Solving Differential Equation Systems Using Metaheuristics

Liviu Octavian Mafteiu-Scai [1], Alexander Hauer [1], Roxana Teodora Mafteiu-Scai [1]

[1] *Faculty of Informatics, West University of Timisoara, Romania*

Abstract **- Solving systems of ordinary differential equations is important because they model interactions among variables, enabling control and prediction in many real-world problems. This paper analyzes the performance of five metaheuristic algorithms in solving these systems and compares them with two classical mathematical methods**.

Keywords **- particle swarm optimization, genetic algorithms, ant colony optimization, differential evolution, Runge–Kutta, systems of differential equations.**

DOI: 10.18421/SAR92-04
https://doi.org/10.18421/SAR92-04

Corresponding author: Liviu Octavian Mafteiu-Scai,
Faculty of Informatics, West University of Timisoara, Romania
Email: liviu.mafteiu@e-uvt.ro

Received: 15 May 2026.
Revised: 17 June 2026.
Accepted: 23 June 2026.
Published: 27 June 2026.

1. Introduction

In simple terms, a system of differential equations is a set of equations involving several dependent variables, each of which is a function of the same independent variable, where each equation contains these functions and their derivatives. These systems can be linear or nonlinear, and ordinary or partial.

Many real-world problems in fields such as mathematics, chemistry [1], ecology [2], physics [3], biology [4], or engineering [5] are modeled mathematically by systems of differential equations.

1.1. Classical Mathematical Methods

In general, most realistic systems of differential equations do not have exact analytical solutions, so approximation methods have to be used. Runge-Kutta (RK) is an old iterative (1900) method most used for the approximate solutions of nonlinear equation systems. Over time, several variants of the method were developed, especially for special cases of systems of equations. The most well-known member of this family is generally referred to as "RK4" or the "classic Runge–Kutta method" [6], [7], [8]. This method was the first method used in our comparative study. The Fourth-Order Runge-Kutta method (RK4) is based on calculating a weighted average of four slope increments to estimate the next value. It is known as a method with good computational precision.

The second mathematical method used in our comparative study was the Adams-Bashforth-Moulton-fourth-order method (ABM4) [9], a numerical technique used to approximate solutions for ordinary differential equations (ODEs). The fundamental concept is to utilize data from past and present steps to determine the upcoming step.

1.2. Metaheuristics

Particle Swarm Optimization (PSO) [10], [11], [12] is a population-oriented metaheuristic where a collection of particles progresses in increments across a specified area. At every stage, the algorithm assesses the objective function for every particle in the group, and based on this assessment, a decision is made for the subsequent step. Genetic Algorithm (GA) [13] is a search-driven optimization method motivated by Charles Darwin's theory of natural selection. GA can tackle intricate optimization challenges by replicating biological mechanisms (selection, crossover, and mutation) to guide a group of potential solutions towards a global optimum. GAs are suggested for use in extensive and non-linear search areas.

Ant Colony Optimization (ACO) [14] is a nature-inspired algorithm utilized to discover optimal paths in graphs, resembling how ants locate the shortest routes through pheromone trails. Ants favor routes that have more potent pheromones and are shorter in length. A method known as evaporation is employed to prevent local optima. A revised version for continuous domains was suggested in [15]. Ant Colony Optimization for continuous domains (ACOR) employs Gaussian Kernel Density Estimation to modify discrete pheromone-based foraging for real-valued environments.

Differential Evolution (DE) is an evolutionary algorithm utilized to enhance a problem by attempting to refine the existing solution with respect to a preferred standard [16], [17]. The core concept of optimization involves generating new potential solutions by merging existing ones using specific formulas. We proceed with the top candidate solution, which is the one demonstrating the highest fitness. It is important to note that a solution may not be ensured.

1.3. State of the Art

Although there are few published studies on the use of artificial intelligence (AI) methods to solve ODEs and systems of ODEs, some of them are discussed below.

