Caos: Teoría y Experimentos

Alberto

Table of contents

Chapter 1

Caos: Teoría y Experimentos

Chapter 2

Bienvenido

Este libro explora el fascinante mundo del caos desde distintos puntos de vista:

- **Modelos matemáticos simples**, como la función logística, que nos introducen a las bifurcaciones, atractores y comportamiento impredecible en sistemas deterministas.
- **Sistemas físicos reales**, como el péndulo doble, que muestran cómo el caos se manifiesta en el mundo tangible.
- **Meteorología y clima**, donde la sensibilidad a condiciones iniciales limita la predictibilidad de los modelos atmosféricos y climáticos.
- **Experimentos caseros**, que puedes realizar tú mismo para observar el caos en acción.

Cada capítulo está dividido en secciones independientes, con gráficos, simulaciones, ecuaciones en LaTeX y referencias a la literatura científica cuando sea relevante.

> *"El aleteo de una mariposa en Brasil puede provocar un tornado en Texas."*
> — Edward Lorenz

Comencemos el viaje hacia la comprensión del caos.

Part I

1. Introducción al Caos a través de la Función Logística

Chapter 3

El Mapa Logístico

Chapter 4

El Mapa Logístico

4.1 Introducción

El mapa logístico es una de las ecuaciones en diferencia más clásicas de la teoría del caos:

$$x_{n+1} = r\, x_n\, (1 - x_n)$$

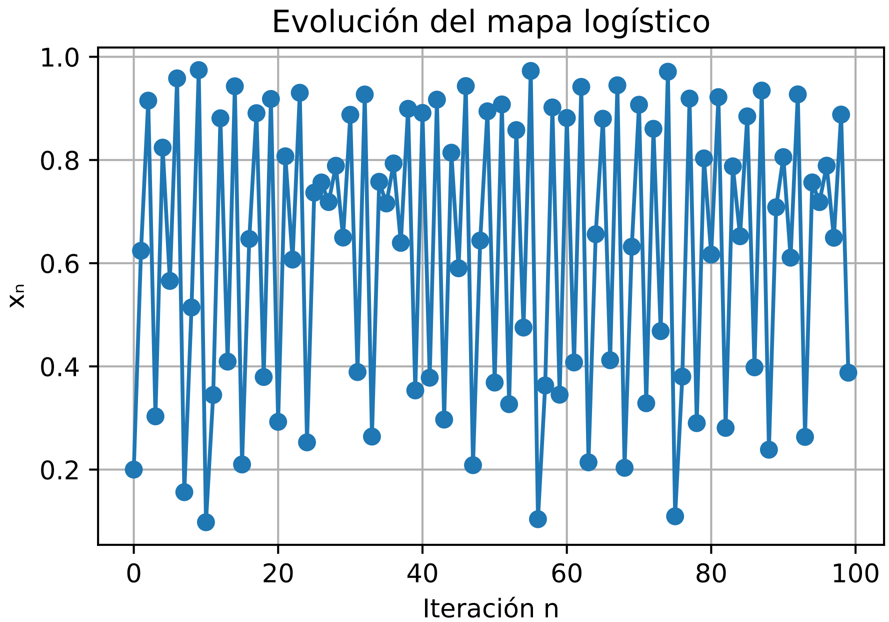
donde:

- $x_n \in [0, 1]$ es la población normalizada.
- r regula la tasa de crecimiento.

4.2 Simulación en Python

```
import numpy as np
import matplotlib.pyplot as plt

r, x0, n = 3.9, 0.2, 100
x = np.zeros(n)
x[0] = x0
for i in range(1, n):
    x[i] = r * x[i-1] * (1 - x[i-1])

plt.plot(x, marker='o')
plt.title("Evolución del mapa logístico")
plt.xlabel("Iteración n")
```

```
plt.ylabel("x ")
plt.grid(True)
plt.show()
```



