UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

Facultad de Ingeniería de Sistemas e Informática

Proyecto Final de Análisis Numérico

Simulación Numérica y Métodos Iterativos

Curso:	Análisis Numérico
Docente:	Dr. Richard Cubas Becerra
Integrantes:	Diego Sotelo
	Alexis Gonzales
	Paolo Villavicencio
	Álvaro Salazar
Fecha:	2 de diciembre de 2025

Índice

1. Resumen

En este trabajo se estudia la dinámica depredador–presa mediante el modelo clásico de Lotka–Volterra, formulado como un sistema de ecuaciones diferenciales ordinarias acopladas. Dado que este sistema no admite en general una solución analítica sencilla para tiempos arbitrarios, se recurre a métodos numéricos para aproximar la evolución temporal de las poblaciones. En particular, se implementa desde cero el método de Runge–Kutta de cuarto orden (RK4) en el lenguaje de programación Python, debido a su buen equilibrio entre precisión y costo computacional frente a métodos de menor orden, como el de Euler. Sobre esta base se desarrolla un simulador que permite configurar parámetros iniciales, generar trayectorias numéricas y visualizar los resultados mediante gráficos y una animación construida con la librería Manim. Finalmente, se presentan una simulación base y diversos experimentos variando los parámetros del modelo, analizando cómo estos cambios afectan la estabilidad del sistema y las oscilaciones observadas en el plano de fases.

Palabras clave: Lotka–Volterra, Runge–Kutta de cuarto orden, métodos numéricos, Python, Manim, simulación.

2. Abstract

This work studies predator–prey dynamics through the classical Lotka–Volterra model, formulated as a system of coupled ordinary differential equations. Since this system does not usually admit a simple closed–form solution for arbitrary times, numerical methods are required to approximate the temporal evolution of both populations. In particular, we implement from scratch the fourth–order Runge–Kutta method (RK4) in the Python programming language, due to its favorable balance between accuracy and computational cost when compared with lower–order methods such as the explicit Euler scheme. On top of this numerical core, we develop a simulation tool that allows the user to set initial parameters, generate numerical trajectories, and visualize the results through plots and an animation created with the Manim library. A baseline simulation and several experiments with different parameter values are presented, in order to analyze how these modifications affect the stability of the system and the oscillatory behavior observed in the phase plane.

Keywords: Lotka–Volterra, fourth–order Runge–Kutta, numerical methods, Python, Manim, simulation.

3. Introducción

3.1. Descripción del Problema:

El modelado matemático de sistemas ecológicos constituye un pilar esencial dentro de la biología teórica. Entre los fenómenos más estudiados destaca la dinámica depredador–presa, caracterizada por oscilaciones cíclicas en las poblaciones de ambas especies. En este tipo de interacción, un incremento en la población de presas (por ejemplo, liebres) genera una mayor disponibilidad de alimento, permitiendo que los depredadores (como los linces) aumenten su población. Con el tiempo, la mayor presión de depredación reduce la abundancia de presas, lo que posteriormente provoca una disminución en la población de depredadores por escasez de alimento, permitiendo que el ciclo se reinicie. Este comportamiento cíclico, ampliamente documentado en la ecología matemática, ha sido señalado como uno de los fundamentos históricos del desarrollo de la ecología teórica moderna (Kingsland, 2015).

3.2. Planteamiento del Proyecto:

Para traducir esta dinámica biológica en un modelo cuantitativo, este proyecto se basa en el sistema formulado independientemente por Alfred J. Lotka (1925) y Vito Volterra (1926). Ambos propusieron un sistema de ecuaciones diferenciales ordinarias (EDOs) acopladas que hoy se conoce como el modelo de Lotka-Volterra, que aunque relativamente simple en su planteamiento, capta la esencia de la dependencia mutua y oscilatoria entre las especies. Dado que este sistema general no ofrece una solución analítica simple para tiempos arbitrarios, el proyecto busca resolverlo numéricamente y desarrollar un simulador computacional como herramienta de exploración.

3.3. Modelo y Método:

Para resolver el sistema de EDOs del modelo de Lotka-Volterra, se requiere un método numérico robusto. Métodos simples como el de Euler tienen ventajas de sencillez, pero tienden a acumular errores de truncamiento en simulaciones prolongadas. Por ello, este trabajo opta por emplear el método de Runge-Kutta de cuarto orden (RK4), reconocido por su buen equilibrio entre precisión y coste computacional: al evaluar la pendiente en varios puntos intermedios, mejora significativamente la aproximación respecto a los métodos de orden inferior. El objetivo es implementar este algoritmo desde cero en Python, generar los datos de simulación y visualizar la evolución de las poblaciones en el tiempo.

4. Objetivos

4.1. Objetivo general

Crear un programa que simule la evolución de las poblaciones (depredador-presa) a lo largo del tiempo, aplicando los métodos numéricos estudiados en el curso para representar los resultados mediante gráficos y una animación.

4.2. Objetivos específicos

- Analizar el modelo matemático de Lotka-Volterra que describe la interacción entre depredadores y presas.
- Desarrollar una implementación propia del método Runge-Kutta de 4º orden (RK4) para la solución numérica del sistema de EDOs, aplicando la teoría de la Unidad IV del curso.
- Codificar un programa en Python que emplee el método RK4 para calcular la evolución de las poblaciones a través del tiempo.
- Diseñar visualizaciones de datos para ilustrar los resultados, incluyendo un gráfico de población contra tiempo y un diagrama de fase que muestre la relación entre especies.
- Evaluar el comportamiento del ecosistema mediante la experimentación con distintos parámetros iniciales (ej. tasas de natalidad o eficiencia de caza) y analizar su impacto en la estabilidad del sistema.

5. Marco teórico y Metodología

5.1. Modelos Matemáticos

Los métodos numéricos permiten aproximar soluciones cuando las formas analíticas son inviables. En este trabajo se consideran tanto sistemas lineales como ecuaciones diferenciales ordinarias.

$$A\mathbf{x} = \mathbf{b}, \quad \frac{dy}{dt} = f(t, y)$$

5.2. Ecuación Diferencial Lotka-Volterra

El modelo depredador-presa de Lotka-Volterra se define mediante un sistema que incluye las siguientes dos ecuaciones diferenciales ordinarias:

$$\begin{cases} \dfrac{dx}{dt} = \alpha x - \beta xy & \text{(Ecuación de la presa)} \\ \dfrac{dy}{dt} = -\gamma y + \delta xy & \text{(Ecuación del depredador)} \end{cases} \tag{1}$$

donde las constantes positivas representan:

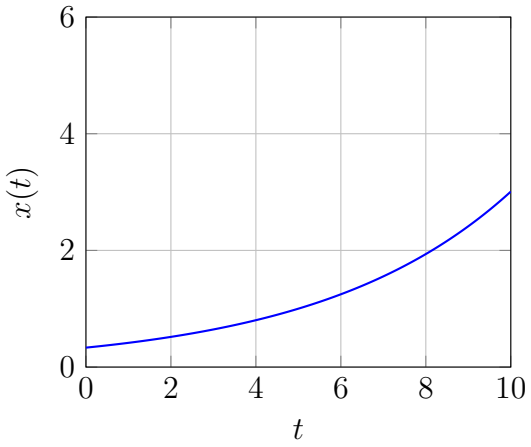
- α : tasa de crecimiento de las presas.
- β : tasa de encuentros presa–depredador que reduce la población de presas.
- γ : tasa de mortalidad de los depredadores en ausencia de presas.
- δ : incremento en la población de depredadores debido al consumo de presas.