In paper [18], a PSO algorithm is proposed for solving linear and nonlinear ordinary differential equations (ODE), and solution approached as an optimization problem. In addition to the PSO method, traditional methods such as Fourier series were also used. Thus, the partial sum of the Fourier series with the center at one end of the ODE solution interval is used as the approximation function coefficients.

A different application of PSO for approximating solutions to ODEs constructs a fitness function by combining the discrete least-squares weighted function with a penalty function [19].

A modified ACO algorithm is proposed in paper [20] for solving ordinary differential equations (ODEs) and partial differential equations (PDEs). The changes brought to the standard ACO variant consist in not using the distance criterion and using a probability function for the quantity of the pheromone level.

In paper [21], a method for solving Stochastic Differential Equations (SDE) is proposed that uses Ant Colony Programming (ACP) to build the fitness function.

In paper [22], a method is proposed to solve the ordinary differential equations (ODEs) and partial differential equations (PDEs) that uses a modified Genetic Algorithm (GA). In fact, the GA does not use a population of solutions but instead will make a sequence of mutations of an individual solution using a classical fitness function.

To obtain an approximate solution of ODEs, in paper [23], an adaptive differential evolution algorithm is proposed, where the solving process is viewed as an optimization problem. The Fourier periodic expansion function is used for a solution as close as possible to the real one.

2. Proposed Methods to Solve Systems of ODEs

In the following subsections, heuristic methods proposed for solving systems of ODEs are presented.

2.1. PSO in Solving ODE Systems

Particle Swarm Optimization (PSO) is a method inspired by the social behavior of flocks of birds. Each solution is a "particle" that "flies" through the search space, adjusting its trajectory according to the best personal experience and the best discovery/experience of the entire particle swarm. The strength of the method is its high convergence speed in the case of continuous search spaces.

To ensure good convergence on highly nonlinear cost domains of differential equations and due to the fact that PSO suffers from "speed explosion", an architecture based on constriction coefficients [12] was developed that automatically adjusts the speed equation, eliminating the need to impose arbitrary speed limits that can be incorrect. The basic algorithm is presented below.

Algorithm 1: Particle Swarm Optimization (PSO)

Require: Objective function f(x, t), Population N_{pop}, Max Iterations T_{max}
Require: Search Space $[X_{min},X_{max}]$, Parameters w,w_{damp}, $c1$, $c2$
Ensure: Global best position g_{best}
1: $V_{max} \leftarrow 0.2 \times (X_{max} - X_{min})$ // Set maximum velocity threshold
2: $V_{min} \leftarrow -V_{max}$
3: for $i = 1 \ldots N_{pop}$ do // Initialization Phase
4: $xi \sim U(X_{min},X_{max})^D$
5: $vi \leftarrow 0^D$
6: $C_i \leftarrow f(x_i, t)$
7: $p_i \leftarrow x_i$; $P_{cost,i} \leftarrow C_i$
8: if $P_{cost,i} < G_{cost}$ then
9: $g_{best} \leftarrow p_i$; $G_{cost} \leftarrow P_{cost,i}$
10: end if
11: end for
12: for $t = 1 \ldots T_{max}$ do // Main Optimization Loop
13: for $i = 1 \ldots N_{pop}$ do
14: $r_1, r_2 \sim U(0, 1)^D$
15: $v_i \leftarrow wv_i + c1r_1 \circ (p_i - x_i) + c_2r_2 \circ (g_{best} - x_i)$
16: $v_i \leftarrow max(V_{min}, min(V_{max}, v_i))$ // Apply velocity clamping
17: $x_i \leftarrow x_i + v_i$
18: $x_i \leftarrow max(X_{min}, min(X_{max}, x_i))$ // Apply spatial bounds
19: $C_i \leftarrow f(x_i, t)$
20: if $C_i < P_{cost,i}$ then
21: $p_i \leftarrow x_i$; $P_{cost,i} \leftarrow C_i$
22: if $P_{cost,i} < G_{cost}$ then
23: $g_{best} \leftarrow p_i$; $G_{cost} \leftarrow P_{cost,i}$
24: end if
25: end if
26: end for
27: $w \leftarrow w \times w_{damp}$ // Inertia decay
28: end for
29: return g_{best}