Chapter 5

Diagrama Cobweb Interactivo

Chapter 6

Diagrama Cobweb Interactivo

6.1 Diagrama Cobweb Interactivo con Serie de (x__n)

A la izquierda se muestra el diagrama cobweb y a la derecha la evolución temporal de (x__n). Ajusta el parámetro r con el deslizador.

```
import numpy as np
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Función que calcula coordenadas de cobweb y serie de x_n
def compute_cobweb_and_series(r, x0=0.2, steps=40):
    # Línea logística y la identidad
    xs = np.linspace(0, 1, 200)
    ys = r * xs * (1 - xs)

    # Serie de iteraciones de la función logística
    series = [x0]
    x = x0
    for _ in range(steps):
        x = r * x * (1 - x)
        series.append(x)

    # Coordenadas para el diagrama cobweb
    xc, yc = [series[0]], [series[0]]
    for val in series[1:]:
```

```
        # vertical
        xc.append(xc[-1]); yc.append(val)
        # horizontal
        xc.append(val); yc.append(val)

    return xs, ys, xc, yc, series

# Valores de r para el slider
e_rs = np.linspace(2.5, 4.0, 31)
frames = []
for r in e_rs:
    xs, ys, xc, yc, series = compute_cobweb_and_series(r)
    frames.append(
        go.Frame(
            name=f"{r:.2f}",
            data=[
                go.Scatter(x=xs, y=ys, mode='lines'),
                go.Scatter(x=xs, y=xs, mode='lines', line=dict(dash='dash')),
                go.Scatter(x=xc, y=yc, mode='lines', line=dict(color='red')),
                go.Scatter(x=list(range(len(series))), y=series, mode='lines+markers')
            ]
        )
    )

# Crear figura con subplots 1x2
grid = make_subplots(rows=1, cols=2, subplot_titles=("Cobweb","Evolución de x_n"))
# Trazas iniciales (r = e_rs[0])
r0 = e_rs[0]
xs0, ys0, xc0, yc0, series0 = compute_cobweb_and_series(r0)
grid.add_trace(go.Scatter(x=xs0, y=ys0, mode='lines', name='f(x)'), row=1, col=1)
grid.add_trace(go.Scatter(x=xs0, y=xs0, mode='lines', name='y=x', line=dict(dash='dash
grid.add_trace(go.Scatter(x=xc0, y=yc0, mode='lines', name='Cobweb', line=dict(color='
grid.add_trace(go.Scatter(x=list(range(len(series0))), y=series0, mode='lines+markers'
# Asignar frames y configurar animación

grid.frames = frames
steps = [dict(label=f"{r:.2f}", method="animate",
              args=[[f"{r:.2f}"], dict(mode="immediate", frame=dict(duration=0, redraw=
         for r in e_rs]

grid.update_layout(
    width=1000, height=500,
    sliders=[dict(active=0, pad=dict(t=50), steps=steps)],
    updatemenus=[dict(type="buttons", showactive=False,
                      buttons=[dict(label="Play", method="animate",
```

```
                                    args=[None, dict(frame=dict(duration=100, redraw=True), fromc
)

grid.show()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

Chapter 7

El Mapa Logístico: Una Ventana al Caos

Chapter 8

El Mapa Logístico: Una Ventana al Caos

Duración de la explicación original: 0:00 – 17:17

8.1 1. ¿Qué es el mapa logístico?

El **mapa logístico** es un modelo discreto que, con una única ecuación, describe la evolución de una población normalizada x_n paso a paso y muestra comportamientos que van desde la convergencia hasta el caos.

Se define mediante la iteración:

$$x_{n+1} = r\, x_n\, (1 - x_n)$$

- $x_n \in [0, 1]$**:** población normalizada en la iteración n.
- r**:** parámetro de control o tasa de crecimiento (capacidad reproductiva).

  Nota: Para garantizar que $x_{n+1} \in [0, 1]$, se restringe r a $0 < r \leq 4$.

8.2 2. Breve historia

1. **Ecuación continua:** proviene de la ecuación logística diferencial, usada en biología para modelar el crecimiento con límite de recursos.
2. **Discretización:** Robert May (1976) introdujo su forma iterada y mostró que, pese a la regla sencilla, su dinámica es muy rica.

8.3 3. Forma y dominio de la función

La función asociada es:

$$f(x) = r\,x\,(1 - x)$$

1. **Parábola invertida:** abre hacia abajo.
2. **Punto máximo:**
   - Se alcanza en $x = 1/2$.
   - Valor máximo: $f(1/2) = r/4$.
3. **Dominio y recorrido:**
   - Dominio: $x \in [0, 1]$.
   - Para $0 < r \leq 4$, el recorrido queda en $[0, 1]$.

8.4 4. Puntos fijos y estabilidad

8.5 Cálculo de la derivada

Partimos de la función del mapa logístico:

$$f(x) = r\,x\,(1 - x).$$

Para obtener su derivada:

1. Aplicamos la regla del producto:
$$f'(x) = r\frac{d}{dx}\big[x(1 - x)\big] = r\big[(1 - x) + x(-1)\big].$$
2. Simplificamos:
$$f'(x) = r\,(1 - x - x) = r\,(1 - 2x).$$

Por tanto,

$$\boxed{f'(x) = r\,(1 - 2x).}$$

8.6 Criterio de estabilidad

Sea x^* un punto fijo (es decir, $f(x^*) = x^*$). Consideremos una pequeña perturbación δ tal que:

$$x_n = x^* + \delta.$$

Al aplicar el mapa:

$$x_{n+1} = f(x_n) = f(x^* + \delta) \approx f(x^*) + f'(x^*)\,\delta = x^* + f'(x^*)\,\delta.$$

La nueva desviación respecto a x^* es

$$x_{n+1} - x^* \approx f'(x^*)\,\delta.$$

- Si $|f'(x^*)| < 1$, entonces $|x_{n+1} - x^*| < |\delta|$. Cada iteración reduce la desviación: el punto fijo **atrae** las trayectorias (es **estable**).
- Si $|f'(x^*)| > 1$, la desviación crece y las trayectorias se **alejan** del punto fijo (es **inestable**).

En resumen:

$$|f'(x^*)| < 1 \quad \Longrightarrow \quad x^* \text{ es estable.}$$

Un **punto fijo** x^* satisface:

$$f(x^*) = x^*.$$

Para el mapa logístico:

- $x_1^* = 0$.
- $x_2^* = 1 - 1/r$ (si $r > 1$).

La derivada es:

$$f'(x) = r(1 - 2x).$$

Un punto fijo es estable si $|f'(x^*)| < 1$. Una perturbación δ en $x_n = x^* + \delta$ evoluciona como:

$$x_{n+1} - x^* \approx f'(x^*)\,\delta,$$

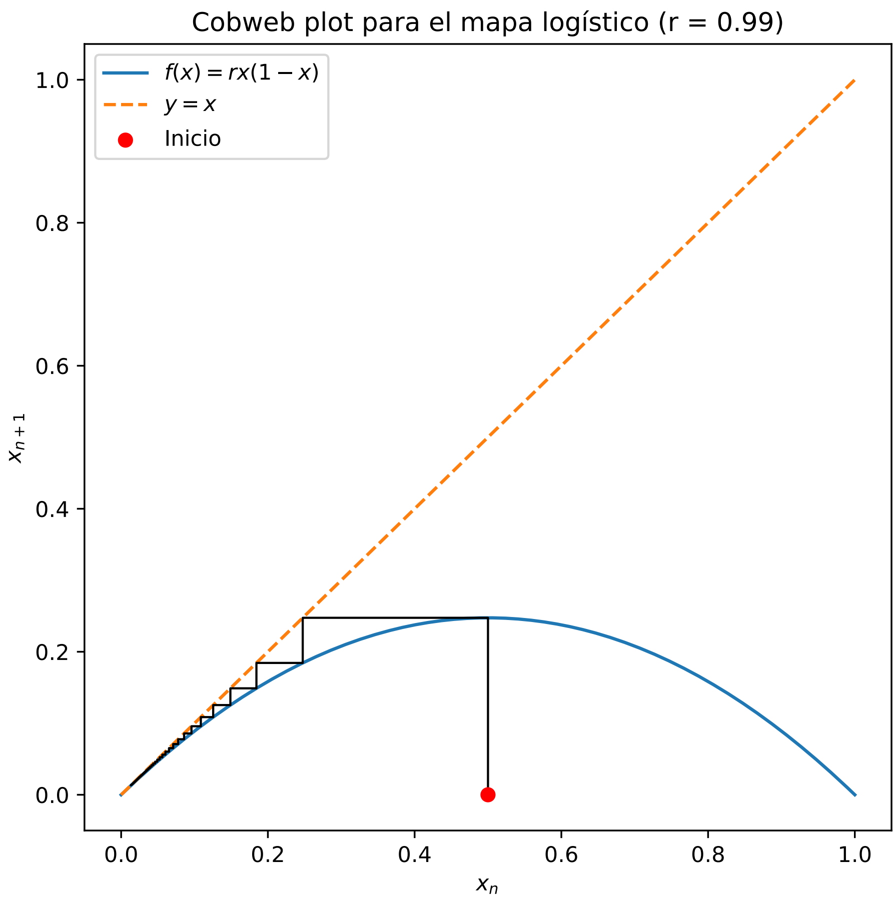
por lo que si $|f'(x^*)| < 1$, la desviación disminuye.

- **En $x_1^* = 0$:** $f'(0) = r$. Estable si $0 < r < 1$.
- **En $x_2^* = 1 - 1/r$:** $f'(x_2^*) = 2 - r$. Estable si $|2 - r| < 1 \to 1 < r < 3$.

Resumen de convergencia:

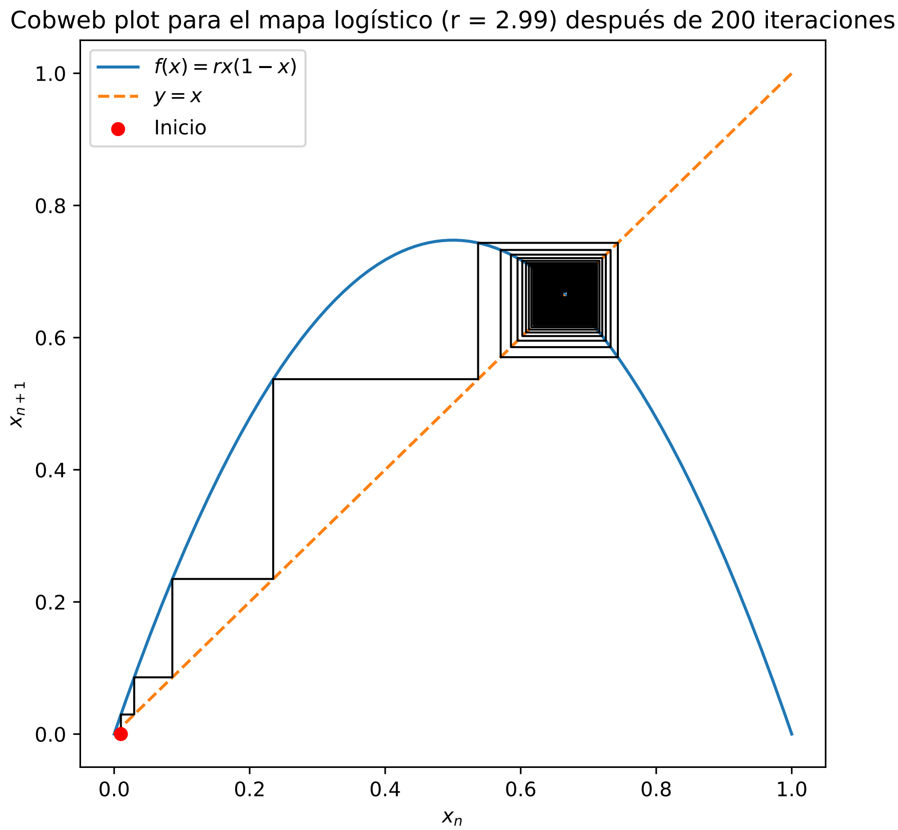
- $0 < r < 1$: converge a $x = 0$.
- $1 < r < 3$: converge a $x = 1 - 1/r$.

