
    print("- For x0 = {:.1f}".format(x0), "we get k =", k, "and V10 = ",
          [round(x,3) for x in V10], "+- {:.0e}".format(err))

plt.plot(orb)
plt.axis([0, n, 0, 1]);


#%%
"""
Exercise 1
"""
arr = np.linspace(3,3.45,5)

arX0 = np.linspace(0.1,0.9,9)

n = 1000
epsilon = 1e-5

for r in arr:
    logistic = lambda x: r * x * (1-x)
    print("For r = {:.2f}".format(r))
    for x0 in arX0:
        (orb,k,V0) = find_V0(logistic,x0,n,epsilon)
        if (k == -1):
            print("- For x0 = {:.1f}".format(x0),
                  "no period found smaller than",n)
        else:
           (V10,err) = error(logistic,V0)
           print("- For x0 = {:.1f}".format(x0), "we get k =", k, "and V10 = ",
                 [round(x,3) for x in V10], "+- {:.0e}".format(err))
#%%
"""
```

```
Exercise 2
"""
arr = np.linspace(3.5,3.7,8)
arr = np.append(arr,np.linspace(3.8,3.9,8))


x0 = 0.5

n = 1000
epsilon = 1e-5


for r in arr:
    logistic = lambda x: r * x * (1-x)
    (orb,k,V0) = find_V0(logistic,x0,n,epsilon)
    if (k == -1):
        print("- For r = {:.2f}".format(r), "no period found smaller than",n)
    else:
        (V10,err) = error(logistic,V0)
        print("- For x0 = {:.1f}".format(x0), "we get k =", k, "and V10 = ",
              [round(x,3) for x in V10], "+- {:.0e}".format(err))
#%%
"""
Error
"""
r = 3.569
x0 = 0.5

narr = [100, 1000, 10000]
epsilon = 1e-5

logistic = lambda x: r * x * (1-x)

for n in narr:
    (orb,k,V0) = find_V0(logistic,x0,n,epsilon)
    if (k == -1):
        print("- For n =",n, "no period found")
    else:
        (V10,err) = error(logistic,V0)
        print("- For n =",n, "we get k =", k, "and error = {:.0e}".format(err))
```