Notation Legend:

- N_{pop}, T_{max}, D: Number of particles, max iterations, and problem dimensions.
- X_{min}, X_{max}: Spatial boundaries of the search space.
- V_{min}, V_{max}: Dynamic boundaries clamping the maximum velocity of a particle.
- w, w_{damp}: Inertia weight and its per-iteration decay multiplier.
- c_1, c_2: Cognitive (personal) and social (global) acceleration coefficients.
- x_i, v_i, C_i: Current position vector, velocity vector, and evaluation cost of the i-th particle.
- p_i, P_{cost}, i : The best historical position vector found by the i-th particle and its cost.
- g_{best}, G_{cost}: The best historical position vector found by the entire swarm and its cost.
- r_1, r_2: Vectors of uniformly distributed random numbers $\in$ [0, 1].
- The Hadamard product (element-wise vector multiplication).

Notes:

- Velocity Clamping: Velocity limits (Vmax and Vmin) are strictly clamped to 20% of the search space domain. This prevents particles from oscillating out of control or overshooting the bounds.
- Inertia Damping: An inertia damping multiplier (w_{damp}) is applied at the end of each iteration. This gradually shrinks the inertia weight (w), autonomously shifting the swarm's behavior from broad global exploration early on to focused local exploitation at the end.
- Strict Bounding: Positions are explicitly clipped to the search space boundaries immediately after the velocity update to ensure no invalid parameters are evaluated.

2.2. GA in Solving ODE Systems

Based on evolutionary theory, the GA algorithm functions with a "population" of solutions that undergo selection, crossover, and mutation. The primary benefit: possesses outstanding global search ability, preventing the algorithm from becoming trapped in local minima.

Instead of binary representation, a real-coded GA representation was used using the BLX-α crossover operator, allowing the algorithm to extrapolate beyond the strict limits of the parents. The intensity of the Gaussian mutation decreases dynamically over the iterations. The basic algorithm is presented below.

Algorithm 2: Real-Coded GA with BLX-α and Dynamic Gaussian Mutation

Require: Objective function f(x, t), Population N_{pop}, Max Gens T_{max}, Dims, D
Require: Search Space $[X_{min},X_{max}]$, Crossover rate p_c, Tournament size k
Ensure: Global best individual g_{best}
1: $p_m \leftarrow 1/D$
2: $\sigma_{start} \leftarrow 0.1 \times (X_{max} - X_{min})$
3: $\sigma_{end} \leftarrow 0.001 \times (X_{max} - X_{min})$
4: Generate initial population $P_{current} = \{x_1, \ldots, x_{Npop}\}$ where $x_i \sim U(X_{min},X_{max})^D$
5: Initialize gbest with $C_{best} \leftarrow \infty$
6: for gen = 1 . . . T_{max} do // Main Evolutionary Loop
7: for $x_i \in P_{current}$ do // Evaluation Phase
8: $C_i \leftarrow f(x_i, t)$
9: if $C_i < C_{best}$ then
10: $g_{best} \leftarrow x_i$; $C_{best} \leftarrow C_i$
11: end if
12: end for
13: $\sigma \leftarrow (\sigma_{start} - gen) / T_{max} \times (\sigma_{start} - \sigma_{end})$ // Dynamic mutation decay
14: $P_{next} \leftarrow \{g_{best}\}$ // Elitism: Preserve best individual
15: while $|P_{next}| < N_{pop}$ do // Reproduction Phase
16: $p_1 \leftarrow TournamentSelection(P_{current}, k)$
17: $p_2 \leftarrow TournamentSelection(P_{current}, k)$
18: $r \sim U(0, 1)$
19: if $r < p_c$ then // BLX-α Crossover
20: $\alpha \sim U(-0.2, 1.2)$
21: $c_1 \leftarrow \alpha p_1 + (1 - \alpha)p_2$
22: $c_2 \leftarrow (1 - \alpha)p_1 + \alpha p_2$
23: else
24: $c_1 \leftarrow p_1$; $c_2 \leftarrow p_2$
25: end if
26: for $c \in \{c_1, c_2\}$ do // Gaussian Mutation
27: for j = 1 . . .D do
28: if $U(0, 1) < p_m$ then
29: $c_j \leftarrow c_j + N(0, \sigma_2)$
30: $cj \leftarrow max(X_{min}, min(X_{max}, c_j))$ // Apply bounds
31: end if
32: end for
33: if $|P_{next}| < N_{pop}$ then
34: $P_{next} \leftarrow P_{next} \cup \{c\}$
35: end if
36: end for
37: end while
38: $P_{current} \leftarrow P_{next}$
39: end for
40: return g_{best}

