

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 (función `orbita`):

$$\forall n \in \{1, \dots, N-1\} : x_n := rx_{n-1}(1 - x_{n-1}).$$

2. Con estos valores y fijada una tolerancia ε he estimado el periodo de la órbita, teniendo en cuenta posibles periodos $p \in \{1, \dots, N_ULT/2\}$ y restando los últimos valores de la órbita (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 (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 (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 conjunto 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:

$$\blacksquare x_{N-1} = 0.613772 \pm 5 \cdot 10^{-6}$$

$$\blacksquare x_{N-2} = 0.716880 \pm 8 \cdot 10^{-6}$$

2. Para $r_2 = 3.4657$ he obtenido una estimación del periodo $p = 4$ y los siguientes valores del conjunto atractor:

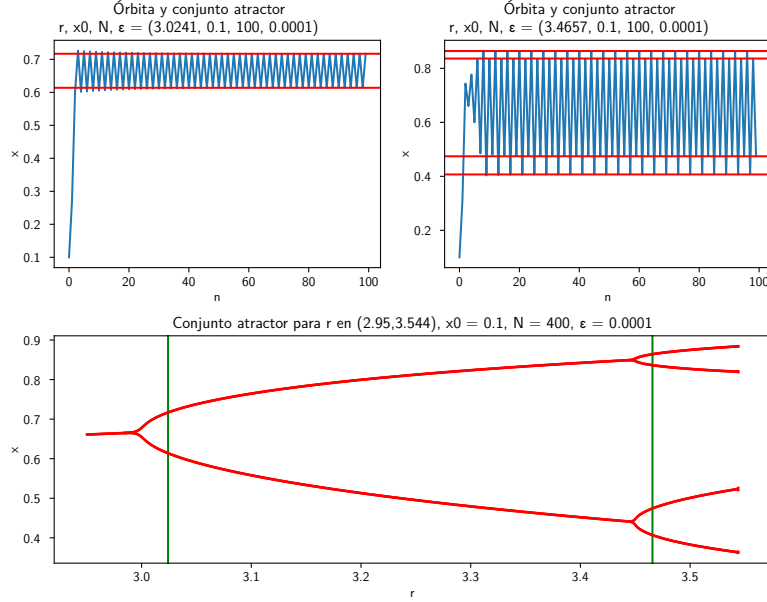
$$\blacksquare x_{N-1} = 0.4069223 \pm 4 \cdot 10^{-7}$$

$$\blacksquare x_{N-3} = 0.8364000 \pm 2 \cdot 10^{-7}$$

$$\blacksquare x_{N-2} = 0.4742289 \pm 1 \cdot 10^{-7}$$

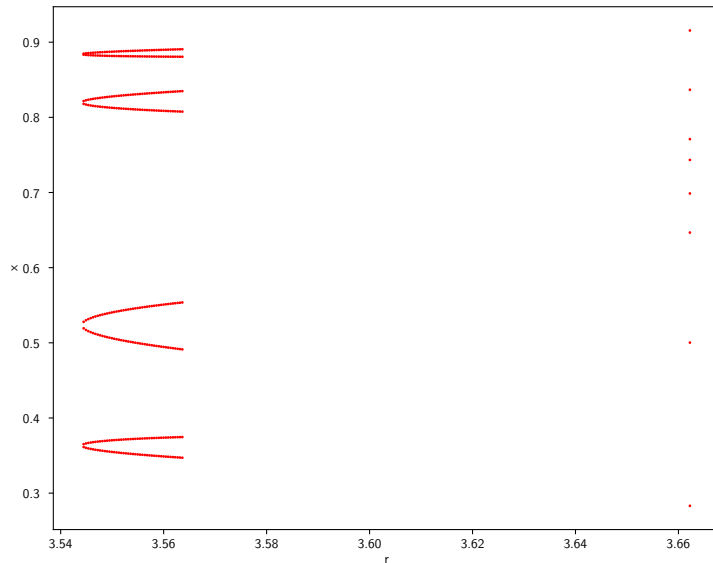
$$\blacksquare x_{N-4} = 0.8641233 \pm 1 \cdot 10^{-7}$$

