Simulación de un Ecosistema Artificial: Un Modelo Computacional de la Dinámica Depredador - Presa

Lotka-Volterra y Runge-Kutta 4

Diego Sotelo
Alexis Gonzales
Paolo Villavicencio
Alvaro Salazar
Facultad de Ingeniería de Sistemas e Informática

Overleaf FISI ▪ flowxy.org

December 2, 2025

Tabla de contenidos

Contexto del problema

- En ecología matemática es clásico estudiar sistemas **depredador–presa**, donde ambas poblaciones exhiben **oscilaciones cíclicas** en el tiempo.
- Cuando la población de presas aumenta, los depredadores disponen de más alimento y su población crece.
- Luego, la mayor presión de depredación reduce las presas, lo que provoca una disminución posterior de depredadores.
- Este comportamiento cíclico ha sido uno de los motores históricos del desarrollo de la **ecología teórica**.

Planteamiento del proyecto

- Se modela la interacción depredador–presa mediante el sistema de **Lotka–Volterra**.
- El sistema no tiene, en general, una solución analítica cerrada sencilla para tiempos largos.
- **Idea del proyecto:**
  - Resolver el sistema numéricamente.
  - Construir un **simulador computacional** que permita explorar la dinámica del sistema.
- Para ello se elige el método **Runge–Kutta de cuarto orden (RK4)**, por su buen equilibrio entre precisión y costo.

Objetivo general y específicos

Objetivo general

- Implementar un programa que simule la evolución de las poblaciones depredador–presa en el tiempo y visualice los resultados mediante gráficas y animaciones.

Objetivos específicos

- Analizar el modelo matemático de Lotka–Volterra.
- Desarrollar una implementación propia del método RK4 para el sistema de EDOs.
- Codificar la solución numérica en Python.
- Diseñar visualizaciones: población vs. tiempo y diagrama de fase.
- Experimentar con distintos parámetros y analizar la estabilidad del sistema.

Modelo matemático depredador–presa

El modelo clásico de Lotka–Volterra:

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

donde:

- $P(t)$: población de **presas**.
- $D(t)$: población de **depredadores**.
- α: tasa de crecimiento natural de presas.
- β: tasa de encuentros presa–depredador (muerte de presas).
- δ: eficacia reproductiva de depredadores por consumo de presas.
- γ: mortalidad natural de depredadores.

Puntos de equilibrio

Se imponen condiciones de equilibrio:

$$\frac{dP}{dt} = 0, \qquad \frac{dD}{dt} = 0.$$

Se obtienen dos puntos de equilibrio:

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

- $(0, 0)$: extinción de ambas poblaciones.
- $(\bar{P}, \bar{D})$: equilibrio no trivial con coexistencia.
- Alrededor de $(\bar{P}, \bar{D})$ aparecen **órbitas cerradas** en el plano de fases (ciclos poblacionales).

Interpretación cualitativa

- Si no hay depredadores ($D = 0$):

$$\frac{dP}{dt} = \alpha P \quad \Rightarrow \quad P(t) = P(0)\,\mathrm{e}^{\alpha t} \quad \text{(crecimiento exponencial)}.$$

- Si no hay presas ($P = 0$):

$$\frac{dD}{dt} = -\gamma D \quad \Rightarrow \quad D(t) = D(0)\,\mathrm{e}^{-\gamma t} \quad \text{(decaimiento exponencial)}.$$

- Con ambas especies presentes y parámetros positivos, el sistema genera **órbitas cerradas** alrededor del equilibrio no trivial.
- Esto se interpreta como **ciclos presa–depredador** en el plano (P, D).

PVI y métodos de un paso

Consideramos el problema de valor inicial (PVI):

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

- Dividimos el intervalo en puntos $t_k = t_0 + kh$.
- Un **método de un paso** genera aproximaciones y_k mediante:

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

- Ejemplos:
  - Euler explícito.
  - Euler mejorado (Heun, RK2).
  - Métodos de Runge–Kutta (RK4, etc.).

