FÍSICA COMPUTACIONAL

TAREA 8

Presentado por:
Cinthia Alejandra Olvera Bautista

Índice general

Introducción

1.1 Oscilador de Van der Pol

El oscilador de Van der Pol es un sistema dinámico no lineal clásico, originalmente propuesto por Balthasar Van der Pol en 1920 para modelar circuitos eléctricos con tubos de vacío. Su ecuación diferencial está dada por:

$$\frac{d^2x}{dt^2} - \mu(1-x^2)\frac{dx}{dt} + x = 0, \tag{1.1}$$

donde μ es un parámetro que controla la no linealidad y la disipación del sistema. Esta ecuación describe un sistema oscilatorio con un amortiguamiento no lineal dependiente de la variable x. Cuando $\mu = 0$, la ecuación se reduce a la ecuación del oscilador armónico simple:

$$\frac{d^2x}{dt^2} + x = 0. \tag{1.2}$$

Figura 1.1: Balthasar Van der Pol. *Imagen recuperada de: https://ethw.org/Balthasar_Van_der_Pol*

Para valores positivos de μ, el sistema presenta un comportamiento no lineal característico, donde pequeñas oscilaciones son amplificadas y grandes oscilaciones son atenuadas, lo que da lugar a la formación de un ciclo límite.

1.1.1 Forma de Primer Orden

Para resolver esta ecuación numéricamente, se reescribe como un sistema de ecuaciones de primer orden introduciendo una nueva variable $y = \frac{dx}{dt}$:

$$\begin{cases} \frac{dx}{dt} = y, \\ \frac{dy}{dt} = \mu(1-x^2)y - x. \end{cases} \tag{1.3}$$

Este sistema es más adecuado para la simulación numérica mediante métodos como Runge-Kutta.

1.1.2 Interpretación Física

El oscilador de Van der Pol puede modelar circuitos eléctricos con elementos no lineales. En particular, se puede asociar a un circuito RLC en serie con un diodo de túnel que introduce una resistencia negativa. La ecuación del circuito equivalente está dada por:

$$L\frac{d^2q}{dt^2} + R(q)\frac{dq}{dt} + \frac{q}{C} = 0, \tag{1.4}$$

donde q es la carga en el capacitor, L es la inductancia, C la capacitancia, y $R(q)$ es una resistencia dependiente de la carga. Si se modela la resistencia como $R(q) = R_0 - \beta q^2$, se obtiene la ecuación de Van der Pol con $x = q$ y μ relacionado con β.

1.1.3 Caso con Resistencia

Si se incluye una resistencia externa R_e, la ecuación se modifica a:

$$\frac{d^2x}{dt^2} - \mu(1-x^2)\frac{dx}{dt} + x + R_e\frac{dx}{dt} = 0. \tag{1.5}$$

Para $R_e = 0$, se recupera la ecuación clásica de Van der Pol. Si $R_e > 0$, la disipación adicional puede modificar el ciclo límite o incluso llevar al sistema a un estado de equilibrio.

1.2 Oscilador de Duffing

El oscilador de Duffing es un sistema dinámico no lineal que describe la evolución de un oscilador con rigidez no lineal. Se utiliza en la modelización de sistemas mecánicos, circuitos eléctricos y en la dinámica de estructuras.

Comenzamos con la ecuación del oscilador armónico simple, derivada de la segunda ley de Newton:

$$F = m\ddot{x} = -kx. \tag{1.6}$$

Reescribiendo en forma estándar,

$$\ddot{x} + \omega_0^2 x = 0, \tag{1.7}$$

donde $\omega_0^2 = \frac{k}{m}$ es la frecuencia natural del oscilador.

Para incluir la no linealidad en la rigidez del sistema, se añade un término cúbico:

$$\ddot{x} + \omega_0^2 x + \beta x^3 = 0. \tag{1.8}$$

La constante β determina la naturaleza de la no linealidad:

Si $\beta > 0$, el sistema se endurece (la frecuencia aumenta con la amplitud).

Si $\beta < 0$, el sistema se ablanda (la frecuencia disminuye con la amplitud).