Los puntos de equilibrio del sistema se obtienen igualando las derivadas a cero:

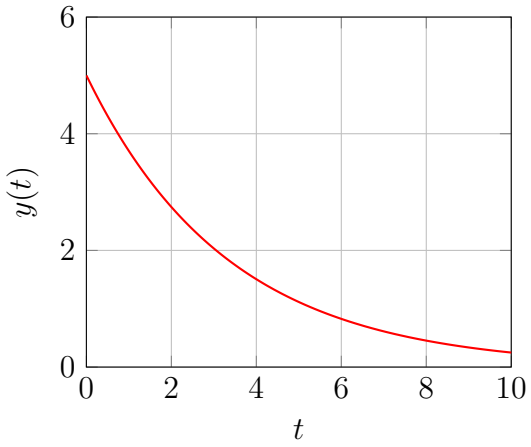
$$(x^*, y^*) = (0, 0), \qquad (\bar{x}, \bar{y}) = \left(\frac{\gamma}{\delta}, \frac{\alpha}{\beta}\right).$$

El equilibrio trivial $(0, 0)$ corresponde a la extinción de ambas poblaciones, mientras que $(\bar{x}, \bar{y})$ describe un equilibrio no trivial donde coexisten presas y depredadores, alrededor del cual surgen órbitas cerradas (curvas ovoidales) en el plano de fases.

Visualización cualitativa del modelo Lotka–Volterra



(a) Evolución de la población de presas cuando no hay depredadores.



(b) Evolución de la población de depredadores cuando no hay presas.

Figura 1: Escenarios extremos del modelo Lotka–Volterra: solo presas y solo depredadores.

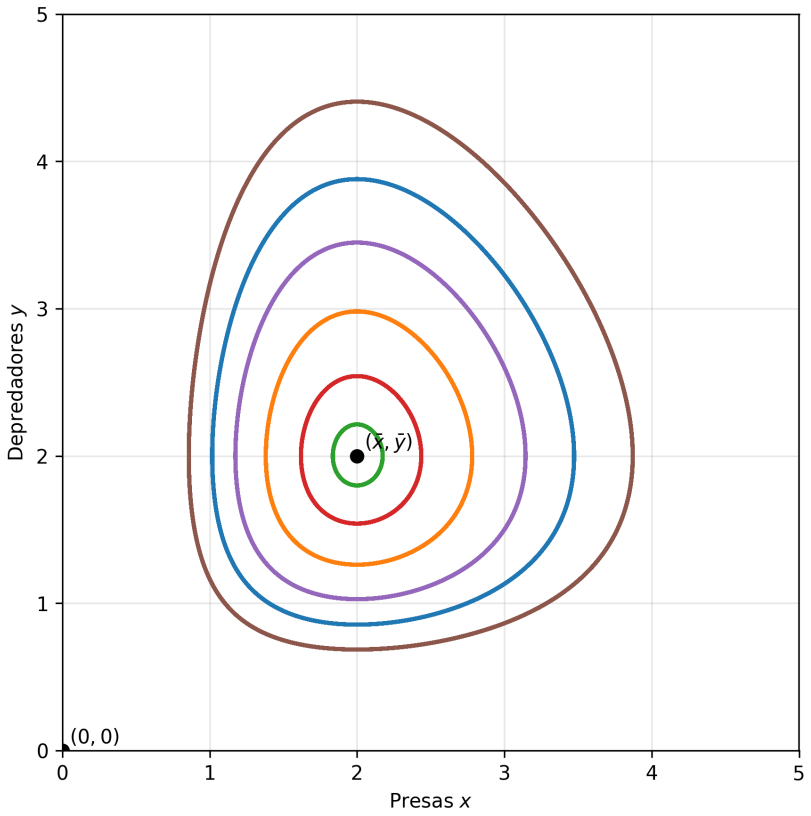


Figura 2: Plano de fases del modelo Lotka–Volterra con órbitas cerradas irregulares alrededor del punto de equilibrio no trivial.

5.3. Runge–Kutta de cuarto orden

Problema de valor inicial y métodos de un paso

Para resolver numéricamente el PVI general $y'(t) = f(t, y)$, $y(t_0) = y_0$ se emplean con frecuencia métodos de un paso, que construyen una sucesión $\{y_k\}_{k=0}^{N}$ en puntos $t_k = t_0 + kh$ mediante una regla de actualización

$$y_{k+1} = \Phi(t_k, y_k, h),$$

donde Φ es una función que depende de f, del valor actual y_k y del tamaño de paso h. Ejemplos clásicos de este tipo son el método de Euler explícito y los métodos de Runge–Kutta, tal como se presenta en los textos de Burden y Faires (2011) y de Chapra y Canale (2015).

Los métodos de Taylor de orden p logran gran precisión, pero requieren derivadas de alto orden de f, lo cual resulta poco práctico en muchos problemas reales. Los métodos de Runge–Kutta se diseñan precisamente para alcanzar órdenes altos de precisión imitando a los métodos de Taylor, pero sin calcular derivadas superiores: en su lugar, evalúan varias veces la función $f(t, y)$ en puntos distintos dentro del intervalo $[t_k, t_{k+1}]$ y combinan esas pendientes de forma adecuada, como se discute en detalle en Kiusalaas (2013) y Parker (2021).

Formulación del método RK4

El método clásico de Runge–Kutta de cuarto orden (RK4) se define, para el PVI $y'(t) = f(t, y)$, $y(t_0) = y_0$, mediante las pendientes intermedias

$$k_1 = f(t_k, y_k), \tag{2}$$

$$k_2 = f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2}k_1\right), \tag{3}$$

$$k_3 = f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2}k_2\right), \tag{4}$$

$$k_4 = f(t_k + h, y_k + hk_3), \tag{5}$$

y la actualización

$$y_{k+1} = y_k + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right). \tag{6}$$

En el caso de sistemas vectoriales, como el modelo Lotka–Volterra, y_k es un vector y las pendientes k_i se calculan componente a componente usando la función vectorial f correspondiente, tal como se describe en textos clásicos de análisis numérico de Burden y

Faires, y de Kiusalaas. La implementación genérica de este esquema se muestra en la función `runge_kutta4` del Listado 1 (ver Anexos), sobre la cual se especializa posteriormente la función `rk4_lotka` para el sistema depredador–presa.

Orden y error del método

Suponiendo que la solución exacta $y(t)$ del problema es suficientemente suave (por ejemplo, $y \in C^5$ en el intervalo de integración), se sabe por los resultados teóricos presentados en Burden y Faires y en Parker que el método RK4 presenta un error local de truncamiento de orden $\mathcal{O}(h^5)$ y un error global acumulado de orden $\mathcal{O}(h^4)$. En términos prácticos, esto significa que, si se reduce el tamaño de paso h a la mitad, el error global se reduce aproximadamente por un factor de $2^4 = 16$, siempre que el error de redondeo no sea dominante.

Desde un punto de vista integral, el avance de la solución entre dos instantes t_0 y $t_1 = t_0 + h$ puede escribirse como

$$y(t_1) - y(t_0) = \int_{t_0}^{t_1} f(t, y(t))\, dt.$$

El esquema RK4 puede interpretarse como una aproximación de esta integral mediante una regla compuesta similar a la de Simpson con subpaso $h/2$: las cuatro evaluaciones k_1, k_2, k_3, k_4 juegan el papel de pendientes en el inicio, puntos medios y final del intervalo, y la combinación ponderada

$$\frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

aproxima el valor medio de f en $[t_k, t_{k+1}]$, tal como se discute en Parker.

Ejemplo de aplicación del método RK4

Como ejemplo ilustrativo, consideremos el problema

$$y'(t) = \frac{t - y}{2}, \qquad y(0) = 1,$$

con tamaño de paso $h = 0{,}25$ en el intervalo $[0, 3]$. Calcularemos el primer paso desde $t_0 = 0$ hasta $t_1 = 0{,}25$.

