1. Introducción

Se ha desarrollado un código en python para estimar parcialmente el atractor de la función logística. He hecho una implementación de los algoritmos lo más modular posible, para evitar la repetición de código.

En la primera parte de la entrega he estimado el valor de dos conjuntos atractores para dos valores del parámetro r.

En la segunda parte he estimado los valores del parámetro $r \in (3{,}544, 4)$ tales que las órbitas correspondientes tienen periodo 8.

2. Método

Para realizar las estimaciones de los atractores he procedido de la siguiente manera:

1. He fijado el parámetro r y un valor inicial x_0 y he calculado los primeros N valores de la órbita (ver función `orbita`):
$$\forall n \in \{1, \dots, N-1\} : \; x_n := r x_{n-1}(1 - x_{n-1}).$$
2. Con estos valores y fijada una tolerancia ε he estimado el periodo de la órbita, teniendo en cuenta posibles periodos $p \in \{1, \dots, \texttt{N\_ULT}/2\}$ y restando los últimos valores de la órbita (ver función `periodo`). Es decir, he determinado que la órbita tiene periodo p si
$$\forall j = \{1, \dots, p\} : \; |x_{N-1-j} - x_{N-1-j-p}| < \varepsilon.$$
3. Habiendo estimado el periodo de la órbita es fácil obtener la estimación del conjunto atractor: es suficiente con considerar los últimos p valores de la sección inicial de la órbita, donde p es el periodo (ver función `atractor`). Para estimar el error de estos puntos de la órbita se consideran los últimos $2 \cdot p$ valores y se resta cada uno con su correspondiente (ver funciones `estimar_errores_atractor` y `estimar_error_atractor`):
$$e_{x_{N-1-j}} := |x_{N-1-j} - x_{N-1-j-p}| \, .$$

Para estimar los valores de $r \in (3{,}544, 4)$ correspondientes a las órbitas de periodo 8 he procedido como sigue:

1. He considerado M puntos equidistantes en el intervalo $(3{,}544, 4)$, separados entre sí por la distancia $\delta = \frac{4-3{,}544}{M-1}$. Para esto he utilizado la función `linspace` de *numpy*.
2. Para cada uno de estos valores de r he calculado el periodo y si este periodo corresponde con el buscado (8) también conjunto atractor el cojunto atractor correspondiente (ver función `atractores_con_periodo`).
3. El error estimado para los valores de r obtenidos se corresponde con el valor δ.

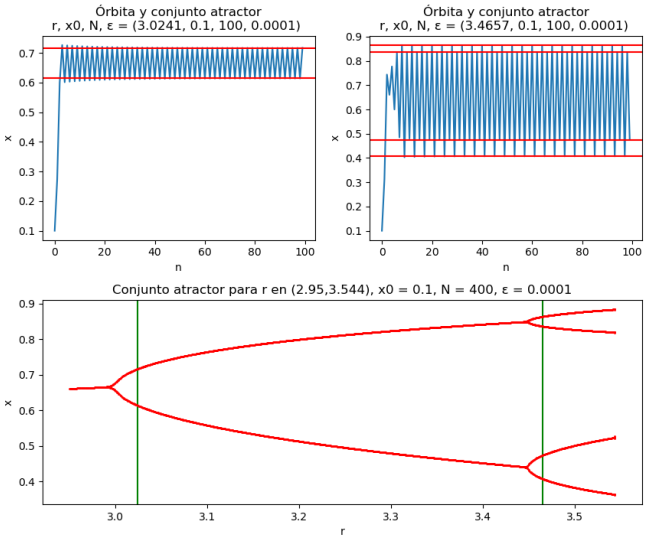
3. Resultados

En el primer apartado he estimado los conjuntos atractores para dos valores de r, con $x_0 = 0{,}1$, $\varepsilon = 10^{-4}$ y $N = 100$:

1. Para $r_1 = 3{,}0241$ he obtenido una estimación del periodo $p = 2$ y los siguientes valores del conjunto atractor:
   - $x_{N-1} = 0{,}613772835379854 \pm 5{,}655200368437363 \cdot 10^{-6}$
   - $x_{N-2} = 0{,}7168802691693898 \pm 8{,}218621085354094 \cdot 10^{-6}$
2. Para $r_2 = 3{,}4657$ he obtenido una estimación del periodo $p = 4$ y los siguientes valores del conjunto atractor:
   - $x_{N-1} = 0{,}4069223551243272 \pm 4{,}0606841750223666 \cdot 10^{-7}$