```
import numpy as np
import matplotlib.pyplot as plt

# Parámetros
r = 0.99
x0 = 0.5
n_iter = 50

# Función logística
def f(x): return r * x * (1 - x)

# Valores de x para la curva
x_vals = np.linspace(0, 1, 400)
y_vals = f(x_vals)

# Iteraciones para el cobweb
x_cobweb = [x0]
y_cobweb = [0]
x, y = x0, 0
for _ in range(n_iter):
    y = f(x)
    x_cobweb.append(x)
    y_cobweb.append(y)
    x = y
    x_cobweb.append(x)
    y_cobweb.append(x)

# Graficar
plt.figure(figsize=(6,6))
plt.plot(x_vals, y_vals, label='$f(x)=r x(1-x)$')
plt.plot(x_vals, x_vals, '--', label='$y=x$')
plt.plot(x_cobweb, y_cobweb, color='black', linewidth=1)
plt.scatter(x0, 0, color='red', zorder=5, label='Inicio')
plt.title('Cobweb plot para el mapa logístico (r = 0.99)')
plt.xlabel('$x_n$')
plt.ylabel('$x_{n+1}$')
plt.legend()
plt.tight_layout()
plt.show()
```



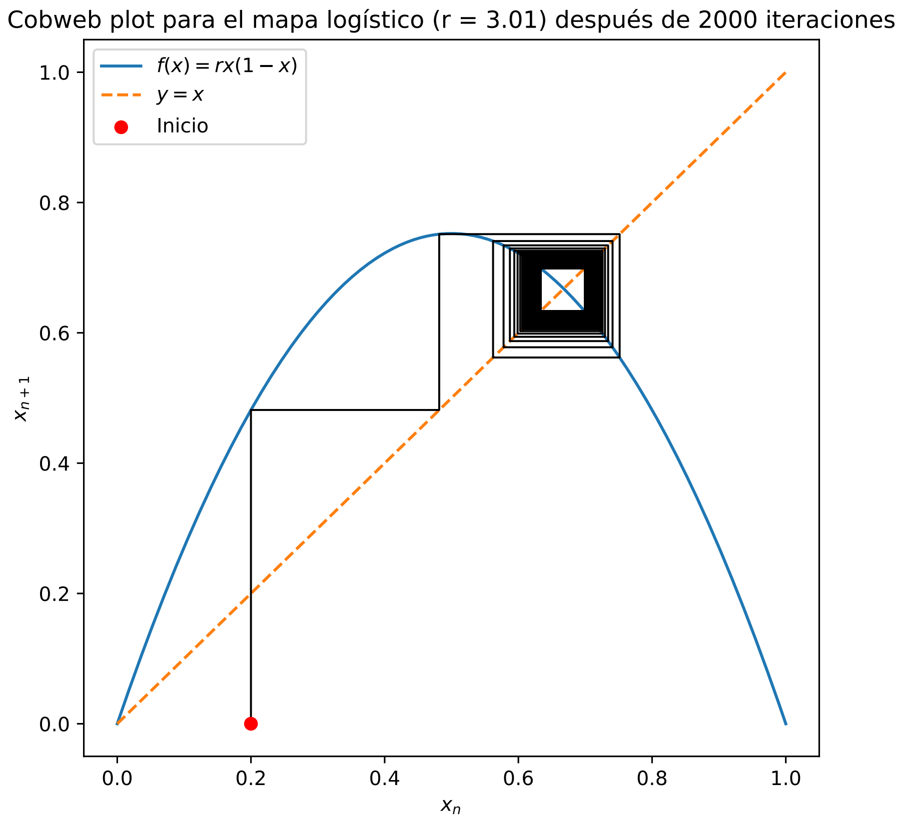