Este comportamiento es común en materiales elásticos no lineales, resortes mecánicos con comportamiento dependiente de la deformación, y circuitos eléctricos con inductores no lineales.

Añadiendo amortiguamiento proporcional a la velocidad:

$$\ddot{x} + \delta\dot{x} + \omega_0^2 x + \beta x^3 = 0, \tag{1.9}$$

donde δ es el coeficiente de amortiguamiento, que representa pérdidas de energía por fricción o resistencia eléctrica en circuitos.

Finalmente, si el sistema es sometido a una fuerza externa periódica $F(t) = \gamma\cos(\omega t)$, la ecuación completa es:

$$\ddot{x} + \delta\dot{x} + \omega_0^2 x + \beta x^3 = \gamma\cos(\omega t). \tag{1.10}$$

Este modelo describe un oscilador no lineal forzado con amortiguamiento, exhibiendo fenómenos como resonancias múltiples, bifurcaciones y caos, fundamentales en la ingeniería estructural y en sistemas físicos con excitaciones periódicas.

1.3 Sistema de Lorenz

Las ecuaciones de Lorenz modelan la convección térmica en fluidos y fueron derivadas a partir de las ecuaciones de Navier-Stokes bajo ciertas simplificaciones. Son cruciales en meteorología, dinámica de fluidos y sistemas caóticos. Se expresan como un sistema de ecuaciones diferenciales acopladas:

$$\frac{dx}{dt} = \sigma(y - x), \ \frac{dy}{dt} \ = x(\rho - z) - y, \ \frac{dz}{dt} = xy - \beta z. \tag{1.11}$$

Los parámetros físicos tienen el siguiente significado:

x: velocidad del fluido,

y: diferencia de temperatura,

z: variación del perfil térmico.

Los coeficientes son:

$\sigma = 6$: número de Prandtl, que relaciona viscosidad y difusión térmica.

$\rho = 28$: número de Rayleigh, que mide el gradiente de temperatura.

β: parámetro geométrico relacionado con la geometría del sistema.

Este sistema muestra un comportamiento caótico para ciertos valores de los parámetros, lo que llevó al descubrimiento del atractor de Lorenz, una estructura fractal en el espacio fase. Su aplicación es fundamental en la predicción meteorológica y en modelos de turbulencia.

1.4 Modelo de Lotka-Volterra (Presa-Depredador)

El modelo de Lotka-Volterra describe la dinámica poblacional de dos especies que interactúan en un ecosistema: una especie presa y una especie depredadora. Fue desarrollado independientemente por Alfred Lotka en 1925 y Vito Volterra en 1926. Este modelo se aplica en ecología, biología de poblaciones y estudios de sistemas dinámicos en economía y química.

1.4.1 Ecuaciones del Modelo

El sistema de ecuaciones diferenciales que describe la evolución temporal de las poblaciones de presas y depredadores es:

$$\frac{dx}{dt} = \alpha x - \beta xy, \quad \frac{dy}{dt} = \delta xy - \gamma y, \tag{1.12}$$

donde:

$x(t)$ representa la población de presas,

$y(t)$ representa la población de depredadores,

α es la tasa de crecimiento de la población de presas en ausencia de depredadores,

β es la tasa a la que los depredadores cazan presas,

δ es la eficiencia con la que los depredadores convierten presas consumidas en descendencia,

γ es la tasa de mortalidad natural de los depredadores.

1.4.2 Interpretación Física y Biológica

Crecimiento de las presas (αx): En ausencia de depredadores ($y = 0$), la población de presas crecería exponencialmente con una tasa α.

Depredación (βxy): La interacción entre ambas especies reduce la población de presas de manera proporcional al producto xy, reflejando la probabilidad de encuentros entre presas y depredadores.

Crecimiento de los depredadores (δxy): A medida que los depredadores consumen presas, su población aumenta en proporción a la cantidad de presas cazadas.

Mortalidad de los depredadores (γy): Si no hay presas ($x = 0$), la población de depredadores disminuye exponencialmente con una tasa de mortalidad γ.

Este modelo asume que el ambiente es constante, no hay migración ni competencia intraespecífica, y las tasas de interacción son constantes en el tiempo.