Idea general de Runge–Kutta

- Los métodos de Runge–Kutta buscan la precisión de un método de Taylor de orden alto, pero **sin** calcular derivadas de orden superior.
- En vez de derivadas, se evalúa $f(t, y)$ varias veces dentro del intervalo $[t_k, t_{k+1}]$.
- Se combinan esas pendientes para aproximar $y(t_{k+1})$.
- El método clásico de orden 4 (RK4):
  - tiene buena precisión y estabilidad,
  - es fácil de programar,
  - es suficiente para la mayoría de problemas prácticos.

RK4: fórmula clásica con k_1, k_2, k_3, k_4

Para un PVI $y' = f(t, y)$:

$$
\begin{aligned}
k_1 &= f(t_k, y_k), \\
k_2 &= f\left(t_k + \frac{h}{2},\, y_k + \frac{h}{2}k_1\right), \\
k_3 &= f\left(t_k + \frac{h}{2},\, y_k + \frac{h}{2}k_2\right), \\
k_4 &= f(t_k + h,\, y_k + hk_3),
\end{aligned}
$$

y la actualización es

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

Es un método de **orden 4**: el error global es $O(h^4)$.

Ejemplo de aplicación del método RK4

Problema de valor inicial

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

Usamos $h = 0.25$ y calculamos el primer paso de $t_0 = 0$ a $t_1 = 0.25$.

Pendientes k_1 y k_2

$$k_1 = f(t_0, y_0) = \frac{0-1}{2} = -0.5,$$

$$k_2 = f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1\right)$$
$$= f(0.125, 0.9375) = \frac{0.125 - 0.9375}{2} = -0.40625.$$

Pendientes k_3 y k_4

$$k_3 = f\left(t_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_2\right)$$
$$= f(0.125, 0.94921875) \approx -0.4121094,$$

$$k_4 = f(t_0 + h, y_0 + h\,k_3)$$
$$= f(0.25, 0.89697) \approx -0.3234863.$$

Actualización

$$y_1 = y_0 + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$
$$\approx 0.89749.$$

Por tanto,

$$y(0.25) \approx 0.89749.$$

Orden y error de RK4

Suponiendo que $y(t)$ es suficientemente suave:

- **Error local** de truncamiento: $O(h^5)$.
- **Error global** en todo el intervalo: $O(h^4)$.
- Si se reduce h a la mitad, el error global baja aproximadamente por un factor $2^4 = 16$ (mientras no domine el error de redondeo).

Interpretación vía integral y regla de Simpson

Recordemos que, en un subintervalo $[t_0, t_1]$,

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

- RK4 puede interpretarse como una aproximación de esa integral usando la **regla de Simpson** con paso $h/2$.
- Los valores f_1, f_2, f_3, f_4 juegan el papel de pendientes en:
  - inicio del intervalo (f_1),
  - punto medio (f_2 y f_3),
  - final del intervalo (f_4).
- La combinación ponderada $\frac{1}{6}(f_1 + 2f_2 + 2f_3 + f_4)$ aproxima el promedio de f en el subintervalo.

¿Por qué elegimos el método RK4?

Idea general

- Para resolver numéricamente EDOs podemos usar métodos de un paso: Euler, Euler mejorado (RK2), Runge–Kutta de 4to orden (RK4), etc.
- Todos aproximan la solución real, pero difieren en: **precisión**, **estabilidad** y **costo** por paso.

Orden de los métodos (error global)

Método	Evaluaciones de f	Orden global
Euler explícito	1	$\mathcal{O}(h)$
Euler mejorado / RK2	2	$\mathcal{O}(h^2)$
RK4 clásico	4	$\mathcal{O}(h^4)$

Ventajas de RK4 para este proyecto

- Mucho más preciso que Euler y RK2 para un mismo tamaño de paso h.
- Permite usar un h moderado (`h = 0.05`) sin que la solución "explote" ni se vuelva inestable.
- Las trayectorias (P(t),D(t)) salen suaves y coherentes, algo clave para las gráficas y el video.
- Es un buen compromiso entre costo (4 evaluaciones de f por paso) y calidad de la aproximación.