```
import numpy as np
import matplotlib.pyplot as plt

# Parámetros
r = 2.99
x0 = 0.01
n_iter = 200

# Función logística
def f(x): return r * x * (1 - x)

# Valores de x para la curva
x_vals = np.linspace(0, 1, 400)
y_vals = f(x_vals)

# Iteraciones para el cobweb
```

```
x_cobweb = [x0]
y_cobweb = [0]
x, y = x0, 0
for _ in range(n_iter):
    y = f(x)
    x_cobweb.append(x)
    y_cobweb.append(y)
    x = y
    x_cobweb.append(x)
    y_cobweb.append(x)

# Graficar
plt.figure(figsize=(6,6))
plt.plot(x_vals, y_vals, label='$f(x)=r x(1-x)$')
plt.plot(x_vals, x_vals, '--', label='$y=x$')
plt.plot(x_cobweb, y_cobweb, color='black', linewidth=1)
plt.scatter(x0, 0, color='red', zorder=5, label='Inicio')
plt.title('Cobweb plot para el mapa logístico (r = 2.99) después de 200 iteraciones')
plt.xlabel('$x_n$')
plt.ylabel('$x_{n+1}$')
plt.legend()
plt.tight_layout()
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

# Parámetros
r = 3.01
x0 = 0.2
n_iter = 2000

# Función logística
def f(x): return r * x * (1 - x)

# Valores de x para la curva
x_vals = np.linspace(0, 1, 400)
y_vals = f(x_vals)

# Iteraciones para el cobweb
x_cobweb = [x0]
y_cobweb = [0]
```

```
x, y = x0, 0
for _ in range(n_iter):
    y = f(x)
    x_cobweb.append(x)
    y_cobweb.append(y)
    x = y
    x_cobweb.append(x)
    y_cobweb.append(x)