Notation Legend:

- p_c, p_m: Probabilities of an individual undergoing crossover and dimension-wise mutation;
- k: Tournament size (number of random individuals competing to become a parent).
- σ_{start}, σ_{end}, σ: The initial, final, and current standard deviations dictating the radius of Gaussian mutation.
- $P_{current}$, P_{next}: The sets of individuals comprising the current and subsequent generations.
- p_1, p_2: The position vectors of the two parents selected for reproduction.
- c_1, c_2: The position vectors of the two children generated via crossover.
- α: The blend factor for BLX-α crossover, permitting bounded spatial extrapolation.

Notes:

- Real-Valued Representation: This GA algorithm uses real values instead of being binary coded.
- Dynamic Mutation: P_m, the mutation rate, is 1/D (where D is the number of dimensions). σ represents the mutation intensity. σ starts big, such that the intensity and, in turn, the search space is high. σ gradually decreases over time, lowering mutation intensity, to fine-tune details.
- Tournament Selection: Takes k random individuals and returns the best one.
- Strict Elitism: To guarantee monotonic convergence, the single best individual (g_{best}) from the current generation is copied directly into the next generation. This prevents “de-evolution” where the optimal trajectory is accidentally destroyed by crossover or mutation.
- BLX-α Crossover: Standard arithmetic crossover traps children strictly between the parameter bounds of their parents, leading to spatial collapse in the swarm. By using a Blend Crossover (BLX-α) with U(−0.2, 1.2), children are permitted to extrapolate 20 percent outside the domain of their parents, preserving genetic diversity and allowing the swarm to expand its search area.

2.3. ACO (Discrete) in Solving ODE Systems

The technique draws inspiration from the way ants deposit a chemical trail (pheromones) towards a destination, meaning that better solutions attract more pheromone, thus progressively directing the rest of the colony. Strength: Robust historical memory of the cost landscape.

In the case of using ACO (discrete) for solving ODE systems, the continuous variables have been discretized into a spatial matrix. The decisions for each Fourier dimension have been decoupled (selecting a value for one coefficient does not restrict the choices for the next). The basic algorithm is presented below.