1.4.3 Extensiones del Modelo

El modelo de Lotka-Volterra básico puede modificarse para incluir efectos adicionales:

Capacidad de carga de la presa: Se agrega un término logístico para evitar crecimiento ilimitado de la presa:

$$\frac{dx}{dt} = \alpha x\left(1 - \frac{x}{K}\right) - \beta xy. \tag{1.13}$$

Competencia intraespecífica del depredador: Se introduce un término de competencia para reflejar la lucha por recursos dentro de la población de depredadores.

Factores ambientales variables: Se modelan variaciones estacionales en los parámetros, como disponibilidad de alimento.

Metodología

En esta sección s muestran ejemplos de aplciación de los temas incluídos en introducción.

2.1 Oscilador Van der Poll

2.1.1 Ejemplo de aplicación

La ecuación de Van der Pol se define como:

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0. \tag{2.1}$$

Se pide:

Resolver la ecuación para $\mu = 2$ en el intervalo $t \in [0, 20]$ con condiciones iniciales $x(0) = 2$, $\dot{x}(0) = 0$.

Graficar $x(t)$ y el espacio fase.

El programa en python se puede encontrar en el anexo duffing.py

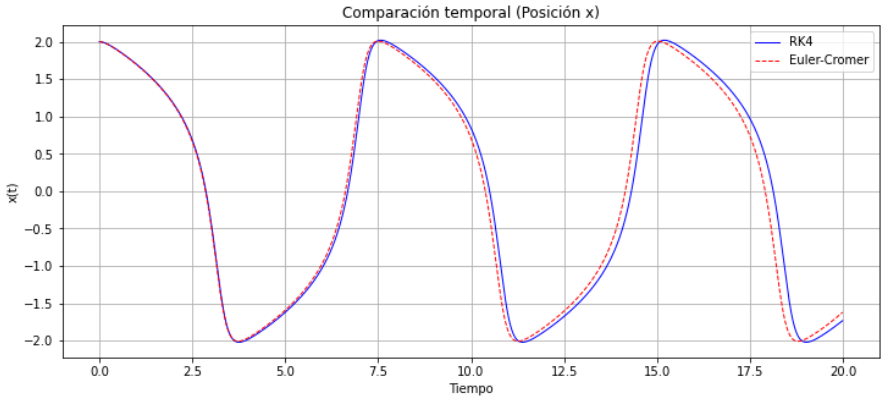

Figura 2.1: Posición y tiempo con euler cromer y RK4

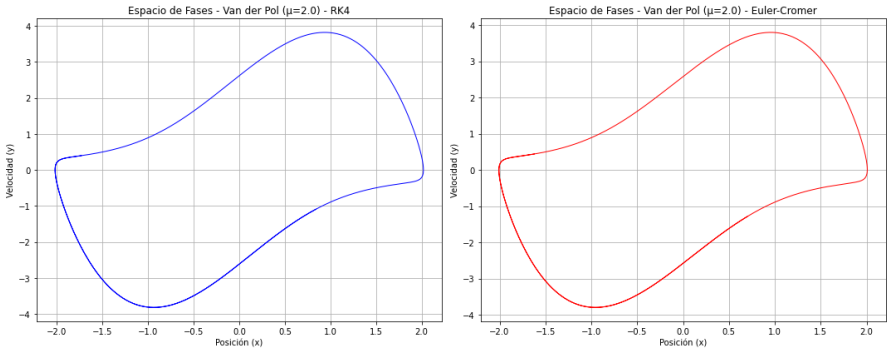

Figura 2.2: Espacio fase con ambos métodos

2.2 Duffing

2.2.1 Ejemplo de aplicación

Considere la ecuación de Duffing con parámetros específicos:

$$\alpha = -1{,}0, \quad \beta = 0{,}1, \quad \delta = 0{,}2, \quad \gamma = 0{,}5, \quad \omega = 2{,}4 \tag{2.2}$$

Realice lo siguiente:

Resolver numéricamente la ecuación en el intervalo $t \in [0, 100]$ con condiciones iniciales $x(0) = 0{,}1$, $\dot{x}(0) = 0$.