# Graficar
plt.figure(figsize=(6,6))
plt.plot(x_vals, y_vals, label='$f(x)=r x(1-x)$')
plt.plot(x_vals, x_vals, '--', label='$y=x$')
plt.plot(x_cobweb, y_cobweb, color='black', linewidth=1)
plt.scatter(x0, 0, color='red', zorder=5, label='Inicio')
plt.title('Cobweb plot para el mapa logístico (r = 3.01) después de 2000 iteraciones')
plt.xlabel('$x_n$')
plt.ylabel('$x_{n+1}$')
plt.legend()
plt.tight_layout()
plt.show()
```



8.7 4. Primera Bifurcación: Duplicación de Período en $r = 3$

En $r = 3$, la derivada en el punto fijo $x^* = 1 - \frac{1}{r}$ se vuelve -1, lo cual genera una órbita de período 2.

Surgen dos nuevos puntos p y q que no son puntos fijos, sino puntos de período 2 tales que:

$$f(p) = q, \quad f(q) = p$$

Esto significa que:

$$f(f(p)) = p$$

Lo cual implica que p es un punto fijo del mapa iterado f^2.

Dado que $f(x) = rx(1-x)$, podemos escribir:

$$f(p) = rp(1-p)$$

Entonces:

$$f(f(p)) = r \cdot f(p) \cdot (1 - f(p)) = r \cdot [rp(1-p)] \cdot [1 - rp(1-p)]$$

Queremos encontrar los puntos de período 2, así que igualamos:

$$f(f(p)) = p$$

Desarrollando completamente:

$$r^2 p(1-p)(1 - rp(1-p)) = p$$

Pasando todo al mismo lado:

$$r^2 p(1-p)(1 - rp(1-p)) - p = 0$$

Factorizamos p:

$$p\left[r^2(1-p)(1 - rp(1-p)) - 1\right] = 0$$

Una de las soluciones es $p = 0$ (punto fijo trivial), pero las otras soluciones corresponden a los puntos de período 2.

Expandimos el polinomio:

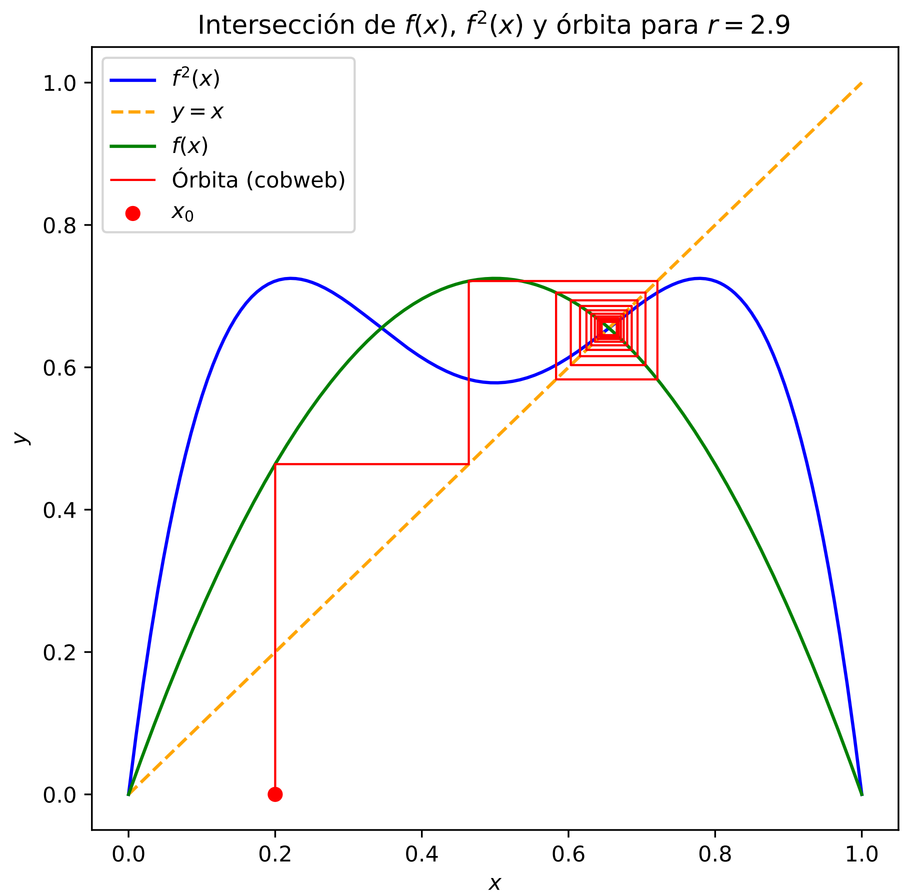
$$f(f(p)) = r^2 p(1-p)(1 - rp(1-p)) = p$$

Expandimos paso a paso:

1. $f(p) = rp(1-p)$
2. $1 - f(p) = 1 - rp(1-p)$
3. $(1-p)(1 - rp(1-p)) = 1 - p - rp(1-p) + rp^2(1-p)$
4. Multiplicamos todo por $r^2 p$
5. Resulta en un polinomio de cuarto grado en p

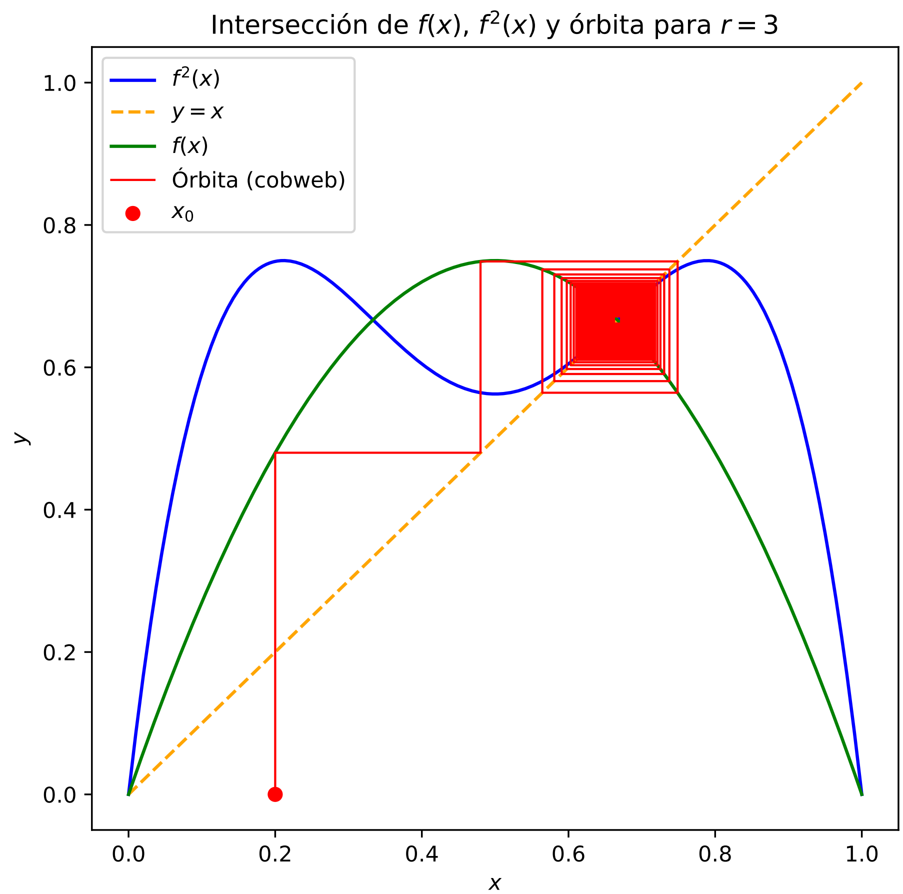
Este polinomio tiene hasta 4 raíces reales, de las cuales dos corresponden a los nuevos puntos de período 2. Las otras dos pueden ser los puntos fijos ya conocidos o raíces no relevantes dinámicamente.

```
import numpy as np
import matplotlib.pyplot as plt

# Parámetro y condición inicial
r = 2.9
x0 = 0.2
n_iter = 20

# Definición de funciones
def f(x): return r * x * (1 - x)
def f2(x): return f(f(x))

# Dominio
x_vals = np.linspace(0, 1, 400)
y_f = f(x_vals)
y_f2 = f2(x_vals)

# Construcción del cobweb sobre f
x_coords = [x0]
y_coords = [0]
x, y = x0, 0
for _ in range(n_iter):
    y = f(x)
    x_coords.extend([x, y])
    y_coords.extend([y, y])
    x = y