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 $\epsilon = 10^{-5}$ he obtenido 44 valores de r con periodo 8. El error de estimación es $\delta = \frac{4-3.544}{1000-1} \approx 4 \cdot 10^{-4}$. Estos son algunos ejemplos:

- $r = 3.5444 \pm \delta$ es el valor más pequeño.
- $r = 3.5636 \pm \delta$ es el valor más grande de la zona en la que se aprecia *continuidad*.
- $r = 3.6622 \pm \delta$ 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, junto con esta memoria, en el repositorio git en el siguiente enlace: github.com/haztecaso/gcomp22.

```
4 import matplotlib.pyplot as plt
5 import numpy as np
6
7 import logging
8 from typing import Callable
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{(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
99 def estimar_error_atractor(orb:np.ndarray, per:int):
100 """
101 Dada una órbita y un periodo devuelve el error estimado (el mayor de todos)
102 de los puntos del atractor correspondiente.
103 """
104 errs = estimar_errores_atractor(orb, per)
105 return max(errs)
106
107 def orbita_atractor_plot(r:float, x0:float, N:int, ε:float = DEFAULT_ε, show:bool = True):
108 """
109 Gráfico de una órbita y el conjunto atractor correspondiente
110
111 :param float r: parámetro r de la función logística
112 :param float x0: valor inicial de la órbita
113 :param int N: número de iteraciones
114 :param float ε: Precisión
115 """
116 orb = orbita(r, x0, N)
117 per = periodo(orb, ε)
118 atr = atractor(orb, ε, per)
119 plt.ylabel("x")
120 plt.xlabel("n")
121 plt.plot(orb)
122 for valor in atr:
123     plt.axhline(y = valor, color = 'r', linestyle = '-')
124 plt.title(f"Órbita y conjunto atractor\n{r, x0, N, ε = } ")
125 if show: plt.show()
126 return (orb, per, atr)
127
128 def conjunto_atractor_plot(rs:np.ndarray, x0:float, N:int, ε:float =DEFAULT_ε, show:bool = True):
129 """
130 Dibujo de un conjunto atractor para múltiples r's
131
132 :param np.ndarray rs: Valores de r
133 :param float x0: valor inicial de las órbita
134 :param int N: número de iteraciones
135 :param float ε: Precisión
136 :param bool show: Pintar la gráfica
137 """
138 for r in rs:
139     try:
140         orb = orbita(r, x0, N)
141         atr = atractor(orb, ε)
142         for v in atr:
143             plt.plot(r, v, 'ro', markersize = 1)
144     except PeriodoNoEncontrado:
145         print(f"Periodo no encontrado para {r, N, ε = }")
146 plt.title(f"Conjunto atractor para r en ({rs[0]}, {rs[-1]}), {x0 = }, {N = }, {ε = }")
147 plt.ylabel("x")
148 plt.xlabel("r")
149 if show: plt.show()
150
151 def atractores_con_periodo(p:int, rs:np.ndarray, x0:float, N:int, ε:float = DEFAULT_ε, **kwargs):
152 """
153 Dado un periodo fijo encuentra los valores de r, con sus atractores
154 correspondientes, cuyas órbitas tienen ese periodo.
155 También incluye la opción plot para dibujar los atractores obtenidos.
156
157 :param int p: El periodo que se busca
158 :param np.ndarray rs: Valores de r que testear
159 :param float x0: valor inicial de las órbita
160 :param int N: número de iteraciones
161 :param float ε: Precisión
162 :param bool plot: Plotear la gráfica. Valor por defecto True.
163 :param str fmt: Formato de la gráfica. Valor por defecto 'ro'.
164 :param bool show: Pintar la gráfica. Valor por defecto True.
165 """
166 logging.info(f"Buscando atractores con periodo {p} en el intervalo {rs[0],rs[-1]} ({N = })")
167 result_rs = []
168 result_atrs = []
169 plot = kwargs.get('plot', True)
170 show = kwargs.get('show', True)
171 fmt = kwargs.get('fmt', 'ro')
172 if plot: plt.xlabel('r'); plt.ylabel('x')
173 for r in rs:
174     logging.debug(f"{r = }")