Graficar la evolución de $x(t)$.

Graficar el espacio fase $(x, \dot{x})$.

El programa en python se puede encontrar en el anexo duffing.py

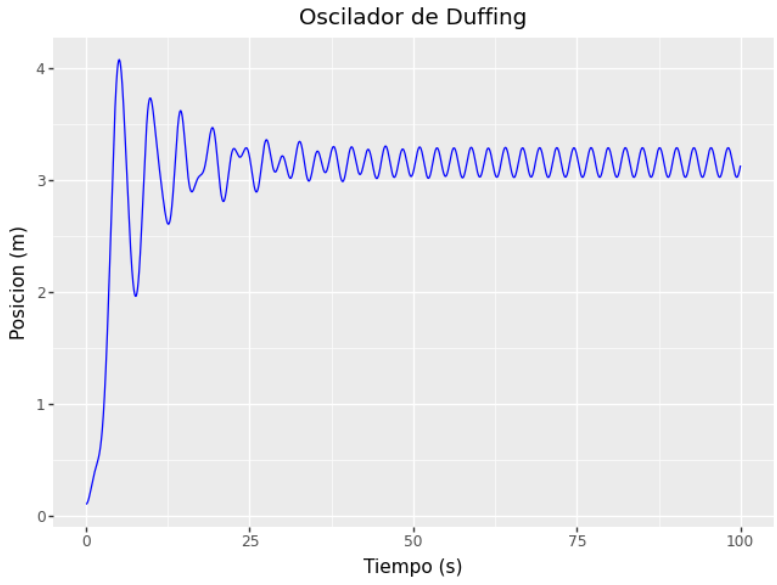

Figura 2.3: Posición en el tiempo

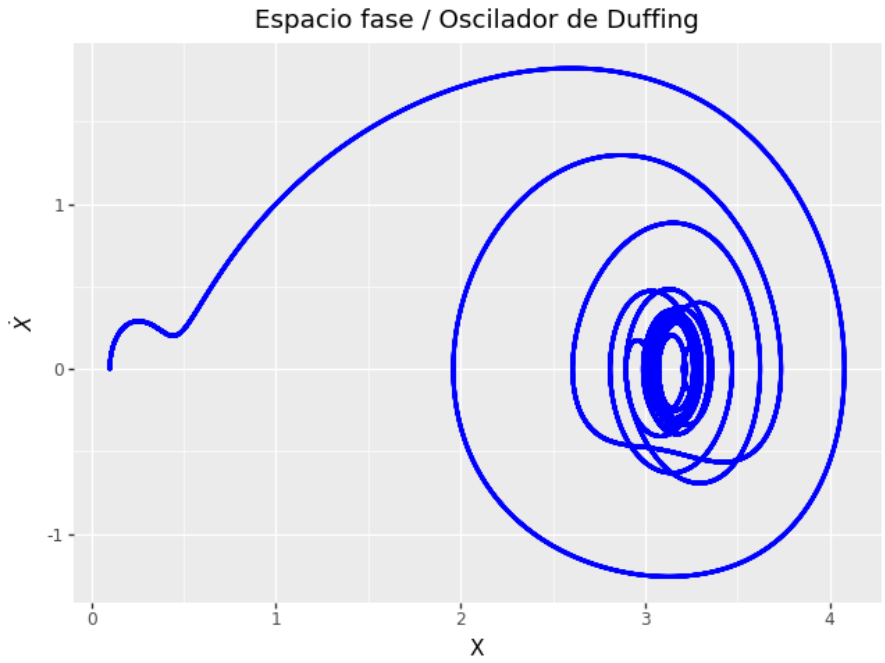

Figura 2.4: Espacio fase

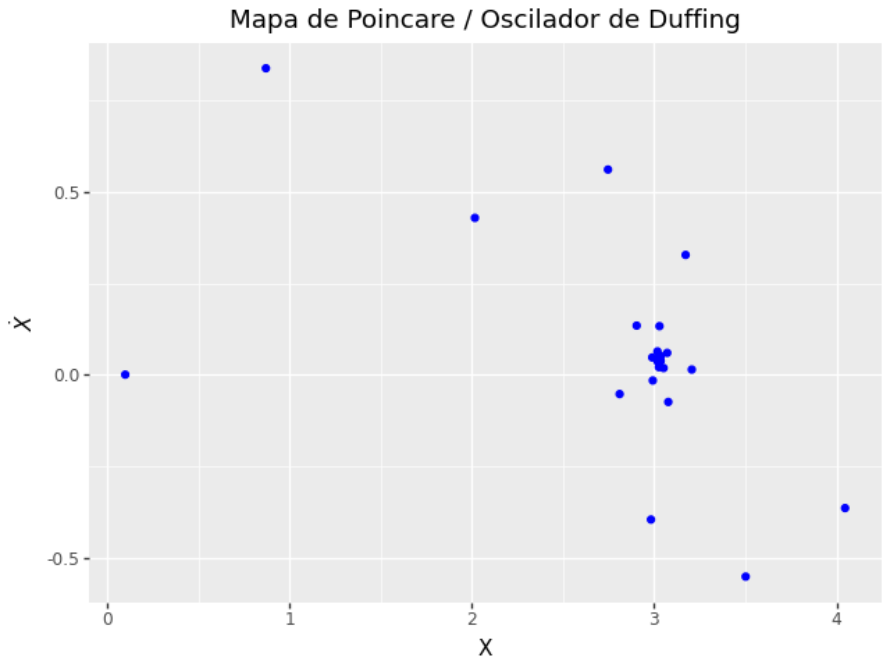

Figura 2.5: Mapa de poincaré

2.3 Lorenz

2.3.1 Ejemplo de aplicación

El sistema de Lorenz está dado por:

$$\begin{aligned}\frac{dx}{dt} &= 10(y-x),\\ \frac{dy}{dt} &= x(28-z)-y,\\ \frac{dz}{dt} &= xy-\frac{8}{3}z.\end{aligned} \tag{2.3}$$

El programa en python se puede encontrar en el anexo lorenz.py

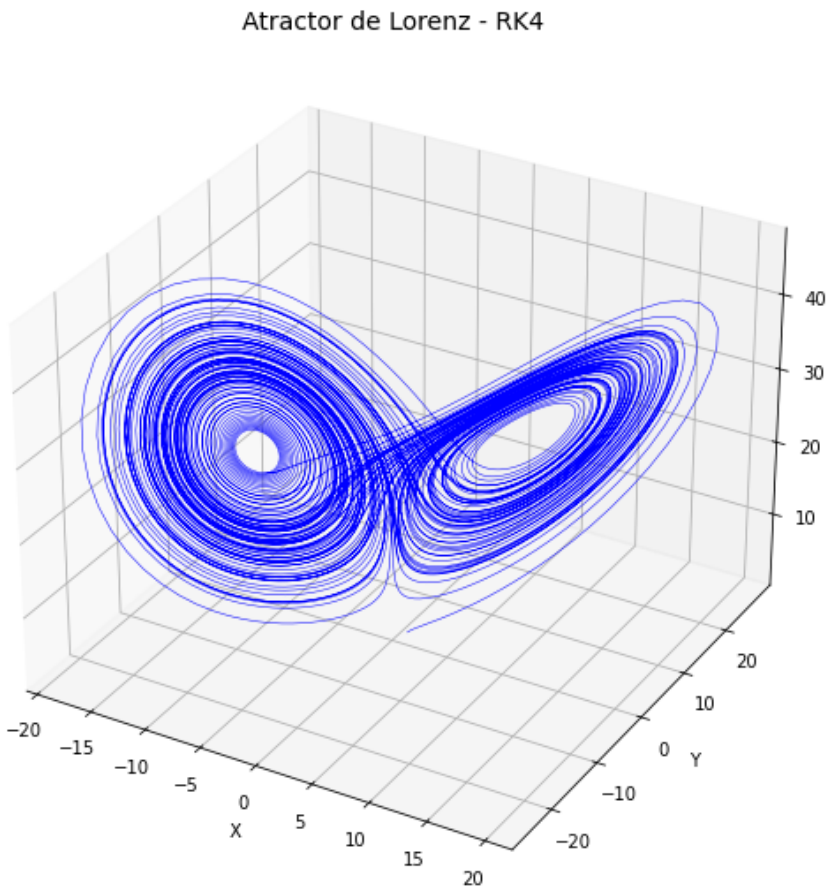

Figura 2.6: Atractor de lorenz

2.4 Lotka Volterra

2.4.1 Ejemplo de Aplicación

Un ecosistema con conejos como presas y zorros como depredadores puede modelarse con los siguientes parámetros:

$$\alpha = 0{,}5, \quad \beta = 0{,}02, \quad \delta = 0{,}01, \quad \gamma = 0{,}3. \tag{2.4}$$

Si inicialmente hay 40 conejos y 10 zorros, la evolución de sus poblaciones mostrará oscilaciones

periódicas: cuando los conejos aumentan, los zorros también, pero luego el exceso de depredación reduce la población de conejos, lo que a su vez provoca la disminución de los zorros.

El programa en python se puede encontrar en el anexo lotkavolterra.py

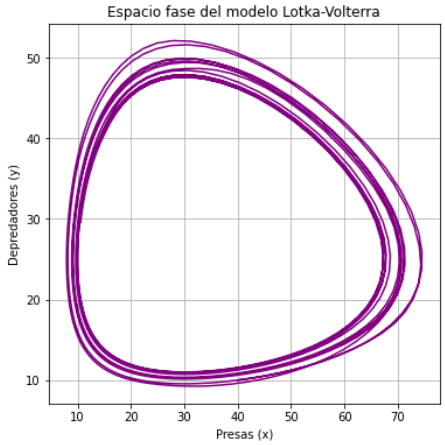

Figura 2.7: Espacio fase Lotka, voltera

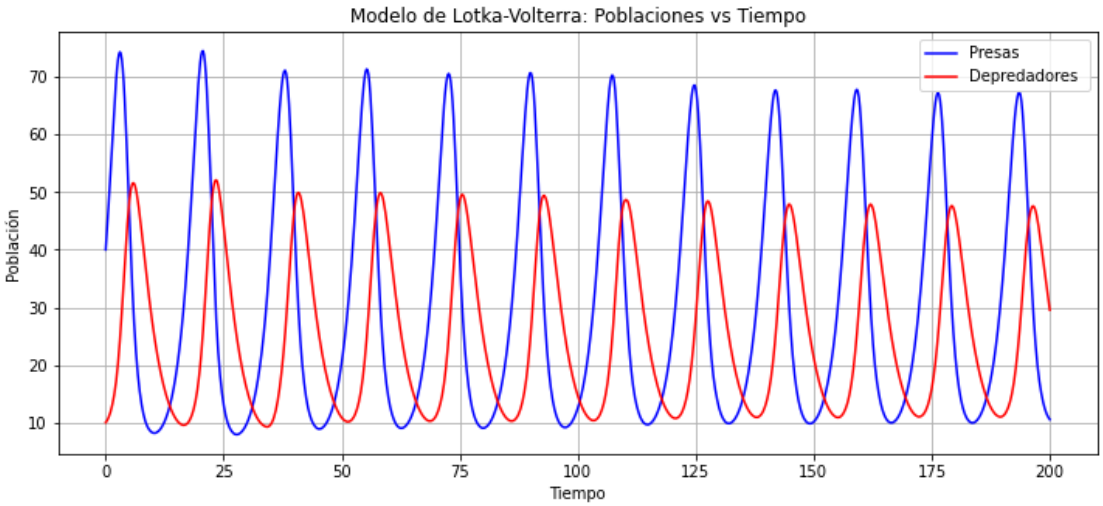

Figura 2.8: Poblaciones versus tiempo Lotka, voltera

12

Conclusión

Anexos

Los códigos se encuentran en el link https://github.com/cinthia-bao/Fisica-Computacional-.git en la carpeta **TAREA 8**.

