1. Introducción

En esta práctica estudiaremos el sistema dinámico discreto $x_{n+1} = f(x_n)$ dominado por la función logística $f(x) = rx(1-x)$. Buscaremos los conjuntos atractores para valores diferentes de r y x y después buscaremos el intervalo de valores de r para los cuales el conjunto atractor tiene un número determinado de elementos.

2. Material empleado

2.1. Encuentra dos conjuntos atractores diferentes para $r \in (3, 3, 5)$ con $x \in [0, 1]$

Para el primer apartado definimos las siguientes funciones:

- *logistica*: dados el parámetro r y un punto x_0 devuelve el valor de la función $f(x)$ definida en la introdución en el punto x_0.
- *fn*: dados un punto x_0, la función $f(x)$ y un entero n devuelve el resultado de aplicar la función $f(x)$ en el punto x_0 n veces.
- *atractor*: dados el parámetro r y un punto x_0 pinta la órbita y devuelve la órbita, el cardinal de esta y el error obtenido en el cálculo. A continuación entraremos más en profundidad en esta función para explicarla detalladamente.
- *fError* y *afinar*: dados los resultados obtenidos por la función atractor devuelve el error obtenido en el cálculo. *fError* es una función auxiliar de *afinar*.

La función *atractor* primero calcula los n primeros términos del sistema dinámico discreto $x_{n+1} = f(x_n)$. El número n debe ser un valor lo suficientemente grande para que la órbita se estabilice. A continuación pintamos una gráfica con el valor de n en el eje de abscisas y el valor de x_n en el eje de ordenadas, con el fin de hacernos una idea de si la órbita se estabiliza.

Para calcular cúal es el periodo de esta órbita, tomamos los últimos k valores de la órbita y los recorremos desde el final hasta encontrar dos elementos iguales (no son exactamente iguales, si no que distan menos de un ϵ muy pequeño). La distancia entre estos dos elementos es el periodo de la órbita.

Calculamos el error afinando la órbita con sucesivas iteraciones. Para ello calculamos los errores entre una iteración y la siguiente y una vez obtenido este vector de errores, lo ordenamos y devolvemos el percentil deseado, en este caso, el de orden 0,9.

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

Para el segundo apartado no hemos tenido que definir ninguna función, únicamente hemos modificado la función *atractor* para que devolviera el cardinal de la órbita (que coincide con el periodo).

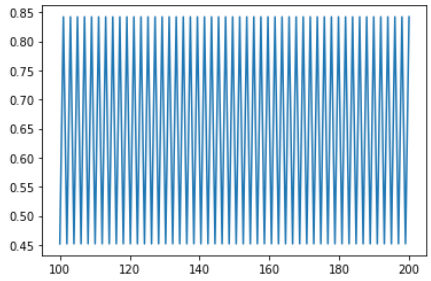
Tomamos un punto aleatorio $x_0 \in [0, 1]$ y hacemos una búsqueda binaria entre los extremos del intervalo $(3, 4)$ hasta encontrar un valor de r en el cúal el cardinal de la órbita sea 8. Podemos utilizar el algoritmo de búsqueda binaria porque la función que dado un valor de r devuelve el cardinal de su órbita es monótona creciente.

Una vez tenemos este punto realizamos dos búsquedas binarias en paralelo a la izquierda y a la derecha de este valor para encontrar los valores de r más a la izquierda y a la derecha, respectivamente que cumplen que el cardinal de su órbita es 8. Estas búsquedas las realizamos con 20 iteraciones para que la longitud de los intervalos sea del orden de 2^{-20} y así minimizar el error.

3. Resultados

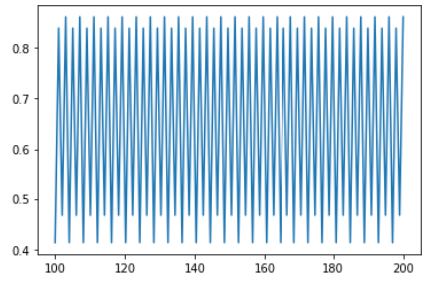
```
APARTADO UNO:

Primer atractor:
```




```
Periodo: 2
V0 está formado por  [0.45212596 0.84204755] , cuyos valores
se han calculado con un error de  2.609024107869118e-15

Segundo atractor:
```




```
Periodo: 4
V0 está formado por  [0.46748585 0.41323415 0.86134217
0.83895204] , cuyos valores se han calculado con un error de
1.3713913837865022e-07

APARTADO DOS:

Para r perteneciente al intervalo  [3.54635526239872,
3.562156096100807]  v0 tiene 8 elementos.
```

4. Conclusión

Tras varias ejecuciones hemos observado que para valores similares de r y de x_0 tanto el número de elementos de la órbita como los valores de estos no varian prácticamente. Un detalle que hemos visto es que a veces encontraba periodo 2 en casos en los que teóricamente el periodo es 4, esto sucede porque el ϵ tomado era mayor que la diferencia entre dos de los puntos de la órbita. Este detalle no es tan sencillo de solucionar porque si disminuíamos el ϵ entonces había casos en los que no encontraba periodo al ser demasiado exigente el margen de error.

En el segundo apartado no se obtienen resultados tan precisos como en el primero y esto se debe a que se acarrea varias veces el error del primero ya que se emplea la misma función pero ejecutada muchas más veces. Como se ha comentado en la segunda sección, si la función atractor fuera perfecta, el error sería del orden de 2^{-20}.