```

```

175     try:
176         orb = orbita(r, x0, N)
177         per = periodo(orb, ε)
178         logging.debug(f"{per = }")
179         if per == p:
180             atr = atractor(orb, ε, per)
181             result_rs.append(r)
182             result_atrs.append(atr)
183             for i in range(per):
184                 if plot: plt.plot(r, atr[i], fmt, markersize=1)
185         except PeriodoNoEncontrado:
186             pass
187     if plot and show: plt.show()
188     return result_rs, result_atrs
189
190 def apartado1():
191     """
192     Ejemplo de conjuntos atractores con sus correspondientes intervalos de error.
193     """
194     x0, N, ε = 0.1, 100, 1e-4
195
196     r1 = 3.0241
197     plt.subplot(2, 2, 1)
198     orb1, per1, atr1 = orbita_atractor_plot(r1, x0, N, ε, show = False)
199     errs1 = estimar_errores_atractor(orb1, per1)
200     print(f"Estimación del atractor para r = {r1} con N = {N} y ε = {ε}:")
201     print(f"- Periodo estimado: {per1}")
202     print(f"- Puntos del atractor (no están escritos en notación estándar, ver memoria):")
203     for i in range(len(atr1)):
204         print(f"    - x_{N-1-i} = {atr1[i]} ±{errs1[i]}")
205     print("")
206
207     r2 = 3.4657
208     plt.subplot(2, 2, 2)
209     orb2, per2, atr2 = orbita_atractor_plot(r2, x0, N, ε, show = False)
210     errs2 = estimar_errores_atractor(orb2, per2)
211     print(f"Estimación del atractor para r = {r2} con N = {N} y ε = {ε}:")
212     print(f"- Periodo estimado: {per2}")
213     print(f"- Puntos del atractor (no están escritos en notación estándar, ver memoria):")
214     for i in range(len(atr2)):
215         print(f"    - x_{N-i} = {atr2[i]} ±{errs2[i]}")
216     plt.subplot(2, 1, 2)
217     plt.axvline(x = r1, color = 'g', linestyle = '-')
218     plt.axvline(x = r2, color = 'g', linestyle = '-')
219
220     # Gráfico de estimaciones del atractor en todo un intervalo
221     conjunto_atractor_plot(np.linspace(2.95, 3.544, 1000), x0, 400, 1e-4)
222
223 def apartado2():
224     """
225     Estimación de valores de r en un intervalo para los que la órbita tiene periodo 8.
226     """
227     p, x0, N, ε = 8, 0.5, 1000, 1e-5
228     M = 1000 # Número de r's que considerar
229     a = 3.544 # Extremo inferior del intervalo de las r's
230     b = 4.0 # Extremo superior del intervalo de las r's
231     delta = (b-a)/(M-1) # Tamaño de los subintervalos en los que hemos dividido
232     rs = np.linspace(a, b, M)
233     rsp, _ = atractores_con_periodo(p, rs, x0, N, ε, plot = True, show = False)
234     print(f"Se han obtenido {len(rsp)} valores de r con periodo {p}:")
235     for r in rsp:
236         print(f"- {r} ±{delta}")
237     plt.show()
238
239 def main():
240     apartado1()
241     apartado2()
242
243 if __name__ == "__main__":
244     # Configuración del logger. Se puede cambiar el nivel del logger para debuguear o no imprimir ningún mensaje.
245     logging.basicConfig(level=logging.WARN, format='%(message)s')
246     logging.getLogger('matplotlib').setLevel(logging.WARN)
247     main()

```