# Plot
plt.figure(figsize=(6, 6))
plt.plot(x_vals, y_f2, color='blue', label='$f^2(x)$')
plt.plot(x_vals, x_vals, color='orange', linestyle='--', label='$y = x$')
plt.plot(x_vals, y_f, color='green', label='$f(x)$')
plt.plot(x_coords, y_coords, color='red', linewidth=1, label='Órbita (cobweb)')
plt.scatter([x0], [0], marker='o', color='red', label='$x_0$')
plt.title('Intersección de $f(x)$, $f^2(x)$ y órbita para $r=2.9$')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.legend()
plt.tight_layout()
plt.show()
```



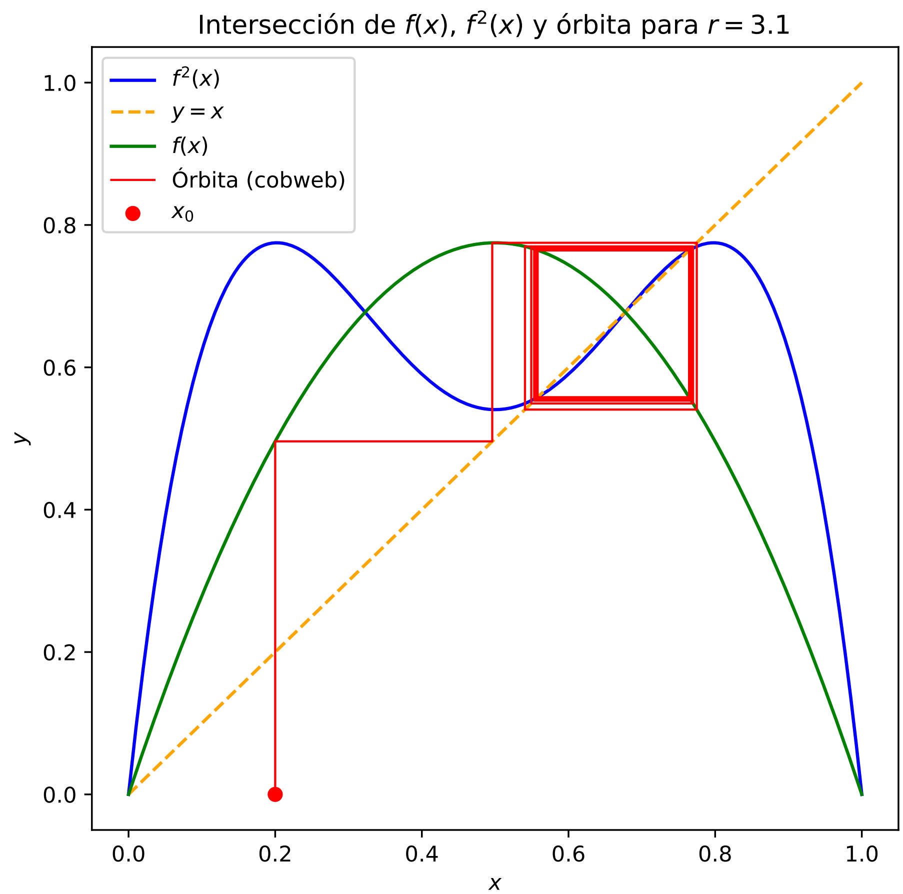

```
import numpy as np
import matplotlib.pyplot as plt

# Parámetro y condición inicial
r = 3
x0 = 0.2
n_iter = 2000

# Definición de funciones
def f(x): return r * x * (1 - x)
def f2(x): return f(f(x))

# Dominio
x_vals = np.linspace(0, 1, 400)
y_f = f(x_vals)
y_f2 = f2(x_vals)
```

```
# Construcción del cobweb sobre f
x_coords = [x0]
y_coords = [0]
x, y = x0, 0
for _ in range(n_iter):
    y = f(x)
    x_coords.extend([x, y])
    y_coords.extend([y, y])
    x = y

# Plot
plt.figure(figsize=(6, 6))
plt.plot(x_vals, y_f2, color='blue', label='$f^2(x)$')
plt.plot(x_vals, x_vals, color='orange', linestyle='--', label='$y = x$')
plt.plot(x_vals, y_f, color='green', label='$f(x)$')
plt.plot(x_coords, y_coords, color='red', linewidth=1, label='Órbita (cobweb)')
plt.scatter([x0], [0], marker='o', color='red', label='$x_0$')
plt.title('Intersección de $f(x)$, $f^2(x)$ y órbita para $r=3$')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.legend()
plt.tight_layout()
plt.show()
```



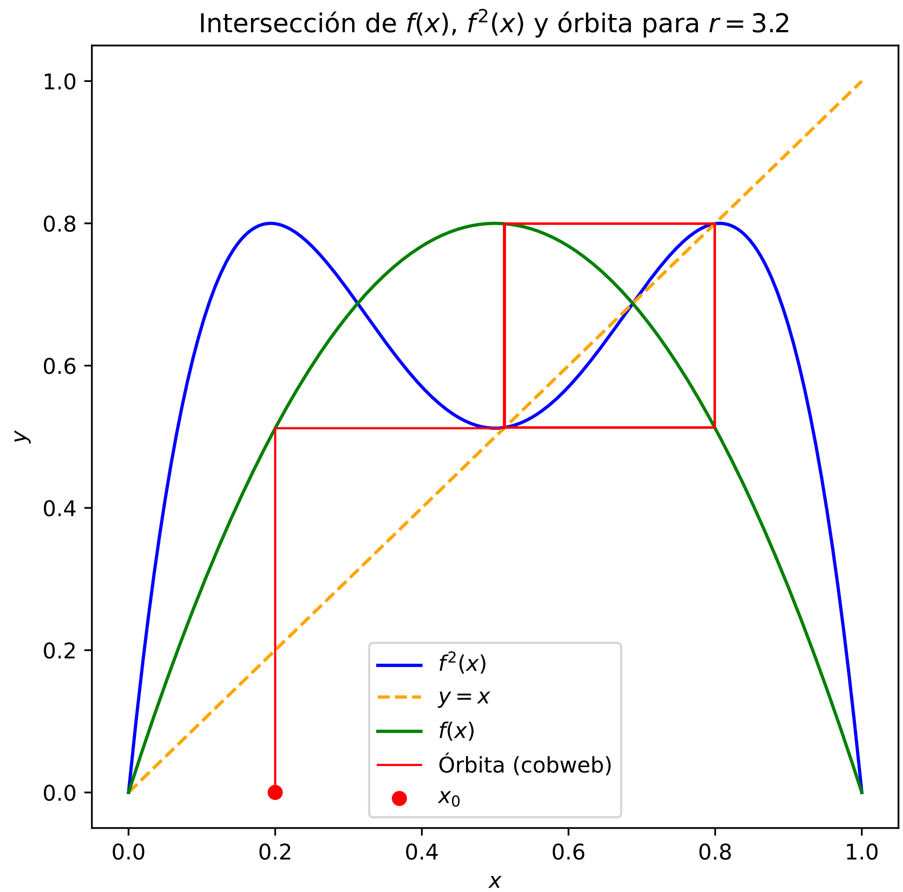