La función es $f(t,y) = \dfrac{t-y}{2}$. Se tienen:

$$\begin{aligned}
k_1 &= f(t_0, y_0) = \frac{0-1}{2} = -0{,}5,\\
k_2 &= f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1\right) = f(0{,}125,\ 1 + 0{,}125(-0{,}5))\\
&= f(0{,}125,\ 0{,}9375) = \frac{0{,}125 - 0{,}9375}{2} = -0{,}40625,\\
k_3 &= f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_2\right) = f(0{,}125,\ 1 + 0{,}125(-0{,}40625))\\
&= f(0{,}125,\ 0{,}94921875) \approx -0{,}4121094,\\
k_4 &= f(t_0 + h, y_0 + h\,k_3) = f(0{,}25,\ 1 + 0{,}25(-0{,}4121094))\\
&= f(0{,}25,\ 0{,}89697) \approx -0{,}3234863.
\end{aligned}$$

Aplicando la fórmula de actualización (6):

$$\begin{aligned}
y_1 &= y_0 + \frac{h}{6}\big(k_1 + 2k_2 + 2k_3 + k_4\big)\\
&\approx 1 + \frac{0{,}25}{6}\left(-0{,}5 - 0{,}8125 - 0{,}8242 - 0{,}3235\right)\\
&\approx 0{,}89749.
\end{aligned}$$

Por lo tanto, la aproximación obtenida es

$$y(0{,}25) \approx 0{,}89749.$$

Este tipo de cálculo paso a paso es el que realiza el programa de forma automática al integrar el modelo de Lotka–Volterra.

Ventajas frente a otros métodos

En la familia de métodos de un paso, RK4 ofrece un compromiso muy favorable entre precisión y costo computacional:

- El método de Euler explícito requiere una sola evaluación de f por paso, pero sólo es de orden $\mathcal{O}(h)$ y puede presentar problemas de estabilidad para ciertos tamaños de paso.
- Los métodos de segundo orden (Euler mejorado o método de Heun, entre otros) mejoran la precisión a costa de dos evaluaciones de f por paso y un error global de orden $\mathcal{O}(h^2)$.

- RK4 necesita cuatro evaluaciones de f por paso, pero alcanza un error global de orden $\mathcal{O}(h^4)$, lo que permite usar tamaños de paso relativamente grandes manteniendo una buena precisión, como se discute en los textos de Burden y Faires, Chapra y Canale, y Kiusalaas.

Por estas razones, RK4 se considera un método estándar en cursos introductorios de análisis numérico y en aplicaciones prácticas donde se requiere un algoritmo robusto, preciso y fácil de programar. En este proyecto se adopta como método principal para integrar el sistema de Lotka–Volterra.

Comparación numérica con otros métodos

Para ilustrar de manera cuantitativa la diferencia entre Euler, Heun y RK4, consideremos el problema sencillo

$$y'(t) = -2y(t), \qquad y(0) = 1,$$

cuya solución exacta es $y(t) = e^{-2t}$. Integramos en el intervalo $[0, 1]$ con tamaño de paso $h = 0{,}25$, de modo que se realizan cuatro pasos numéricos.

En la Tabla 1 se muestran las aproximaciones de $y(1)$ obtenidas con cada método y el error absoluto comparado con la solución exacta $y(1) = e^{-2} \approx 0{,}135335$.

Método	Aproximación $y(1)$	Error absoluto
Euler	0,062500	$\approx 7{,}28 \times 10^{-2}$
Heun	0,152588	$\approx 1{,}73 \times 10^{-2}$
RK4	0,135550	$\approx 2{,}15 \times 10^{-4}$

Cuadro 1: Comparación numérica de Euler, Heun y RK4 para $y' = -2y$, $y(0) = 1$ en el intervalo $[0, 1]$ con $h = 0{,}25$.

Se observa que, con el mismo tamaño de paso, el método de Euler produce un error notable, el método de Heun mejora la aproximación, y RK4 obtiene un valor prácticamente coincidente con la solución exacta. En términos prácticos, esto significa que, para alcanzar una precisión similar a la de RK4, los métodos de orden inferior tendrían que emplear pasos mucho más pequeños, aumentando el número total de iteraciones y el costo de cómputo. Esto justifica la elección de RK4 como método principal en las simulaciones del presente trabajo.

5.4. Aplicación al modelo Depredador–Presa (Lotka–Volterra)

Consideremos el modelo de Lotka–Volterra:

$$\begin{cases} \dfrac{dP}{dt} = \alpha P - \beta PD, \\ \dfrac{dD}{dt} = \delta PD - \gamma D, \end{cases}$$

donde $P(t)$ es la población de presas, $D(t)$ la población de depredadores y $\alpha, \beta, \delta, \gamma > 0$ son parámetros del modelo.

Para un sistema como este, el método RK4 calcula las poblaciones en el siguiente instante de tiempo $t_{n+1} = t_n + h$ mediante

$$P_{t+h} = P_t + \frac{h}{6}(k_{1P} + 2k_{2P} + 2k_{3P} + k_{4P}),$$
$$D_{t+h} = D_t + \frac{h}{6}(k_{1D} + 2k_{2D} + 2k_{3D} + k_{4D}),$$

donde los k_{iP} y k_{iD} se obtienen evaluando las ecuaciones anteriores con los valores intermedios de P y D (según el esquema RK4).

Al aplicar el método RK4 al modelo Depredador–Presa, se observa que las poblaciones de presas y depredadores oscilan periódicamente. Cuando la población de presas aumenta, los depredadores disponen de más alimento y también crecen; sin embargo, al incrementarse los depredadores, la población de presas disminuye, lo que causa una posterior reducción en los depredadores. Estas oscilaciones se repiten en el tiempo, representando un equilibrio dinámico entre ambas especies.

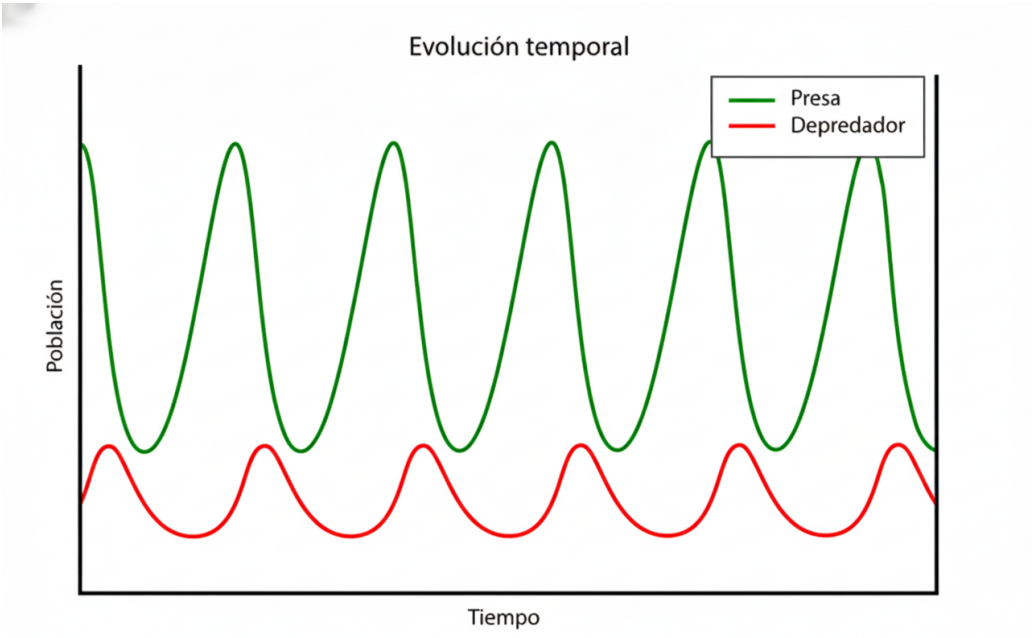


Figura 3: Evolución temporal de las poblaciones de presas (P) y depredadores (D), obtenida mediante el método de Runge–Kutta de 4° orden.