4.1 duffing.py

```
import numpy as np
import pandas as pd
from pi import *

# Parametros de la ecuacion de Duffing
alpha = -1.0
beta = 0.1
delta = 0.2
gamma = 0.5
omega = 2.4

# Parametros para la simulacion
dt = 0.001
t_max = 100
npasos = int(t_max / dt)
T = 2 * np.pi / omega


# Condiciones iniciales
x = 0.1
v = 0.0

xi = []
vi = []
```

```
vip = []
xip = []
tip = []
ti = np.linspace(0,t_max,npasos)

# Recordemos que la ecuacion de Duffing es: d2x/dt2 + delta*dx/dt + alpha*
    x + beta*x**3 = f0*c0s(om*t)
# Hacemos que v = dx/dt y a = dv/dt
for t in ti:
    a = -delta * v - alpha*x - beta*x**3 + gamma*np.cos(omega*t)
    v += a * dt
    x += v * dt
    xi.append(x)
    vi.append(v)
    if np.isclose(t % T, 0, atol=dt):
        xip.append(x)
        vip.append(v)
        tip.append(t)

df = pd.DataFrame({
    'Tiempo': ti,
    'Posicion': xi,
    'Velocidad':vi
})

dfp = pd.DataFrame({
    'Tiempo': tip,
    'Posicion': xip,
    'Velocidad':vip
})


plot = (
    ggplot(df, aes(x='Tiempo',y='Posicion')) +
    geom_line(color='blue') +
    labs(title='Oscilador de Duffing', x='Tiempo (s)', y='Posicion (m)')
)

plot.show()
```

```
ef = (
    ggplot(df, aes(x='Posicion',y='Velocidad')) +
    geom_point(color='blue', size = 0.2) +
    labs(title='Espacio fase / Oscilador de Duffing', x='X', y=r'$\dot{X}$'
        )
)

ef.show()


poincare = (
    ggplot(dfp, aes(x='Posicion',y='Velocidad')) +
    geom_point(color='blue') +
    labs(title='Mapa de Poincare / Oscilador de Duffing', x='X', y=r'$\dot{
        X}$')
)

poincare.show()
```

4.2 lorenz.py

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

# Parmetros del sistema de Lorenz
sigma, rho, beta = 10.0, 28.0, 8.0/3.0

# Funcin del sistema de Lorenz
def lorenz(state):
    x, y, z = state
    dxdt = sigma * (y - x)
    dydt = x * (rho - z) - y
    dzdt = x * y - beta * z
    return np.array([dxdt, dydt, dzdt])

# Mtodo RK4
def rk4_step(state, dt):
    k1 = lorenz(state)
    k2 = lorenz(state + 0.5 * dt * k1)
    k3 = lorenz(state + 0.5 * dt * k2)
    k4 = lorenz(state + dt * k3)
    return state + (dt / 6.0) * (k1 + 2*k2 + 2*k3 + k4)

# Simulacin
dt = 0.01
steps = 10000
state = np.array([1.0, 1.0, 1.0]) # Condicin inicial

# Almacenar resultados
trajectory = np.zeros((steps, 3))
for i in range(steps):
    trajectory[i] = state
    state = rk4_step(state, dt)

# Grfico 3D
fig = plt.figure(figsize=(12, 9))
ax = fig.add_subplot(111, projection='3d')
x, y, z = trajectory.T
ax.plot(x, y, z, lw=0.5, color='blue')
```

```
ax.set_title("Atractor de Lorenz - RK4", fontsize=14)
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
plt.show()
```

4.3 lotkavolterra.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 3 13:12:06 2025

@author: ale
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# Definimos el sistema de ecuaciones Lotka-Volterra
def lotka_volterra(t, z, alpha, beta, delta, gamma):
    x, y = z
    dxdt = alpha * x - beta * x * y
    dydt = delta * x * y - gamma * y
    return [dxdt, dydt]

# Parmetros del sistema
alpha = 0.5 # Tasa de crecimiento de la presa
beta = 0.02 # Tasa de depredacin
delta = 0.01 # Tasa de conversin de presas en depredadores
gamma = 0.3 # Tasa de mortalidad del depredador
# Condiciones iniciales
x0 = 40 # Poblacin inicial de presas
y0 = 10 # Poblacin inicial de depredadores
z0 = [x0, y0]

# Tiempo de simulacin
t_span = (0, 200) # Intervalo de tiempo
sol = solve_ivp(lotka_volterra, t_span, z0, args=(alpha, beta, delta,
    gamma), dense_output=True)
t = np.linspace(0, 200, 1000)
z = sol.sol(t)

# Graficamos las poblaciones en funcin del tiempo
plt.figure(figsize=(12, 5))
```

```
plt.plot(t, z[0], label="Presas ", color="blue")
plt.plot(t, z[1], label="Depredadores ", color="red")
plt.xlabel("Tiempo")
plt.ylabel("Poblacin")
plt.title("Modelo de Lotka-Volterra: Poblaciones vs Tiempo")
plt.legend()
plt.grid()
plt.show()

# Graficamos el espacio fase
plt.figure(figsize=(6, 6))
plt.plot(z[0], z[1], color="purple")
plt.xlabel("Presas (x)")
plt.ylabel("Depredadores (y)")
plt.title("Espacio fase del modelo Lotka-Volterra")
plt.grid()
plt.show()
```