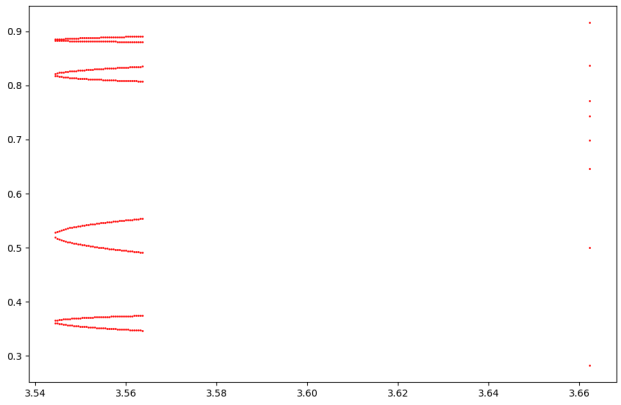
- $x_{N-2} = 0{,}4742289921180187 \pm 1{,}7414951214433927 \cdot 10^{-7}$
- $x_{N-3} = 0{,}8364000883509567 \pm 2{,}6993256668772503 \cdot 10^{-7}$
- $x_{N-4} = 0{,}8641233457379652 \pm 1{,}0695121777093419 \cdot 10^{-7}$

Estas son las gráficas correspondientes a las órbitas de estos valores de r. También he dibujado una gráfica con estimaciones del conjunto atractor para valores de r en el intervalo $(2{,}95, 3{,}544)$.



Para la segunda estimación, con los valores $M = 1000$, $x_0 = 0{,}5$ y $\varepsilon = 10^{-5}$ he obtenido 44 valores de r con periodo 8. El error de estimación es $\delta = \frac{4-3{,}544}{1000-1} = 0{,}000456456$. Estos son algunos ejemplos:

- $r = 3{,}5444564564564565 \pm \delta$ es el valor más pequeño.
- $r = 3{,}5636276276276275 \pm \delta$ es el valor más grande de la zona en la que se aprecia *continuidad*.
- $r = 3{,}6622222222222223 \pm \delta$ es un valor que está visiblemente aislado de los demás.