5.5. Ejemplo numérico del método RK4 aplicado al modelo Depredador–Presa (Lotka–Volterra)

Para ilustrar la aplicación del método de Runge–Kutta de cuarto orden (RK4) al modelo Depredador–Presa de Lotka–Volterra, consideremos nuevamente el sistema

$$\begin{cases} \dfrac{dP}{dt} = \alpha P - \beta PD, \\ \dfrac{dD}{dt} = \delta PD - \gamma D, \end{cases}$$

con los parámetros biológicos

$$\alpha = 1{,}00, \quad \beta = 0{,}05, \quad \delta = 0{,}02, \quad \gamma = 0{,}60,$$

y condiciones iniciales

$$P(0) = 80{,}00, \qquad D(0) = 20{,}00.$$

El punto de equilibrio no trivial del modelo viene dado por

$$(\bar{P}, \bar{D}) = \left(\frac{\gamma}{\delta}, \frac{\alpha}{\beta}\right) = \left(\frac{0{,}60}{0{,}02}, \frac{1{,}00}{0{,}05}\right) = (30{,}00, 20{,}00).$$

Obsérvese que la población inicial de depredadores coincide con su valor de equilibrio $D(0) = \bar{D} = 20{,}00$, mientras que la población inicial de presas es mucho mayor $P(0) = 80{,}00 > \bar{P}$, lo que representa un ecosistema con abundancia inicial de presas.

Para aproximar numéricamente la solución del sistema se utiliza el esquema RK4 descrito anteriormente, con paso temporal

$$h = 0{,}01$$

en el intervalo $t \in [0{,}00, 30{,}00]$. Esto implica realizar $N = 3000$ pasos de integración, lo que proporciona una aproximación suficientemente precisa a la dinámica del sistema.

5.5.1. Cálculo detallado de un paso de RK4

Tomando $P_0 = 80{,}00$, $D_0 = 20{,}00$ y $h = 0{,}01$, el procedimiento de RK4 para avanzar desde $t_0 = 0{,}00$ hasta $t_1 = t_0 + h = 0{,}01$ se organiza en los siguientes pasos:

1. **Cálculo de k_{1P} y k_{1D}.**

$$\begin{aligned} k_{1P} &= \alpha P_0 - \beta P_0 D_0 = 1{,}00 \cdot 80{,}00 - 0{,}05 \cdot 80{,}00 \cdot 20{,}00 \\ &= 80{,}00 - 80{,}00 = 0{,}00, \\ k_{1D} &= \delta P_0 D_0 - \gamma D_0 = 0{,}02 \cdot 80{,}00 \cdot 20{,}00 - 0{,}60 \cdot 20{,}00 \\ &= 32{,}00 - 12{,}00 = 20{,}00. \end{aligned}$$

   Con estos valores se obtiene el primer punto intermedio:

$$P^{(1)} = P_0 + \frac{h}{2}\, k_{1P} = 80{,}00, \qquad D^{(1)} = D_0 + \frac{h}{2}\, k_{1D} = 20{,}10.$$

2. **Cálculo de k_{2P} y k_{2D}.**

$$\begin{aligned} k_{2P} &= \alpha P^{(1)} - \beta P^{(1)} D^{(1)} \\ &= 1{,}00 \cdot 80{,}00 - 0{,}05 \cdot 80{,}00 \cdot 20{,}10 \\ &= 80{,}00 - 80{,}40 = -0{,}40, \\ k_{2D} &= \delta P^{(1)} D^{(1)} - \gamma D^{(1)} \\ &= 0{,}02 \cdot 80{,}00 \cdot 20{,}10 - 0{,}60 \cdot 20{,}10 \\ &= 32{,}16 - 12{,}06 = 20{,}10. \end{aligned}$$

   El segundo punto intermedio es entonces

$$P^{(2)} = P_0 + \frac{h}{2}\, k_{2P} = 79{,}998, \qquad D^{(2)} = D_0 + \frac{h}{2}\, k_{2D} = 20{,}1005.$$

3. **Cálculo de k_{3P} y k_{3D}.**

$$\begin{aligned} k_{3P} &= \alpha P^{(2)} - \beta P^{(2)} D^{(2)} \approx -0{,}4020, \\ k_{3D} &= \delta P^{(2)} D^{(2)} - \gamma D^{(2)} \approx 20{,}10. \end{aligned}$$

   De aquí se obtiene el tercer punto intermedio:

$$P^{(3)} = P_0 + h\, k_{3P} \approx 79{,}9960, \qquad D^{(3)} = D_0 + h\, k_{3D} \approx 20{,}2010.$$

4. **Cálculo de k_{4P} y k_{4D}.**

$$\begin{aligned} k_{4P} &= \alpha P^{(3)} - \beta P^{(3)} D^{(3)} \approx -0{,}8039, \\ k_{4D} &= \delta P^{(3)} D^{(3)} - \gamma D^{(3)} \approx 20{,}20. \end{aligned}$$

Finalmente, se aplican las fórmulas de actualización de RK4:

$$P_{t+h} = P_t + \frac{h}{6}\Big(k_{1P} + 2k_{2P} + 2k_{3P} + k_{4P}\Big),$$
$$D_{t+h} = D_t + \frac{h}{6}\Big(k_{1D} + 2k_{2D} + 2k_{3D} + k_{4D}\Big),$$

de donde, para $t_0 = 0{,}00$ y $h = 0{,}01$,

$$P(0{,}01) \approx 79{,}996, \qquad D(0{,}01) \approx 20{,}201.$$

Este cálculo ilustra cómo el método RK4 combina cuatro evaluaciones de las ecuaciones diferenciales para producir una aproximación de alta precisión al estado del sistema en el instante siguiente.

5.5.2. Tabla de valores numéricos.

Repitiendo el procedimiento anterior para todo el intervalo $t \in [0{,}00, 30{,}00]$ con paso $h = 0{,}01$, se obtiene la evolución aproximada de las poblaciones de presas $P(t)$ y depredadores $D(t)$. En la Tabla 2 se muestran los valores de $P(t)$ y $D(t)$ en tiempos enteros, redondeados a dos decimales:

t	$P(t)$	$D(t)$
0.00	80.00	20.00
1.00	44.92	41.89
2.00	14.51	39.14
3.00	7.70	26.33
4.00	7.27	16.69
5.00	10.06	10.83
6.00	17.37	7.74
7.00	33.05	6.91
8.00	60.91	9.52
9.00	79.29	22.79
10.00	38.74	43.21
11.00	12.87	37.50
12.00	7.42	24.83
13.00	7.45	15.73
14.00	10.71	10.29
15.00	18.83	7.50

t	$P(t)$	$D(t)$
16.00	36.00	6.99
17.00	65.06	10.38
18.00	77.08	25.86
19.00	33.19	43.88
20.00	11.54	35.79
21.00	7.23	23.40
22.00	7.68	14.83
23.00	11.43	9.79
24.00	20.45	7.30
25.00	39.19	7.13
26.00	69.06	11.44
27.00	73.39	29.13
28.00	28.36	43.96
29.00	10.47	34.05
30.00	7.10	22.04

Cuadro 2: Valores aproximados de las poblaciones de presas $P(t)$ y depredadores $D(t)$ obtenidos mediante RK4 con paso $h = 0{,}01$, para el intervalo $t \in [0{,}00, 30{,}00]$.

5.6. Herramientas y Tecnologías

- **Python:** Lenguaje principal para la lógica del sistema.
- **Manim:** Utilizado para animaciones y visualización de procesos.

6. Desarrollo del Proyecto

6.1. Núcleo del simulador: `backend/simulation.py`

El archivo `backend/simulation.py` es el corazón matemático del proyecto. Aquí es donde las ecuaciones del modelo Lotka-Volterra (presentadas en la Sección 4) se traducen en código Python ejecutable usando el método RK4.