Comparación numérica: Euler vs Heun vs RK4

Problema de valor inicial

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

Integramos en el intervalo $[0,1]$ con tamaño de paso $h = 0{,}25$ (4 pasos). La solución exacta es

$$y(t) = t - 2 + 3e^{-t/2} \quad \Rightarrow \quad y(1) \approx 0{,}819592.$$

Método	Aproximación $y(1)$	Error absoluto
Euler	$0{,}758545$	$\approx 6{,}10 \times 10^{-2}$
Heun	$0{,}822196$	$\approx 2{,}60 \times 10^{-3}$
RK4	$0{,}819594$	$\approx 2{,}05 \times 10^{-6}$

Conclusión

- Con el *mismo* h, Euler comete un error grande, Heun mejora bastante, y RK4 es prácticamente idéntico a la solución exacta.
- Para lograr la precisión de RK4 usando Euler/Heun habría que usar pasos mucho más pequeños $\Rightarrow$ más iteraciones y más costo.

Formulación vectorial

Escribimos el sistema como:

$$\mathbf{y}(t) = \begin{pmatrix} P(t) \\ D(t) \end{pmatrix}, \qquad \mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}),$$

con

$$\mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} \alpha P - \beta PD \\ \delta PD - \gamma D \end{pmatrix}.$$

RK4 se aplica componente a componente:

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4).$$

Etapas de RK4 para presa y depredador

Sea $\mathbf{y}_k = (P_k, D_k)^\top$. Definimos:

$$
\begin{aligned}
\mathbf{k}_1 &= \mathbf{f}(t_k, P_k, D_k),\\
\mathbf{k}_2 &= \mathbf{f}\left(t_k + \frac{h}{2},\ P_k + \frac{h}{2}k_{1P},\ D_k + \frac{h}{2}k_{1D}\right),\\
\mathbf{k}_3 &= \mathbf{f}\left(t_k + \frac{h}{2},\ P_k + \frac{h}{2}k_{2P},\ D_k + \frac{h}{2}k_{2D}\right),\\
\mathbf{k}_4 &= \mathbf{f}(t_k + h,\ P_k + hk_{3P},\ D_k + hk_{3D}),
\end{aligned}
$$

donde k_{iP} y k_{iD} son las componentes de $\mathbf{k}_i$.

$$
\begin{aligned}
P_{k+1} &= P_k + \frac{h}{6}(k_{1P} + 2k_{2P} + 2k_{3P} + k_{4P}),\\
D_{k+1} &= D_k + \frac{h}{6}(k_{1D} + 2k_{2D} + 2k_{3D} + k_{4D}).
\end{aligned}
$$

Ejemplo numérico de RK4 en Lotka–Volterra

Datos usados en la exposición:

$$\alpha = 0{,}8,\ \beta = 0{,}05,\ \delta = 0{,}02,\ \gamma = 0{,}6,\ P_0 = 80,\ D_0 = 20,\ h = 0{,}05.$$

Primer paso de $t_0 = 0$ a $t_1 = 0{,}05$:

$$\mathbf{y}_0 = (P_0, D_0) = (80, 20).$$

- $k_1 = f(t_0, \mathbf{y}_0) \approx (-16,\ 20)$.
- $k_2 = f\left(t_0 + \frac{h}{2},\ \mathbf{y}_0 + \frac{h}{2}k_1\right) \approx (-17{,}91,\ 20{,}34)$.
- $k_3 \approx (-17{,}93,\ 20{,}32), \quad k_4 \approx (-19{,}84,\ 20{,}64)$.
- Actualización:

$$\mathbf{y}_1 \approx (79{,}10,\ 21{,}02).$$

Así se repite el proceso para obtener toda la trayectoria.

Interpretación de la simulación

- A cada paso, RK4 actualiza simultáneamente P_k y D_k.
- El esquema captura las **oscilaciones periódicas**: las presas crecen, luego los depredadores aumentan, después las presas disminuyen, etc.
- La trayectoria $(P(t), D(t))$ se aproxima por los puntos numéricos (P_k, D_k).
- Para pasos suficientemente pequeños, las órbitas numéricas se ajustan bien a las órbitas teóricas cerradas del modelo.