4.4 vanderpoll.py

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

# Parmetros del Van der Pol
mu = 2.0 # Parmetro de no linealidad (prueba con 0.1, 1.0, 5.0)

# Definicin del sistema
def van_der_pol(state):
    x, y = state
    dxdt = y
    dydt = mu * (1 - x**2) * y - x
    return np.array([dxdt, dydt])

# Mtodo RK4
def rk4_step(state, dt):
    k1 = van_der_pol(state)
    k2 = van_der_pol(state + 0.5 * dt * k1)
    k3 = van_der_pol(state + 0.5 * dt * k2)
    k4 = van_der_pol(state + dt * k3)
    return state + (dt / 6.0) * (k1 + 2*k2 + 2*k3 + k4)

# Mtodo Euler-Cromer adaptado
def euler_cromer_step(state, dt):
    x, y = state
    # Primero actualizamos la velocidad (y)
    new_y = y + dt * (mu * (1 - x**2) * y - x)
    # Luego la posicin (x) con la nueva velocidad
    new_x = x + dt * new_y
    return np.array([new_x, new_y])

# Simulacin
dt = 0.01
fin = 20
steps = int(fin/dt)
state_rk4 = np.array([2, 0]) # Condicin inicial
state_ec = state_rk4.copy()

# Almacenamiento
```

```
traj_rk4 = np.zeros((steps, 2))
traj_ec = np.zeros((steps, 2))

for i in range(steps):
    traj_rk4[i] = state_rk4
    traj_ec[i] = state_ec
    state_rk4 = rk4_step(state_rk4, dt)
    state_ec = euler_cromer_step(state_ec, dt)

# Grfico del espacio de fases
plt.figure(figsize=(15, 6))

# RK4
plt.subplot(121)
x_rk4, y_rk4 = traj_rk4[1000:].T # Ignoramos transitorio inicial
plt.plot(x_rk4, y_rk4, 'b', lw=1, label=f'RK4 (dt={dt})')
plt.title(f'Espacio de Fases - Van der Pol (={mu}) - RK4')
plt.xlabel('Posicin (x)')
plt.ylabel('Velocidad (y)')
plt.grid(True)

# Euler-Cromer
plt.subplot(122)
x_ec, y_ec = traj_ec[1000:].T
plt.plot(x_ec, y_ec, 'r', lw=1, label=f'Euler-Cromer (dt={dt})')
plt.title(f'Espacio de Fases - Van der Pol (={mu}) - Euler-Cromer')
plt.xlabel('Posicin (x)')
plt.ylabel('Velocidad (y)')
plt.grid(True)

plt.tight_layout()
plt.show()

# Diagrama de tiempo para comparar mtodos
plt.figure(figsize=(12, 5))
plt.plot(np.arange(steps)*dt, traj_rk4[:, 0], 'b-', lw=1, label='RK4')
plt.plot(np.arange(steps)*dt, traj_ec[:, 0], 'r--', lw=1, label='Euler-
    Cromer')
plt.title('Comparacin temporal (Posicin x)')
plt.xlabel('Tiempo')
```

```
plt.ylabel('x(t)')
plt.legend()
plt.grid(True)
plt.show()
```