4. Código

Este es el código con el que he realizado las estimaciones y las gráficas. También está adjunto en la entrega y disponible en el repositorio git en el siguiente enlace: github.com/haztecaso/gcomp.

```
4 import matplotlib.pyplot as plt
5 import numpy as np
6
7 import logging
8 from typing import Callable, List
```

```
9
10  # Valores por defecto de los parámetros
11  DEFAULT_ε   = 1e-4
12
13  # Variables globales
14  MAX_ITERS  = int(1e4) # Máximo de iteraciones al calcular las órbitas
15  N_ULT      = 32 # Número de valores que considerar para calcular el periodo
16
17  class PeriodoNoEncontrado(Exception):
18      def __init__(self):
19          self.message = f"No se ha podido encontrar un periodo."
20          super().__init__(self.message)
21
22  def logistica(r:float) -> Callable[[float], float]:
23      """
24      Función logística parametrizada por el parámetro r.
25      Dado un parámetro r de tipo float devuelve la función logística
26      correspondiente de tipo float -> float.
27
28      :param float r: Parámetro r de la función logística
29      :return: Función logística de tipo float -> float
30      """
31      return lambda x: r*x*(1-x)
32
33  def orbita(r:float, x0:float, N:int):
34      """
35      :param float r: parámetro r de la función logística
36      :param float x0: valor inicial de la órbita
37      :param int N: número de iteraciones
38      :return: Órbita de longitud N (array de numpy)
39      """
40      assert N < MAX_ITERS, "Demasiadas iteraciones"
41      logging.info(f"Calculando órbita\t\t\t {(r, x0, N) = }")
42      f = logistica(r)
43      orb = np.empty((N,))
44      orb[0] = x0
45      logging.debug(f"{orb[0] = }")
46      for n in range(1, N):
47          orb[n] = f(orb[n-1])
48          logging.debug(f"{n = };{orb[n] = }")
49      return orb
50
51  def periodo(orb:np.ndarray, ε:float = DEFAULT_ε):
52      """
53      Calcula el periodo de una órbita.
54
55      :param orbita: Array con la órbita (generado con la función orbita)
56      :param float ε: Precisión
57      """
58      logging.info(f"Estimando el periodo de una órbita\t {ε = }")
59      assert N_ULT <= len(orb), f"No se pueden seleccionar {N_ULT} valores de una órbita de longitud {len(orb)}"
60      ultimos = orb[range(-N_ULT, 0, 1)]
61      for p in range(2, N_ULT-1, 1):
62          logging.debug(f"{p = }; {abs(ultimos[-1] - ultimos[N_ULT-p-1]) = }")
63          result = True
64          for j in range(0,p):
65              result = result and abs(ultimos[N_ULT-1-j] - ultimos[N_ULT-p-j]) < ε
66          if result:
67              return p-1
68      raise PeriodoNoEncontrado()
69
70  def atractor(orb:np.ndarray, ε:float = DEFAULT_ε, per:int=None):
71      """
72      Estima el conjunto atractor de una órbita concreta.
73
74      :param np.ndarray orbita: Array con la órbita (generado con la función orbita)
75      :param float ε: Precisión
76      """
77      logging.info(f"Estimando el conjunto atractor\t\t {ε = }")
78      if per is None: per = periodo(orb, ε)
79      logging.debug(f"{per = }")
80      result = np.sort([orb[-1-i] for i in range(per)])
81      logging.info(f"Conjunto atractor estimado: {result}")
82      return result
83
84  def estimar_errores_atractor(orb:np.ndarray, per:int):
85      """
86      Dada una órbita y un periodo devuelve los errores estimados de los puntos
87      del atractor correspondiente.
88
89      :param np.ndarray orb: Array con la órbita (generado con la función orbita)
90      :param int per: Periodo estimado de la órbita
91      :param float ε: Precisión
92      """
93      assert len(orb) >= 2*per, "Se necesitan al menos 2*{per}={2*per} valores para estimar el intervalo de error"
94      errs = []
95      for i in range(per):
96          errs.append(abs(orb[-1-i] - orb[-1-i-per]))
97      return errs
98
```

```
def estimar_error_atractor(orb:np.ndarray, per:int):
    """
    Dada una órbita y un periodo devuelve el error estimado (el mayor de todos)
    de los puntos del atractor correspondiente.
    """
    errs = estimar_errores_atractor(orb, per)
    return max(errs)

def orbita_atractor_plot(r:float, x0:float, N:int, ε:float = DEFAULT_ε, show:bool = True):
    """
    Gráfico de una órbita y el conjunto atractor correspondiente

    :param float r: parámetro r de la función logística
    :param float x0: valor inicial de la órbita
    :param int N: número de iteraciones
    :param float ε: Precisión
    """
    orb = orbita(r, x0, N)
    per = periodo(orb, ε)
    atr = atractor(orb, ε, per)
    plt.ylabel("x")