Herramientas y tecnologías

- **Python**: lenguaje principal para la lógica numérica.
- **NumPy**: manejo eficiente de arreglos y operaciones vectoriales.
- **Matplotlib**: gráficos estáticos.
- **Dash**: interfaz web interactiva (sliders, botones, etc.).
- **Manim**: animaciones y visualización de la dinámica.
- Backend organizado en módulos:
  - `rhs_lotka.py`: campo vectorial del modelo.
  - `simulation.py`: integración con RK4.
  - `app.py`: arranque de la aplicación.

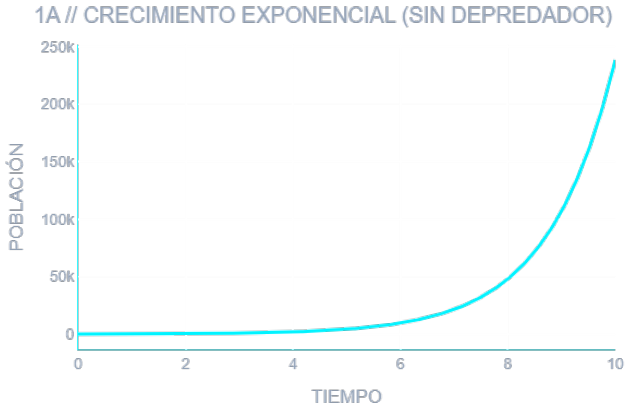
Modelo Lotka–Volterra en Python

```python
import numpy as np

def rhs_lotka(t, y, a, b, d, g):
    # Campo vectorial del modelo depredador-presa
    P, D = y  # P: presas, D: depredadores

    dPdt = a * P - b * P * D
    dDdt = d * P * D - g * D

    return np.array([dPdt, dDdt])
```

Función RK4 para Lotka–Volterra

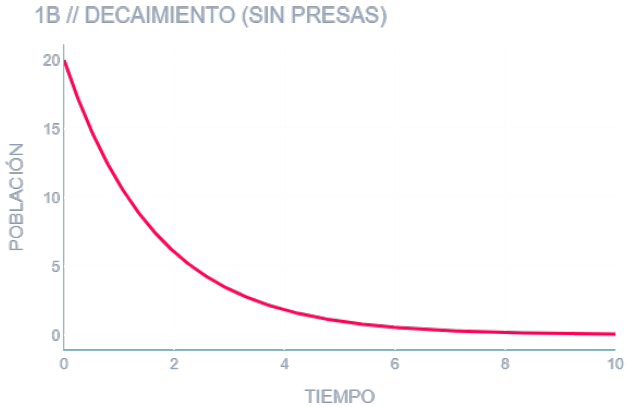
```python
def rk4_lotka(a, b, d, g, P0, D0,
              t0=0.0, tf=50.0, h=0.05):
    # Integra el sistema Lotka-Volterra con RK4
    t = np.arange(t0, tf + h, h)
    Y = np.zeros((len(t), 2))
    Y[0] = [P0, D0]

    for i in range(1, len(t)):
        ti = t[i - 1]
        yi = Y[i - 1]

        k1 = rhs_lotka(ti, yi, a, b, d, g)
        k2 = rhs_lotka(ti + h/2.0, yi + h*k1/2.0, a, b, d, g)
        k3 = rhs_lotka(ti + h/2.0, yi + h*k2/2.0, a, b, d, g)
        k4 = rhs_lotka(ti + h, yi + h*k3, a, b, d, g)

        Y[i] = yi + (h/6.0) * (k1 + 2*k2 + 2*k3 + k4)
    return t, Y[:, 0], Y[:, 1]
```

Gráfico 1A: crecimiento sin depredador



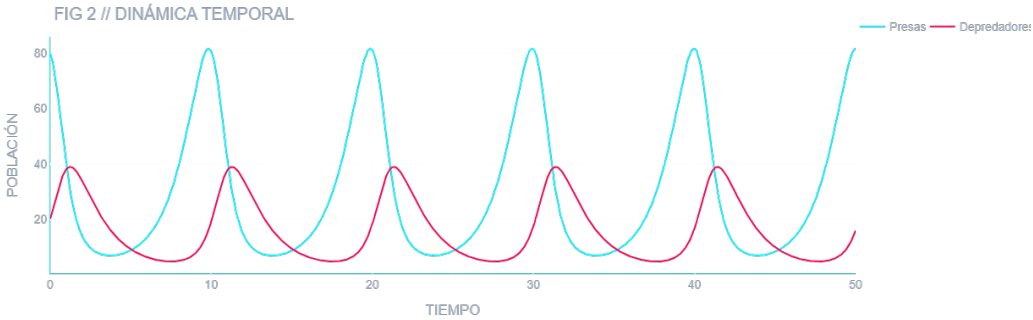
- Se anula la interacción: $\beta = 0$ (sin depredadores efectivos).
- La ecuación se reduce a $dP/dt = \alpha P$.
- La población de presas crece de forma exponencial sin límite.