```python
import numpy as np
import matplotlib.pyplot as plt

# Parámetro y condición inicial
r = 3.1
x0 = 0.2
n_iter = 20

# Definición de funciones
def f(x): return r * x * (1 - x)
def f2(x): return f(f(x))

# Dominio
x_vals = np.linspace(0, 1, 400)
y_f = f(x_vals)
y_f2 = f2(x_vals)
```

```
# Construcción del cobweb sobre f
x_coords = [x0]
y_coords = [0]
x, y = x0, 0
for _ in range(n_iter):
    y = f(x)
    x_coords.extend([x, y])
    y_coords.extend([y, y])
    x = y

# Plot
plt.figure(figsize=(6, 6))
plt.plot(x_vals, y_f2, color='blue', label='$f^2(x)$')
plt.plot(x_vals, x_vals, color='orange', linestyle='--', label='$y = x$')
plt.plot(x_vals, y_f, color='green', label='$f(x)$')
plt.plot(x_coords, y_coords, color='red', linewidth=1, label='Órbita (cobweb)')
plt.scatter([x0], [0], marker='o', color='red', label='$x_0$')
plt.title('Intersección de $f(x)$, $f^2(x)$ y órbita para $r=3.1$')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.legend()
plt.tight_layout()
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

# Parámetro y condición inicial
r = 3.2
x0 = 0.2
n_iter = 20

# Definición de funciones
def f(x): return r * x * (1 - x)
def f2(x): return f(f(x))

# Dominio
x_vals = np.linspace(0, 1, 400)
y_f = f(x_vals)
y_f2 = f2(x_vals)
```

```
# Construcción del cobweb sobre f
x_coords = [x0]
y_coords = [0]
x, y = x0, 0
for _ in range(n_iter):
    y = f(x)
    x_coords.extend([x, y])
    y_coords.extend([y, y])
    x = y

# Plot
plt.figure(figsize=(6, 6))
plt.plot(x_vals, y_f2, color='blue', label='$f^2(x)$')
plt.plot(x_vals, x_vals, color='orange', linestyle='--', label='$y = x$')
plt.plot(x_vals, y_f, color='green', label='$f(x)$')
plt.plot(x_coords, y_coords, color='red', linewidth=1, label='Órbita (cobweb)')
plt.scatter([x0], [0], marker='o', color='red', label='$x_0$')
plt.title('Intersección de $f(x)$, $f^2(x)$ y órbita para $r=3.2$')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.legend()
plt.tight_layout()
plt.show()
```



8.8 5. Cálculo de iteraciones

8.8.1 5.1 Condición inicial

Elige $x_0 \in (0, 1)$, p.ej. 0.1, 0.2, 0.5.

8.8.2 5.2 Procedimiento

```
# Iteración del mapa logístico
def iterar_mapa(r, x0, N):
    x = x0
    resultados = []
    for _ in range(N):
        x = r * x * (1 - x)
        resultados.append(x)
```

```
    return resultados
```

Descarta las primeras 100–200 iteraciones (transitorio) antes de analizar el atractor.

8.9 6. Comportamientos según r

Rango de r	Comportamiento
$0 < r < 1$	Convergencia a 0
$1 < r < 3$	Convergencia a $1 - 1/r$
$3 \leq r < 3.449$	Ciclo de periodo 2
$3.449 \leq r < 3.544$	Ciclo de periodo 4
$3.544 \leq r < 3.56995$	Ciclos de periodos 8, 16, 32, …
$3.56995 < r \leq 4$	Régimen caótico con ventanas periódicas

8.10 7. Duplicación de periodo y constante de Feigenbaum

Conforme r crece, aparecen bifurcaciones que duplican el periodo:

1. $r_1 = 3.0 \rightarrow$ periodo 2
2. $r_2 \approx 3.449 \rightarrow$ periodo 4
3. $r_3 \approx 3.544 \rightarrow$ periodo 8 …

La sucesión $\{r_n\}$ converge a:

$$r_\infty \approx 3.56995.$$

Definimos $\Delta r_n = r_n - r_{n-1}$. La razón

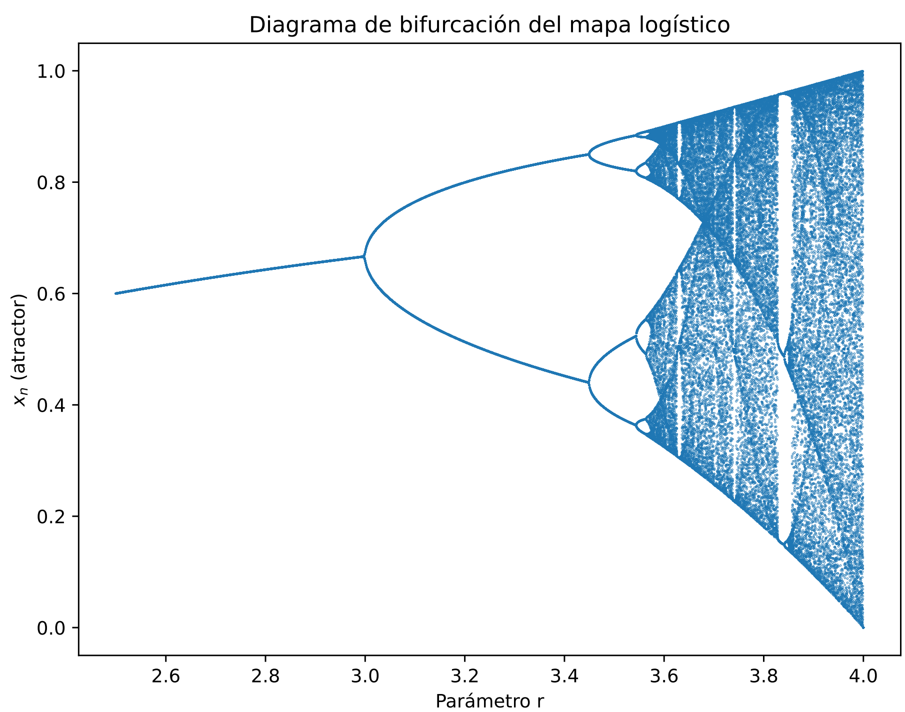
$$\lim_{n\to\infty} \frac{\Delta r_{n-1}}{\Delta r_n} = \delta \approx 4.6692\ldots$$

es la **constante de Feigenbaum**, universal en mapas unimodales.

8.11 8. Diagrama de bifurcación