Algorithm 3: Discrete ACO for Continuous Domains

Require: Objective function f(x, t), Ants N_{ants}, Iterations T_{max}, Dims D
Require: Space $[X_{min},X_{max}]$, Buckets B, Parameters α, ρ, Q
Ensure: Global best position g_{best}
1: $V \leftarrow linspace(X_{min},X_{max},B)$ // Discretize continuous space
2: $\tau \leftarrow 1^{D\times B}$ // Initialize D × B phe←romone matrix
3: Initialize g_{best} with Gcost ← ∞
4: for t = 1 . . . T_{max} do // Main Optimization Loop
5: for k = 1 . . .N_{ants} do // Path Construction
6: $p_k \leftarrow []$; $x_k \leftarrow []$
7: for v = 1 . . .D do
8: $P_{v,j} \leftarrow \frac{\tau_{v,j}^{\alpha}}{\sum_{l=1}^{D}\tau_{v,l}^{\alpha}} \quad \forall j \in \{1 \ldots B\}$
9: $b_{k,v} \leftarrow RouletteWheelSelection(P_{v,1\ldots B})$
10: Append $b_{k,v}$ to p_k
11: Append $V[b_{k,v}]$ to x_k
12: end for
13: $C_k \leftarrow f(x_k, t)$
14: if $C_k < G_{cost}$ then
15: $g_{best} \leftarrow x_k$; $G_{cost} \leftarrow C_k$
16: end if
17: end for
18: $\tau \leftarrow \tau \times (1 - \rho)$
19: // Global Pheromone Evaporation
20: $N_{elite} \leftarrow \lfloor N_{ants}/2 \rfloor$ // Top 50% Elitism
21: for e = 1 . . .N_{elite} do // Pheromone Deposit
22: $\Delta\tau \leftarrow Q / (C_e+10^{-9})$
23: for v = 1 . . .D do
24: $b_{e,v} \leftarrow p_e[v]$
25: $\tau_{v,be,v} \leftarrow \tau_{v,be,v} + \Delta\tau$
26: end for
27: end for
28: end for
29: return g_{best}

Notation Legend:

- B: The number of discrete "buckets" (sub-intervals) the continuous domain is sliced into.
- V: The array of the B discrete values mapped uniformly across $[X_{min}, X_{max}]$.
- τ: The D × B dimensional pheromone matrix.
- α, ρ, Q: Pheromone sensitivity/influence factor, evaporation rate, and deposit scaling factor.
- p_k, x_k: The array of bucket indices chosen by the k-th ant and its corresponding continuous spatial vector.
- $P_{v,j}$: The normalized probability of an ant choosing bucket j for dimension v.
- N_{elite}: The number of top-performing ants permitted to deposit new pheromones.
- Δτ: The volume of pheromone to be deposited, inversely proportional to the ant's cost.

Notes:

- Continuous-to-Discrete Mapping: To use discrete ACO on continuous variables, the search space is sliced into a fixed number of discrete "buckets".
- Independent Dimensions: The pheromone matrix is D × B (Dimensions × Buckets). Dimensions are treated independently, meaning picking a bucket for variable 1 does not restrict the choices for variable 2.
- Roulette Wheel Selection: Bucket selection for each dimension is performed probabilistically using a roulette wheel mechanism. The chance of an ant choosing a specific discrete value is directly proportional to its relative pheromone concentration (τα), balancing the exploitation of historically successful parameters with the exploration of untried regions.
- Top-50% Elitism: after global evaporation, only the best half of the ants in a generation are permitted to deposit pheromones. This prevents the swarm from reinforcing bad trajectories.
- Inverse-Cost Deposit: The amount of pheromone dropped is inversely proportional to the ant's cost (Q/cost). Better solutions drop more pheromone exponentially.

2.4. Continuous ACO (ACOR) in Solving ODE Systems

This approach is a mathematical enhancement of the conventional ACO algorithm applicable to continuous domains. It employs a Gaussian probability distribution derived from a collection of optimal solutions. Strength: Merges the probabilistic reasoning of ants with the accuracy of real numbers.

In order not to destroy the close mathematical relationship (coupling) between the harmonics of the same Fourier series, the algorithm was modified so that an ant chooses a single member from the archive as a guide (Solution-Wise Guide Selection) for the entire multidimensional set of coefficients.

The basic algorithm is presented below.