Función 1: Campo vectorial del modelo

¿Para qué sirve? Esta función calcula las derivadas (tasas de cambio) de las poblaciones en un instante dado. Es la implementación directa de las ecuaciones (4.2).

```
import numpy as np

def lotka_volterra_rhs(P, D, a, b, d, g):
    """Calcula las derivadas del sistema Lotka-Volterra."""
    dP = a * P - b * P * D      # dx/dt (Ecuacion de presas)
    dD = d * P * D - g * D      # dy/dt (Ecuacion de depredadores
   )
    return dP, dD
```

Listing 1: Campo vectorial — `backend/simulation.py`

Lo importante: Esta función representa matemáticamente cómo cambian las poblaciones. El término αP es el crecimiento de presas, $-\beta PD$ es la depredación, y así sucesivamente según el modelo teórico.

Función 2: Paso RK4

¿Para qué sirve? Implementa el método de Runge-Kutta de 4to orden que estudiamos en clase. Es el algoritmo que resuelve numéricamente las ecuaciones diferenciales.

Lo importante:

- Las 4 evaluaciones (k_1, k_2, k_3, k_4) son las pendientes que usa RK4 para calcular el siguiente punto.
- El promedio ponderado (1:2:2:1) es la fórmula característica de RK4.
- `max(P_new, 0)` garantiza que las poblaciones nunca sean negativas (restricción biológica).

```python
def rk4_step(P, D, h, a, b, d, g):
    """Avanza un paso temporal usando RK4."""
    # Calcular las 4 pendientes (k1, k2, k3, k4)
    k1P, k1D = lotka_volterra_rhs(P, D, a, b, d, g)
    k2P, k2D = lotka_volterra_rhs(P + 0.5*h*k1P, D + 0.5*h*k1D, a, b, d, g)
    k3P, k3D = lotka_volterra_rhs(P + 0.5*h*k2P, D + 0.5*h*k2D, a, b, d, g)
    k4P, k4D = lotka_volterra_rhs(P + h*k3P, D + h*k3D, a, b, d, g)

    # Promedio ponderado (1:2:2:1)
    P_new = P + (h/6)*(k1P + 2*k2P + 2*k3P + k4P)
    D_new = D + (h/6)*(k1D + 2*k2D + 2*k3D + k4D)

    # Evitar valores negativos por errores de redondeo
    return max(P_new, 0), max(D_new, 0)
```

Listing 2: Paso de integración RK4 — `backend/simulation.py`

Función 3: Simulador completo

¿Para qué sirve? Ejecuta la simulación completa desde el tiempo inicial hasta el tiempo final, repitiendo el paso RK4 muchas veces.

```python
def simulate_lotka_volterra(alpha, beta, delta, gamma,
                            P0, D0, t_max, dt=0.05):
    """Simula el sistema completo a lo largo del tiempo."""
    # Calcular cuantos pasos necesitamos
    n = int(t_max / dt) + 1
    t = np.linspace(0, t_max, n)

    # Preparar arrays para guardar los resultados
    P = np.zeros(n)  # Poblacion de presas
    D = np.zeros(n)  # Poblacion de depredadores
    P[0], D[0] = P0, D0  # Condiciones iniciales

    # Iterar en el tiempo usando RK4
    Pi, Di = P0, D0
    for k in range(1, n):
        Pi, Di = rk4_step(Pi, Di, dt, alpha, beta, delta, gamma)
        P[k], D[k] = Pi, Di

    return {"t": t, "P": P, "D": D}
```

Listing 3: Simulador completo — `backend/simulation.py`

Lo importante:

- Con $dt = 0{,}05$ y $t_{max} = 50$, se generan aproximadamente 1000 puntos.

- `np.zeros(n)` crea un array vacío que luego llenamos con los resultados.
- El diccionario retornado contiene tres arrays: tiempos, presas y depredadores.

6.2. Sistema de validación: `backend/validators.py`

¿Para qué sirve? Antes de ejecutar cualquier simulación, este código verifica que los parámetros ingresados por el usuario sean válidos. Esto previene errores matemáticos y bloqueos del sistema.

Validaciones implementadas:

1. **Evitar valores vacíos**: Verifica que todos los parámetros hayan sido ingresados.
2. **Evitar valores negativos**: Las poblaciones y tasas negativas no tienen sentido biológico (un error común del usuario).
3. **Evitar división por cero**: Si $\beta = 0$ o $\delta = 0$, al calcular el punto de equilibrio $(\gamma/\delta, \alpha/\beta)$ habría división por cero.
4. **Evitar tiempos excesivos**: Limita el tiempo máximo a 300 unidades para evitar que el navegador se bloquee por arrays muy grandes.

Ejemplo de uso:

```
def validate_inputs(a, b, d, g, P0, D0, tmax):
    """Valida que los parametros sean correctos."""
    # Verificar que no sean negativos
    if any(x < 0 for x in [a, b, d, g, P0, D0, tmax]):
        return False, "ERROR: No se aceptan valores negativos"

    # Evitar division por cero
    if b == 0 or d == 0:
        return False, "ERROR: Beta y Delta no pueden ser 0"

    # Todo OK
    return True, ""
```

Listing 4: Ejemplo de validación — `backend/validators.py`

Esta validación se ejecuta tanto en la interfaz web (antes de graficar) como en el backend (antes de generar videos), garantizando datos correctos en todo el sistema.

6.3. Librerías utilizadas

El proyecto utiliza librerías científicas de Python para implementar el simulador. Las principales son:

Librerías núcleo del proyecto

1. **NumPy** ($\geq$ 2.3.5)
   - **Para qué sirve:** Manejo eficiente de arrays numéricos y operaciones matemáticas.
   - **Dónde se usa:** En todo `simulation.py` para crear y manipular los arrays de tiempo y poblaciones.
   - **Ejemplo:** `np.zeros(n)`, `np.linspace(0, t_max, n)`
2. **Plotly** ($\geq$ 6.5.0)
   - **Para qué sirve:** Generación de gráficos interactivos en la web.
   - **Dónde se usa:** En `pages/simulador.py` para crear los 5 tipos de gráficos (serie temporal, diagrama de fases, etc.).
   - **Ventaja:** Permite hacer zoom, paneo y hover sin recargar la página.
3. **Manim** ($\geq$ 0.19.0)
   - **Para qué sirve:** Creación de animaciones matemáticas de alta calidad (videos MP4).
   - **Dónde se usa:** En `backend/scenes/video3.py` para generar los videos explicativos.
   - **Característica:** Genera videos en 1080p a 60 FPS con ecuaciones LaTeX integradas.
4. **Dash** ($\geq$ 3.3.0)
   - **Para qué sirve:** Framework para crear la interfaz web completa usando solo Python.
   - **Dónde se usa:** En `app.py` y `pages/*.py` para construir la página web interactiva.
   - **Ventaja:** No necesitas HTML/CSS/JavaScript, todo se programa en Python.
5. **FastAPI** ($\geq$ 0.115.0)
   - **Para qué sirve:** Crear una API REST para procesar solicitudes de generación de video.
   - **Dónde se usa:** En `backend/app.py` como servidor independiente (puerto 8000).

- **Función:** Recibe parámetros del usuario, ejecuta Manim y retorna el video generado.

6. **SciPy** ($\geq$ 1.16.3)
   - **Para qué sirve:** Biblioteca de cálculo científico avanzado.
   - **Nota:** Aunque está instalada, implementamos RK4 manualmente con fines pedagógicos.

6.4. Arquitectura modular del backend

El proyecto organiza el código en archivos independientes dentro de `backend/`:

- `simulation.py`: Motor numérico (RK4 + Lotka-Volterra)
- `validators.py`: Verificación de datos de entrada
- `app.py`: Servidor API para videos
- `video_tools.py`: Gestión de renderizado con Manim

Ventaja de esta organización: El mismo código de `simulation.py` se reutiliza tanto para los gráficos en tiempo real (usando Plotly) como para los videos (usando Manim), garantizando que los resultados sean idénticos.