```
import numpy as np
import matplotlib.pyplot as plt

r_values = np.linspace(2.5, 4.0, 1500)
```

```
iterations, last = 1000, 100
r_plot, x_plot = [], []
for r in r_values:
    x = 0.5
    for _ in range(iterations):
        x = r * x * (1 - x)
    for _ in range(last):
        x = r * x * (1 - x)
        r_plot.append(r)
        x_plot.append(x)

plt.figure(figsize=(8,6))
plt.plot(r_plot, x_plot, '.', markersize=0.5)
plt.title('Diagrama de bifurcación del mapa logístico')
plt.xlabel('Parámetro r')
plt.ylabel('$x_n$ (atractor)')
plt.show()
```



8.12 9. Caos y ventanas periódicas

Cuando el parámetro supera el umbral de acumulación de bifurcaciones

$$r_\infty \approx 3.56995,$$

el mapa entra en **un régimen caótico** caracterizado por varias propiedades fundamentales:

1. **Sensibilidad a las condiciones iniciales.**
   - Dos valores iniciales muy cercanos x_0 y $x_0 + \epsilon$ se separan exponencialmente con el tiempo.
   - Se define el **exponente de Lyapunov**:

   $$\lambda = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} \ln\left|f'(x_n)\right|.$$

   Si $\lambda > 0$, las trayectorias divergen.

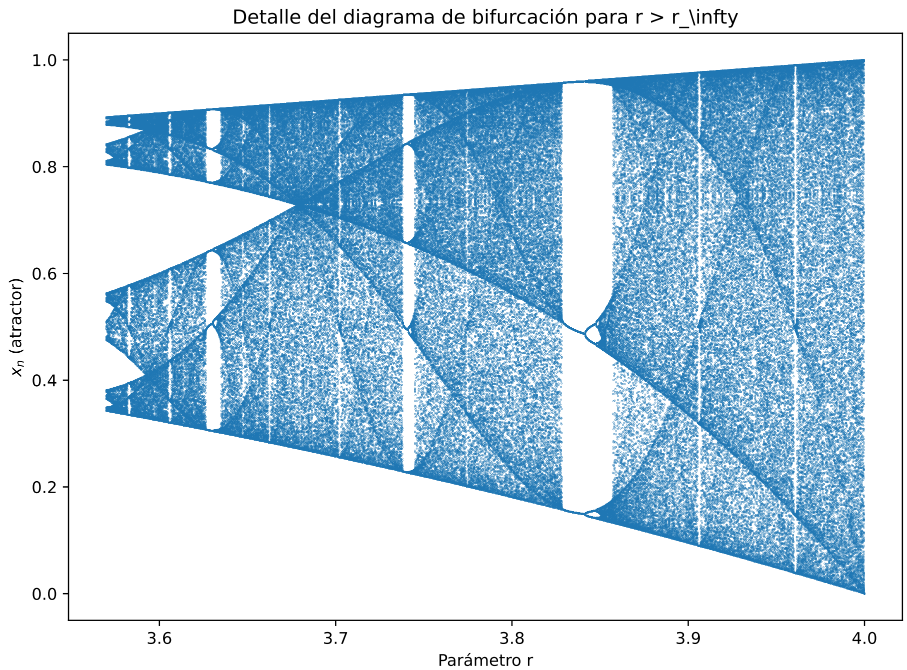
2. **Estructura fractal y autosemejanza.**
   - Aun en la región caótica, aparecen **ventanas periódicas** donde se observan ciclos de período fijo (p.ej., ciclo 3 cerca de $r \approx 3.828$).

3. **Teorema Li–Yorke.**
   - La existencia de un ciclo de período 3 implica ciclos de **todos** los períodos.

8.12.1 9.1 Diagrama detallado para $r > r_\infty$

```
import numpy as np
import matplotlib.pyplot as plt

r_inf = 3.56995
r_values = np.linspace(r_inf, 4.0, 1200)
iterations, last = 1000, 200
r_plot, x_plot = [], []
for r in r_values:
    x = 0.5
    for _ in range(iterations):
        x = r * x * (1 - x)
    for _ in range(last):
        x = r * x * (1 - x)
        r_plot.append(r)
        x_plot.append(x)

plt.figure(figsize=(8,6))
plt.plot(r_plot, x_plot, '.', markersize=0.4)
plt.title('Detalle del diagrama de bifurcación para r > r_\infty')
```

```
plt.xlabel('Parámetro r')
plt.ylabel('$x_n$ (atractor)')
plt.tight_layout()
plt.show()
```



8.13 10. Conclusión y siguientes pasos Conclusión y siguientes pasos

1. Implementa el mapa en Python, R o Excel.
2. Experimenta con distintos r y x_0.
3. Visualiza cobweb plots y diagramas de bifurcación.
4. Estudia la constante de Feigenbaum en otros mapas unimodales.

::: {.quarto-book-part}

`<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4ifQ== -->`{=html}

```{=html}
```

<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4iLCJib29rSXRlbVR5cGUiOiJwYXJ0IiwiYm9va0l0ZW1OdW

Chapter 9

2. El Caos en vivo: El péndulo doble

:::

Chapter 10

test

tttttttttttttttttttttttt

Part II

3. La Meteorología y el Caos

Chapter 11

test

tttttttttttttttttttttttt

Part III

4. El Clima y el Caos

Chapter 12

test

tttttttttttttttttttttttt

Part IV

5. Mis experimentos con el Caos

Chapter 13

test

tttttttttttttttttttttttt