Algorithm 4: Continuous ACOR with Solution-Wise Guide Selection

Require: Objective function f(x, t), Archive Size K, Sample Size M, Dims D
Require: Search Space $[X_{min}, X_{max}]$, *Max Iterations* T_{max}, *Parameters* q, ζ
Ensure: Global best position g_{best}
1: Generate Archive $A = \{s_1, \ldots, s_K\}$ *where* $s_l \sim U(X_{min}, X_{max})^D$
2: Evaluate $C_l \leftarrow f(s_l, t)$ $\forall l \in \{1 \ldots K\}$
3: Sort A such that $C_1 \leq C_2 \leq \cdots \leq C_K$
4: $g_{best} \leftarrow s_1$; $G_{cost} \leftarrow C_1$
5: $w_l \leftarrow exp(-(l-1)^2 / 2q^2K^2)$ $\forall l \in \{1 \ldots K\}$ *// Compute ranks*
6: $P_l \leftarrow \frac{w_l}{\sum_{j=1}^{k} w_j}$ *// Normalize probabilities*
7: for t = 1 . . . T_{max} *do // Main Optimization Loop*
8: for l = 1 . . .K do // Compute deviations
9: $\sigma_{l,i} \leftarrow \zeta \frac{\sum_{r=1}^{k} |s_{l,i} - s_{r,i}|}{k-1}$ $\forall i \in \{1 \ldots D\}$
10: end for
11: // Sample M indices $g_1 \ldots g_M$ *from {1 . .K} using probabilities P*
12: for k = 1 . . .M do // Generate New Ants
13: $\mu \leftarrow s_{gk}$
14: $\sigma_{guide} \leftarrow \sigma_{gk}$
15: $x_k \sim N(\mu, diag(\sigma_{guide}))$ *// Sample from Gaussian kernel*
16: $x_k \leftarrow max(X_{min}, min(X_{max}, x_k))$ *// Apply bounds*
17: $C_k \leftarrow f(x_k, t)$
18: end for
19: $A \leftarrow A \cup \{x_1 \ldots x_M\}$ *// Merge new samples*
20: //Sort A by cost and retain only the top K solutions
21: If $C_1 < G_{cost}$ *then*
22: $g_{best} \leftarrow s_1$; $G_{cost} \leftarrow C_1$
23: end if
24: end for
25: return g_{best}

Notation Legend:

- K, M: Size of the solution archive, and the number of new ants (samples) generated per iteration.
- q: Intensification factor (controls the selection pressure/bias toward top-ranked guides).
- ζ: Deviation-distance ratio (controls the radius/spread of the Gaussian sampling).
- A: The archive storing the K best historical solutions.
- s_l, C_l: The l-th solution vector in the archive and its evaluation cost.
- w_l, P_l: The Gaussian weight and normalized selection probability of the l-th archive member.
- $\sigma_{l,i}$: The standard deviation of the l-th archive member in the i-th dimension, based on spatial crowding.
- μ, σ_{guide}: The position and standard deviation vectors of the single archive member selected to guide a new ant.

Notes:

- Solution-Wise Guide Selection: Standard ACOR selects a different guide for every dimension. To preserve the mathematical coupling between variables, this implementation forces an ant to select a single archive member to guide the entire multidimensional sample.
- Rank-Based Weights (q): Archive members are chosen as guides based on a Gaussian function weighted by their rank. The q parameter controls the selection pressure (smaller q heavily favors the top few elite solutions).
- Dynamic Radius (ζ): The standard deviation (σ_{guide}) used for Gaussian sampling shrinks automatically based on the average distance between archive members. As the archive converges, the search steps naturally become smaller.

2.5. Differential Evolution (DE) in Solving ODE Systems

The approach employs an evolutionary algorithm that improves by leveraging the vectorial difference (geometric distance and direction) among current individuals to steer mutation. Strength: Highly effective and reliable in optimizing intricate functions with closely linked variables (like Fourier coefficients).

The shifting strategy has been changed from a random (rand/1/bin) to a best-individual-oriented (best/1/bin) strategy. The shifting factor (β) is applied as a uniform scalar to preserve the geometric direction of the search path in phase space. The basic algorithm is presented below.