5. Código

```
"""
Created on Mon Feb  3 10:14:07 2020
@author: Jorge Sainero y Lucas de Torre
"""

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

PINTAR_GRAFICA = True
EPSILON = 1e-6

#Dado un punto x y el parámetro r devuelve el valor de la función en x
def logistica(x, r):
    return r*x*(1-x)

#Aplica n veces al función f en x0
def fn(x0, f, n, r):
    x=x0
    for j in range(n):
        x=f(x, r)
    return x

def fError(x0, f, n, r):
    x=x0
    aux=np.zeros(n)
    for j in range(n):
        x=f(x, r)
        aux[j]=x
    return x,aux


#Esta funcion afina V0 y calcula su error realizando las diferencias entre
    sucesivas iteraciones
def afinar(v0,f, r, iteraciones, cuantil):
    error=np.zeros(iteraciones)
    aux=np.zeros((v0.size,iteraciones))
    v0.sort()
    v0ini=v0
    for i in range(v0.size):
        v0[i],aux[i]=fError(v0[i],f,iteraciones, r)
```

```
    for i in range(iteraciones):
        aux[:,i].sort()

    for i in range(iteraciones-1):
        for j in range(v0.size):
            aux[j][iteraciones-1-i]=
                np.abs(aux[j][iteraciones-1-i]-aux[j][iteraciones-2-i])

    for j in range(v0.size):
            aux[j][0]=np.abs(aux[j][0]-v0ini[j])

    for i in range(iteraciones):
        error[i]=max(aux[:,i])
    error.sort()

    return v0,error[max(round(iteraciones*cuantil-1),0)]

#Dado un punto x0 y el parámetro r, calcula la órbita y devuelve su cardinal
def atractor(x, r):
    k = 25
    n = 200
    orb =  np.zeros(n)
    #Calculamos los n primeros términos de la sucesión x_n+1=f(x_n)
    for i in range(n):
        orb[i] = fn(x,logistica,i, r)
    if PINTAR_GRAFICA:
        abscisas = np.linspace(100,n,n-100)
        plt.plot(abscisas,orb[99:n-1])
        plt.show()
    #Tomamos los k últimos términos de la sucesión
    ult=orb[-1*np.arange(k,0,-1)]
    periodo = -1
    #Calculamos el periodo de la órbita para saber cuantos elementos tiene y
        cuales son
    for i in range(1,k,1):
        if abs(ult[k - 1] - ult[k - i - 1]) < EPSILON:
            periodo = i
            break
    if periodo != -1:
        #Si lo encontramos, tomamos esos elementos en v0
        error = 0
        v0 = orb[-1*np.arange(periodo,0,-1)]
        #Afinamos V0 calculando su error con cuantil de orden 0,9
```

```
        v0,error = afinar(v0,logistica, r, 10,0.9)
        return periodo,v0,error
    return periodo,0,0


def apartado1():
    print("APARTADO UNO:\n")
    global PINTAR_GRAFICA
    PINTAR_GRAFICA = True
    r1 = rand.uniform(3.0001 ,3.4999)
    r2 = rand.uniform(3.0001 ,3.4999)
    x01 = rand.random()
    x02 = rand.random()

    print("Primer atractor:")
    per1,v01,error1=atractor(x01, r1)
    if per1 !=-1:
        print("Periodo: "+str(per1))
        print("V0 está formado por ",v01,", cuyos valores se han calculado
            con un error de ",error1)
    else:
        print ("No se ha encontrado un periodo")
    print("\n\n")

    print("Segundo atractor:")
    per2,v02,error2=atractor(x02, r2)
    if per2 != -1:
        print("Periodo: "+str(per2))
        print("V0 está formado por ",v02,", cuyos valores se han calculado
            con un error de ",error2)
    else:
        print ("No se ha encontrado un periodo")
    print("\n\n\n")

def apartado2():
    print("APARTADO DOS:\n")
    global PINTAR_GRAFICA
    PINTAR_GRAFICA = False
    x0 = rand.random()
    a = 3
    b = 4
    cardinalV0 = -1
```

```
    #Iteramos hasta encontar un r tal que V0 tenga 8 elementos
    while cardinalV0 != 8:
        r = (a+b)/2
        cardinalV0,aux,aux = atractor(x0, r)
        if cardinalV0 > 8 or cardinalV0 == -1:
            b = r
        elif cardinalV0 < 8:
            a = r
    #Buscamos los extremos del intervalo de los valores de r tal que el
        cardinal de la órbita es 8
    a1 = a #Extremo izquierdo del intervalo izquierdo
    b1 = r #Extremo derecho del intervalo izquierdo
    a2 = r #Extremo izquierdo del intervalo derecho
    b2 = b #Extremo derecho del intervalo derecho
    #Iteramos 20 veces para reducir el error
    for i in range(20):
        r1 = (a1 + b1)/2
        cardinalV01,aux,aux = atractor(x0,r1)
        r2 = (a2 + b2)/2
        cardinalV02,aux,aux = atractor(x0,r2)
        if cardinalV01 < 8:
            a1 = r1
        else:
            b1 = r1
        if cardinalV02 > 8 or cardinalV02 == -1:
            b2 = r2
        else:
            a2 = r2
    #Devolvemos b1 y a2 para asegurarnos que en esos puntos el periodo vale 8
    print ("Para r perteneciente al intervalo ",[b1,a2]," v0 tiene 8 elementos.")

apartado1()
apartado2()
```