    plt.xlabel("n")
    plt.plot(orb)
    for valor in atr:
        plt.axhline(y = valor, color = 'r', linestyle = '-')
    plt.title(f"Órbita y conjunto atractor\n{r, x0, N, ε = } ")
    if show: plt.show()
    return (orb, per, atr)

def conjunto_atractor_plot(rs:np.ndarray, x0:float, N:int, ε:float =DEFAULT_ε, show:bool = True):
    """
    Dibujo de un conjunto atractor para múltiples r's

    :param np.ndarray rs: Valores de r
    :param float x0: valor inicial de las órbita
    :param int N: número de iteraciones
    :param float ε: Precisión
    :param bool show: Pintar la gráfica
    """
    for r in rs:
        try:
            orb = orbita(r, x0, N)
            atr = atractor(orb, ε)
            for v in atr:
                plt.plot(r, v, 'ro', markersize = 1)
        except PeriodoNoEncontrado:
            print(f"Periodo no encontrado para {r, N, ε = }")
    plt.title(f"Conjunto atractor para r en ({rs[0]},{rs[-1]}), {x0 = }, {N = }, {ε = }")
    plt.ylabel("x")
    plt.xlabel("r")
    if show: plt.show()

def atractores_con_periodo(p:int, rs:np.ndarray, x0:float, N:int, ε:float = DEFAULT_ε, **kwargs):
    """
    Dado un periodo fijo encuentra los valores de r, con sus atractores
    correspondientes, cuyas órbitas tienen ese periodo.
    También incluye la opción plot para dibujar los atractores obtenidos.

    :param int p: El periodo que se busca
    :param np.ndarray rs: Valores de r que testear
    :param float x0: valor inicial de las órbita
    :param int N: número de iteraciones
    :param float ε: Precisión
    :param bool plot: Plotear la gráfica. Valor por defecto True.
    :param str fmt: Formato de la gráfica. Valor por defecto 'ro'.
    :param bool show: Pintar la gráfica. Valor por defecto True.
    """
    logging.info(f"Buscando atractores con periodo {p} en el intervalo {rs[0],rs[-1]} ({N = })")
    result_rs = []
    result_atrs = []
    plot = kwargs.get('plot', True)
    show = kwargs.get('show', True)
    fmt  = kwargs.get('fmt', 'ro')
    for r in rs:
        logging.debug(f"{r = }")
        try:
            orb = orbita(r, x0, N)
            per = periodo(orb, ε)
            logging.debug(f"{per = }")
            if per == p:
                atr = atractor(orb, ε, per)
                result_rs.append(r)
                result_atrs.append(atr)
                for i in range(per):
                    if plot: plt.plot(r, atr[i], fmt, markersize=1)
        except PeriodoNoEncontrado:
            pass
    if plot and show: plt.show()
    return result_rs, result_atrs
```

```python
def apartado1():
    """
    Ejemplo de conjuntos atractores con sus correspondientes intervalos de error.
    """
    x0, N, ε =  0.1, 100, 1e-4

    r1 = 3.0241
    plt.subplot(2, 2, 1)
    orb1, per1, atr1 = orbita_atractor_plot(r1, x0, N, ε, show = False)
    errs1 = estimar_errores_atractor(orb1, per1)
    print(f"Estimación del atractor para r = {r1} con N = {N} y ε = {ε}:")
    print(f"- Periodo estimado: {per1}")
    print(f"- Puntos del atractor (no están escritos en notación estándar, ver memoria):")
    for i in range(len(atr1)):
        print(f"  - x_{N-1-i} = {atr1[i]} ±{errs1[i]}")
    print("")

    r2 = 3.4657
    plt.subplot(2, 2, 2)
    orb2, per2, atr2 = orbita_atractor_plot(r2, x0, N, ε, show = False)
    errs2 = estimar_errores_atractor(orb2, per2)
    print(f"Estimación del atractor para r = {r2} con N = {N} y ε = {ε}:")
    print(f"- Periodo estimado: {per2}")
    print(f"- Puntos del atractor (no están escritos en notación estándar, ver memoria):")
    for i in range(len(atr2)):
        print(f"  - x_{N-i} = {atr2[i]} ±{errs2[i]}")
    plt.subplot(2, 1, 2)
    plt.axvline(x = r1, color = 'g', linestyle = '-')
    plt.axvline(x = r2, color = 'g', linestyle = '-')

    # Gráfico de estimaciones del atractor en todo un intervalo
    conjunto_atractor_plot(np.linspace(2.95,3.544,1000), x0, 400, 1e-4)

def apartado2():
    """
    Estimación de valores de r en un intervalo para los que la órbita tiene periodo 8.
    """
    p, x0, N, ε = 8, 0.5, 1000, 1e-5
    M = 1000        # Número de r's que considerar
    a = 3.544       # Extremo inferior del intervalo de las r's
    b = 4.0         # Extremo superior del intervalo de las r's
    delta = (b-a)/(M-1) # Tamaño de los subintervalos en los que hemos dividido
    rs = np.linspace(a, b, M)
    rsp, _ = atractores_con_periodo(p, rs, x0, N, ε, plot = True, show = False)
    print(f"Se han obtenido {len(rsp)} valores de r con periodo {p}:")
    for r in rsp:
        print(f"- {r}±{delta}")
    plt.show()
```