6.5. Flujo de ejecución del simulador

Cuando el usuario interactúa con el simulador web, el flujo es el siguiente:

1. Usuario ingresa parámetros ($\alpha, \beta, \delta, \gamma, P_0, D_0, t_{max}$) en la interfaz web.
2. `validators.py` verifica que los datos sean válidos.
3. `simulation.py` ejecuta la simulación usando RK4.
4. `Plotly` genera los gráficos interactivos con los resultados.
5. Si el usuario solicita un video, `FastAPI` coordina la generación con Manim.

Este diseño modular permite separar la lógica matemática (simulación) de la presentación (gráficos/videos), facilitando el mantenimiento y las pruebas del código.

7. Resultados y análisis

7.1. Simulación base

En esta sección se presentan los resultados de la simulación base del modelo Depredador–Presa de Lotka–Volterra, utilizando el método de Runge–Kutta de cuarto orden (RK4) con los parámetros:

$$\alpha = 1{,}00, \quad \beta = 0{,}05, \quad \delta = 0{,}02, \quad \gamma = 0{,}60,$$

condiciones iniciales

$$P(0) = 80{,}00, \qquad D(0) = 20{,}00,$$

e intervalo temporal $t \in [0, 30]$ con paso $h = 0{,}01$.

Gráfico 1A y 1B: casos límite sin interacción

En primer lugar se analizan por separado las dos ecuaciones del modelo, para entender el efecto de cada término sin la interacción presa–depredador.

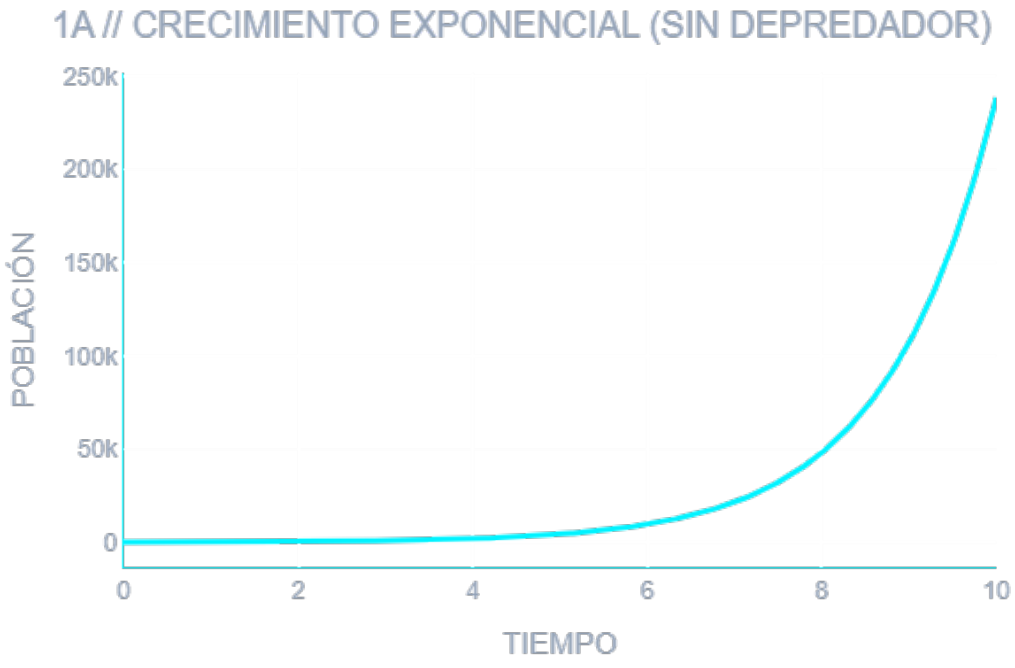


Figura 4: Crecimiento exponencial (sin depredador). Evolución de la población de presas cuando se anula la interacción ($\beta = 0$).

En la Figura 4 se observa que, en ausencia de depredadores, la ecuación se reduce a $\frac{dP}{dt} = \alpha P$. La solución es un crecimiento exponencial $P(t) = P(0)e^{\alpha t}$: partiendo de

$P(0) = 80$, la población aumenta de forma muy rápida y alcanza valores del orden de millones en poco tiempo. Esto representa un escenario idealizado en el que la especie presa no tiene limitaciones externas y crece sin control.

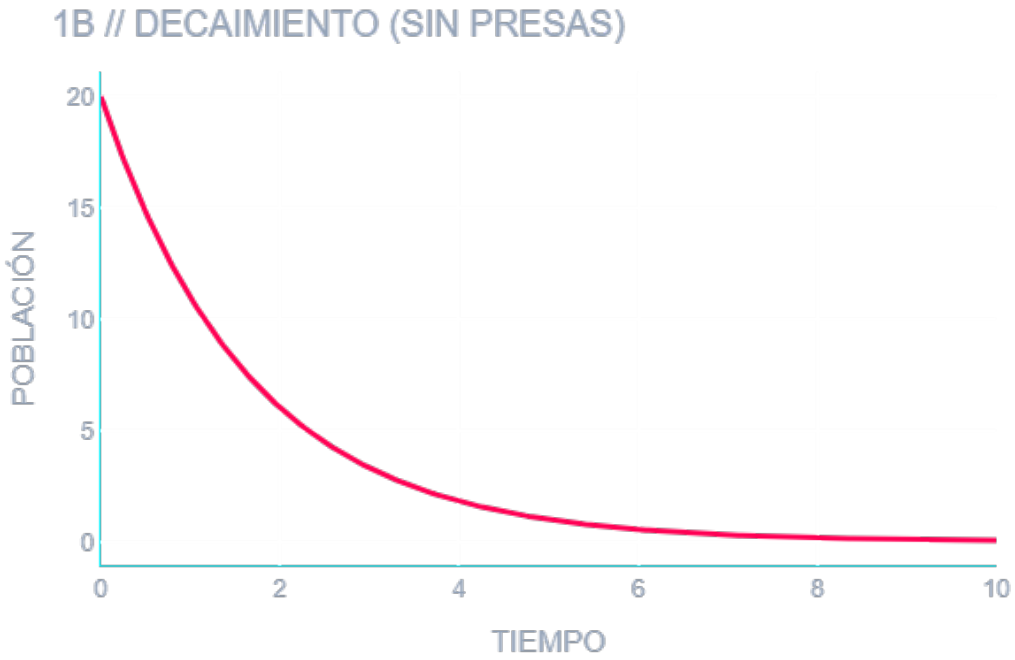


Figura 5: Decaimiento (sin presas). Evolución de la población de depredadores cuando no hay presas disponibles.

En la Figura 5 se muestra el caso contrario: si no hay presas, el modelo queda como $\frac{dD}{dt} = -\gamma D$, cuya solución es $D(t) = D(0)e^{-\gamma t}$. La población de depredadores decrece de forma exponencial desde $D(0) = 20$ hasta valores prácticamente nulos alrededor de $t = 10$. Biológicamente, esto indica que la especie depredadora no puede mantenerse sin alimento.

Estas dos simulaciones extremas justifican la necesidad de estudiar el sistema acoplado: las presas no pueden crecer indefinidamente porque son consumidas, y los depredadores no pueden sobrevivir si no existen presas.