Gráfico 1B: decaimiento sin presas



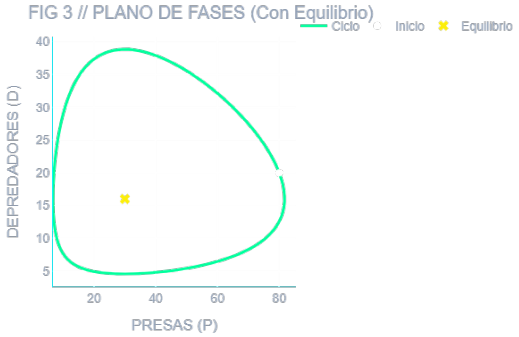
- No hay presas: $P(t) = 0$.
- La ecuación queda $dD/dt = -\gamma D$.
- La población de depredadores decae exponencialmente hasta casi 0.

Fig 2: dinámica temporal presa–depredador



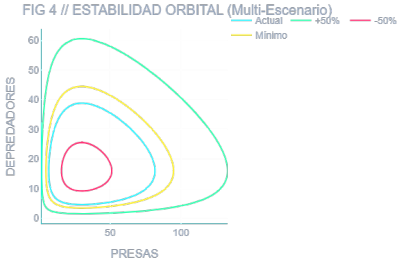
- Se observan **oscilaciones periódicas** de $P(t)$ y $D(t)$.
- Los picos de presas se adelantan a los de depredadores (desfase).
- Cuando las presas son abundantes, crecen los depredadores; luego las presas disminuyen, y después caen los depredadores.

Fig 3: plano de fases con equilibrio



- Trayectoria del sistema en el plano (P, D).
- Órbita cerrada alrededor del equilibrio $(\bar{P}, \bar{D}) = (30, 20)$.
- Punto inicial y punto de equilibrio marcados explícitamente.
- Muestra el carácter cíclico del modelo de Lotka–Volterra.

Fig 4: estabilidad orbital (multi–escenario)



- Comparación entre:
  - órbita base (*Actual*),
  - escenario con parámetros aumentados ($+50\%$),
  - escenario con parámetros reducidos (-50%),
  - caso *Mínimo*.
- Las órbitas cambian de tamaño y posición, pero siguen siendo cerradas.
- Indican **estabilidad orbital**: el comportamiento cualitativo se conserva ante cambios moderados de parámetros.

Conclusiones

- El modelo de Lotka–Volterra, aunque sencillo, captura la esencia de la interacción depredador–presa y los ciclos poblacionales.
- El método RK4 proporciona una aproximación numérica **precisa y estable** para este tipo de sistemas.
- La implementación en Python y la app interactiva permiten experimentar rápidamente con distintos parámetros y visualizar los efectos.
- La combinación de modelado matemático, métodos numéricos y visualización es una herramienta poderosa para estudiar sistemas dinámicos.
- Como trabajo futuro, se pueden incluir modelos más realistas: capacidad de carga, términos logísticos, ruido, más especies, etc.

Referencias


R. L. Burden, J. D. Faires, *Análisis Numérico*, Cengage Learning, 2011.


S. C. Chapra, R. P. Canale, *Métodos Numéricos para Ingenieros*, McGraw–Hill, 2015.


J. Kiusalaas, *Numerical Methods in Engineering with Python 3*, Cambridge University Press, 2013.


S. Forrest, *Predator–Prey Models*, University of New Mexico.


A. E. Parker, *Runge–Kutta 4 (and other numerical methods for ODEs)*.

¡Gracias!