Gráfico 2: dinámica temporal presa–depredador

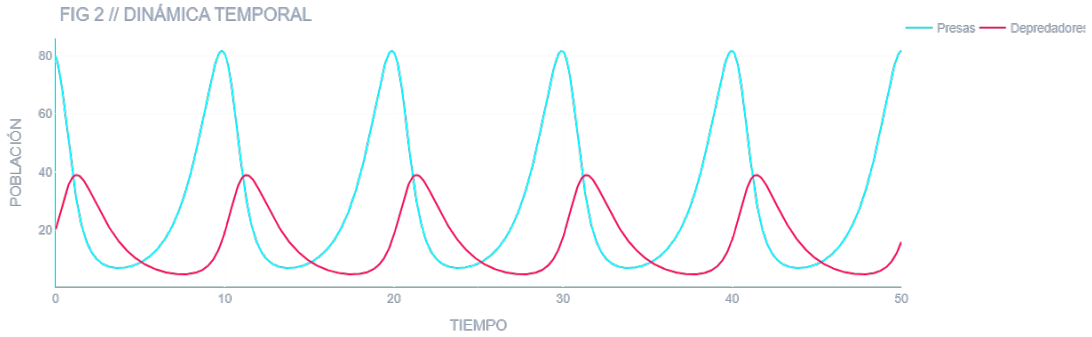


Figura 6: Dinámica temporal. Evolución simultánea de las poblaciones de presas y depredadores en el intervalo $t \in [0, 30]$.

La Figura 6 muestra la evolución temporal conjunta de $P(t)$ (presas) y $D(t)$ (depredadores) para el conjunto de parámetros base. Ambos muestran un comportamiento claramente oscilatorio:

- La población de presas presenta picos agudos (valores cercanos a 80) seguidos de descensos pronunciados hasta valores mínimos alrededor de 7–8 individuos.
- La población de depredadores alcanza máximos en torno a 40–45 individuos, pero estos picos aparecen retrasados en el tiempo respecto a los de las presas.

Se observa un **desfase temporal**: primero aumenta la población de presas; después, cuando el alimento es abundante, crece la población de depredadores; el aumento de depredadores reduce el número de presas; cuando las presas escasean, los depredadores comienzan a morir y su población disminuye, lo que permite que las presas se recuperen. Este ciclo de crecimiento y caída se repite, generando oscilaciones periódicas.

Los valores de la tabla numérica calculada con RK4 muestran que ambas poblaciones oscilan alrededor del punto de equilibrio teórico del modelo:

$$(\bar{P}, \bar{D}) = \left(\frac{\gamma}{\delta}, \frac{\alpha}{\beta}\right) = (30{,}00, 20{,}00).$$

Gráfico 3: plano de fases

Figura 7: Plano de fases (con equilibrio). Trayectoria del sistema en el plano (P, D), mostrando el estado inicial y el punto de equilibrio.

La Figura 7 representa la trayectoria del sistema en el plano de fases (P, D). La curva forma una órbita cerrada alrededor del punto de equilibrio $(30{,}00, 20{,}00)$. El punto inicial $(80{,}00, 20{,}00)$ se muestra sobre la parte derecha de la órbita y el equilibrio se marca como un punto fijo en el interior.

Este tipo de trayectoria es típico del modelo clásico de Lotka–Volterra: el equilibrio interno se comporta como un *centro*, de modo que las soluciones no convergen ni divergen, sino que describen ciclos cerrados. El método RK4, con un paso pequeño $h = 0{,}01$, preserva bien esta estructura orbital sin introducir inestabilidades numéricas apreciables.

7.2. Experimentación y análisis del sistema

A continuación se estudia cómo cambia la dinámica al modificar algunos parámetros biológicos. En la aplicación se consideraron variaciones de α (crecimiento de presas) y γ (mortalidad de depredadores), mostrando en una misma figura la órbita base y las órbitas modificadas.

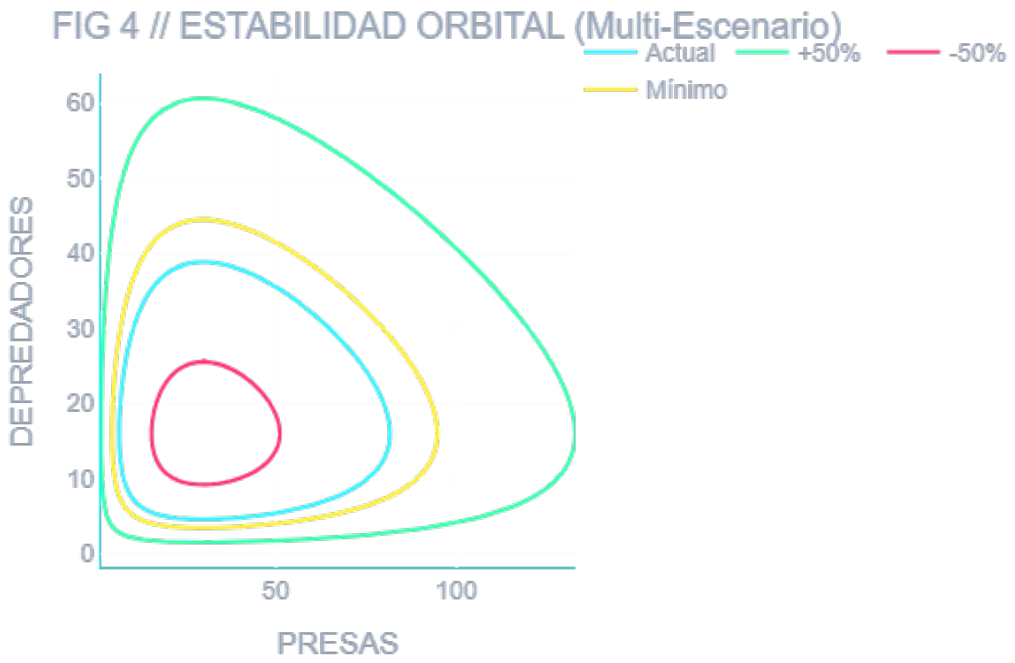


Figura 8: Estabilidad orbital (multi–escenario). Comparación entre la órbita base (*Actual*) y las órbitas obtenidas al modificar los parámetros del modelo ($+50\,\%$, $-50\,\%$ y caso *Mínimo*).

En la Figura 8 se muestran varias órbitas en el plano de fases:

- La curva etiquetada como **Actual** corresponde a la simulación base con los parámetros originales.
- La curva **+50 %** representa un escenario donde se incrementa la capacidad de crecimiento de las presas (o se reduce parcialmente la mortalidad de los depredadores). La órbita resultante es más grande: las poblaciones alcanzan valores máximos mayores y los ciclos son más amplios.
- La curva **-50 %** corresponde a un escenario más restrictivo (por ejemplo, menor crecimiento de presas o mayor mortalidad de depredadores). En este caso la órbita es más pequeña, indicando ciclos de menor amplitud y poblaciones que se mantienen más cercanas al equilibrio.
- La órbita marcada como **Mínimo** ilustra un caso extremo, donde los parámetros favorecen poco la supervivencia de ambas especies: las trayectorias se comprimen y las poblaciones oscilan en un rango reducido.

Aunque las amplitudes de los ciclos cambian y las órbitas se desplazan en el plano (P, D), todas permanecen cerradas alrededor de un punto de equilibrio. Esto muestra una

estabilidad orbital: el sistema sigue presentando ciclos presa–depredador frente a variaciones moderadas en los parámetros, aunque la intensidad y el rango de las oscilaciones sí dependen de dichos parámetros.

En conjunto, los resultados confirman que el modelo de Lotka–Volterra captura adecuadamente la dinámica cualitativa de un sistema Depredador–Presa: crecimiento descontrolado en ausencia de depredación, extinción en ausencia de presas, oscilaciones periódicas cuando ambas especies interactúan y sensibilidad de las órbitas ante cambios en las tasas de crecimiento y mortalidad.

8. Conclusiones

En este proyecto se modeló la interacción entre una población de presas y una población de depredadores mediante el sistema de Lotka–Volterra y se resolvió numéricamente usando el método de Runge–Kutta de cuarto orden (RK4). A partir de las simulaciones y de la experimentación con distintos parámetros se pueden extraer las siguientes conclusiones:

1. **El modelo reproduce bien la intuición biológica básica.**
   Cuando se eliminan los depredadores, la población de presas crece de forma casi explosiva; cuando se eliminan las presas, los depredadores desaparecen por falta de alimento. Cuando ambas especies interactúan, ninguna de las dos se mantiene constante, sino que aparecen ciclos de crecimiento y caída que encajan con la idea de "muchas presas ⇒ más depredadores ⇒ menos presas ⇒ menos depredadores ⇒ vuelven a subir las presas".

2. **Las oscilaciones presa–depredador son una consecuencia natural del modelo.**
   Las simulaciones muestran claramente el desfase: primero suben las presas, luego (con cierto retraso) suben los depredadores, y así sucesivamente. En el plano de fases esto se ve como órbitas cerradas alrededor del punto de equilibrio. No hace falta "forzar" estos ciclos: aparecen solos a partir de las ecuaciones, lo que le da bastante fuerza al modelo como herramienta cualitativa.

3. **RK4 es una buena elección como método numérico.**
   Con un paso relativamente pequeño ($h = 0{,}01$), RK4 entrega soluciones estables y suaves, sin explosiones numéricas raras ni ruido. Esto permite centrarse en interpretar la dinámica del sistema y no en pelear con errores de aproximación. En términos simples: el método hace bien su trabajo y no se "mete" con la física del problema.

4. **El sistema es sensible a los parámetros, pero mantiene su comportamiento global.**
   Al modificar la tasa de crecimiento de las presas α o la mortalidad de los depredadores γ, cambian la amplitud y la posición de las órbitas en el plano de fases: los máximos y mínimos de las poblaciones pueden ser mucho más extremos o más suaves. Sin embargo, la estructura general se conserva: siguen apareciendo ciclos cerrados y la relación presa–depredador se mantiene. Esto muestra una cierta estabilidad cualitativa del modelo.

5. **La aplicación desarrollada facilita entender el modelo.**
   Poder mover deslizadores, cambiar parámetros y ver al instante las curvas de tiempo y los diagramas de fase hace que el modelo deje de ser solo fórmulas y se convierta en algo visual e intuitivo. La aplicación permite experimentar rápidamente con distintos escenarios y ver cómo reacciona el sistema, algo muy difícil de lograr solo con cálculo a mano.

6. **Limitaciones y posibles mejoras.**
   El modelo de Lotka–Volterra que se usó es idealizado: no incluye capacidad de carga del ambiente, ni efectos de saturación, ni cambios estacionales, ni comportamiento más complejo de las especies. Por eso, aunque captura bien la idea general de la interacción depredador–presa, no se debe interpretar como una predicción exacta de poblaciones reales. Un trabajo futuro natural sería extender la aplicación y el modelo para incluir términos más realistas (logísticos, estocásticos, más especies, etc.).

9. Referencias bibliográficas

- Burden, R. L., & Faires, J. D. (2011). *Análisis Numérico*. Cengage Learning.
- Chapra, S. C., & Canale, R. P. (2015). *Métodos Numéricos para Ingenieros*. McGraw-Hill.
- Kingsland, S. E. (2015). *Alfred J. Lotka and the origins of theoretical population ecology*. Proceedings of the National Academy of Sciences, 112(30), 9493–9495.https://pmc.ncbi.nlm.nih.gov/articles/PMC4534218
- Kiusalaas, J. (2013). *Numerical Methods in Engineering with Python 3*. Cambridge University Press.
- Forrest, S. (s. f.).*Predator–Prey Models*. Dept. of Computer Science, University of New Mexico. Recuperado de https://www.cs.unm.edu/~forrest/classes/cs365/lectures/Lotka-Volterra.pdf
- Parker, A. E. (2021).*Runge–Kutta 4 (and other numerical methods for ODE's)* Ursinus College Digital Commons. Recuperado de https://digitalcommons.ursinus.edu/cgi/viewcontent.cgi?article=1007&context=triumphs

10. Anexos

Anexo A. Implementación genérica del método RK4 en Python

En este anexo se muestra una implementación genérica del método de Runge–Kutta de cuarto orden (RK4) para un problema de valor inicial

$$y'(t) = f(t, y), \qquad y(t_0) = y_0,$$

tal como se describe en la Sección 4 del presente informe. Esta versión opera sobre una función escalar f, pero la idea se extiende de forma directa al caso vectorial empleado en el modelo Depredador–Presa.

```python
import numpy as np

def runge_kutta4(f, y0, t0, tf, h):
    """
    Aplica el metodo de Runge-Kutta de 4to orden
    al PVI y'(t) = f(t,y), y(t0) = y0, en el
    intervalo [t0, tf] con paso h.
    """
    # Mallado temporal
    t = np.arange(t0, tf + h, h)
    y = np.zeros(len(t))
    y[0] = y0

    # Bucle principal de integracion
    for i in range(len(t) - 1):
        k1 = f(t[i],          y[i])
        k2 = f(t[i] + h/2.0, y[i] + h*k1/2.0)
        k3 = f(t[i] + h/2.0, y[i] + h*k2/2.0)
        k4 = f(t[i] + h,     y[i] + h*k3)
        y[i+1] = y[i] + (h/6.0)*(k1 + 2*k2 + 2*k3 + k4)

    return t, y
```

Listing 5: Implementación genérica del método RK4 en Python

Esta función sirve como base conceptual para la especialización utilizada en el archivo `backend/simulation.py`, donde se implementa el esquema RK4 adaptado al sistema de Lotka–Volterra.

Anexo B. Estructura de archivos del proyecto de simulación

A continuación se presenta un esquema simplificado de la estructura de directorios del repositorio del proyecto, donde se aprecia la separación entre núcleo numérico, API y componentes de interfaz web:

```
Project-Numerical-Analysis-Lotka-Volterra/
 app.py                      # Punto de entrada de la app Dash
 requirements.txt            # Dependencias del proyecto
 backend/
    simulation.py           # Motor numérico (RK4 + Lotka-Volterra)
    validators.py           # Validación de parámetros de entrada
    app.py                  # API FastAPI para generación de videos
    video_tools.py          # Utilidades para renderizar con Manim
    scenes/
        video3.py           # Escena principal de animación en Manim
 pages/
    simulador.py            # Página principal del simulador web (Dash)
 assets/                     # Estilos, fuentes y recursos gráficos
 media/                      # Carpeta para guardar videos generados
 docs/
     informe.tex             # Informe LaTeX del proyecto
     beamer.tex              # Presentación en Beamer
```

Esta organización modular permite reutilizar el mismo núcleo numérico (`simulation.py`) tanto para los gráficos interactivos como para los videos generados con Manim, facilitando el mantenimiento y la extensibilidad del sistema.

Anexo C. Guía rápida para ejecutar el simulador web

Finalmente, se incluye una guía resumida para instalar y ejecutar el simulador desde el repositorio del proyecto, pensada como referencia rápida para el docente o cualquier usuario que desee reproducir las simulaciones.

1. **Clonar el repositorio**

   - `git clone https://github.com/floowxy/Project-Numerical-Analysis-Lotka-Volterra.git`
   - `cd Project-Numerical-Analysis-Lotka-Volterra`

2. **Crear y activar un entorno virtual de Python** (versión recomendada: 3.12 o 3.13).

```
- python -m venv .venv
# En Linux/macOS
- source .venv/bin/activate
# En Windows
- .venv\Scripts\activate
```

3. **Instalar las dependencias**
   - pip install -r requirements.txt

4. **Ejecutar la aplicación web**
   - python app.py
   - Luego, abrir en el navegador la dirección que indique la consola (normalmente http://127.0.0.1:8050) para interactuar con el simulador.

5. **Generar un video con Manim (opcional)**
   Para producir videos animados con las órbitas del sistema, se levanta el servidor FastAPI del backend y se solicita la generación desde la propia interfaz del simulador. El video se guarda en la carpeta `media/`.