Algorithm 5: Differential Evolution (DE/best/1/bin)

Require: Objective function f(x, t), Population N_{pop}, Dims D, Iterations T_{max}
Require: Search Space $[X_{min},X_{max}]$, Crossover p_{cr}, Scaling limits $[\beta_{min}, \beta_{max}]$
Ensure: Global best individual g_{best}
1: Generate $P = \{x_1, \ldots, x_{Npop}\}$ where $x_i \sim U(X_{min},X_{max})^D$
2: Evaluate $C_i \leftarrow f(x_i, t)$ $\forall i \in \{1 \ldots N_{pop}\}$
3: $g_{best} \leftarrow arg\ min_{xi}\ (C_i)$; $G_{cost} \leftarrow min(C_i)$
4: for t = 1 . . . T_{max} do // Main Evolutionary Loop
5: for i = 1 . . .N_{pop} do
6: Select random indices $r_1, r_2 \in \{1 \ldots N_{pop}\} \setminus \{i\}$ // Mutation Phase
7: $\beta \sim U(\beta_{min}, \beta_{max})$
8: $y \leftarrow g_{best} + \beta(x_{r1} - x_{r2})$ // DE/best/1 strategy
9: $y \leftarrow max(X_{min}, min(X_{max}, y))$ // Apply bounds
10: $j_{rand} \sim U\{1 \ldots D\}$ // Crossover Phase
11: for j = 1 . . .D do
12: $r \sim U(0, 1)$
13: if $j = j_{rand}$ or $r \leq pcr$ then
14: $z_j \leftarrow y_j$
15: else
16: $z_j \leftarrow x_{i,j}$
17: end if
18: end for
19: $C_{trial} \leftarrow f(z, t)$ // Selection Phase
20: if $C_{trial} < C_i$ then
21: $x_i \leftarrow z$; $C_i \leftarrow C_{trial}$
22: if $C_i < G_{cost}$ then
23: $g_{best} \leftarrow x_i$; $G_{cost} \leftarrow Ci$
24: end if
25: end if
26: end for
27: end for
28: return g_{best}

Notation Legend:

- p_{cr}: Crossover probability threshold.
- β: The scalar mutation factor (differential weight) applied to the difference vector.
- β_{min}, β_{max}: The lower and upper bounds for randomly sampling β.
- r_1, r_2: Indices of two randomly selected, distinct population members.
- y: The mutant vector, generated by adding the scaled difference of x_{r1} and x_{r2} to g_{best}.
- j_{rand}: A randomly selected dimensional index mathematically guaranteed to undergo crossover.
- z: The trial vector resulting from the binomial crossover between target x_i and mutant y.
- C_{trial}: The evaluation cost of the trial vector z.

Notes:

- Greedy Mutation (DE/best/1/bin): Instead of using a random base vector (rand/1/bin), the mutation base is always the global best individual. This dramatically accelerates convergence.
- Scalar Scaling Factor (β): A single scalar β is applied to the entire difference vector. Using a vector of random betas would mathematically scramble the geometric direction of the search step.
- Guaranteed Crossover: The binomial crossover generates a random index (j_{rand}) that is mathematically guaranteed to be taken from the mutant vector. This ensures the trial solution is never an exact clone of the target, preventing wasted evaluations.

2.6. Runge-Kutta 4th Order in Solving ODE Systems

Algorithm 6: 4th-Order Runge-Kutta (RK4) with Divergence Protection

Require: Derivative function f(t, y), Initial state $y_0 \in R^D$
Require: Time span $[t_0, t_f]$, Step size Δt
Ensure: Time vector t, State matrix Y
1: $N \leftarrow \lfloor (t_f - t_0)/\Delta t \rfloor + 1$ // Calculate total time steps
2: $t \leftarrow linspace(t_0, t_f, N)$ // Guarantee exact array boundaries
3: $Y \leftarrow 0^{D \times N}$
4: $Y_{*,1} \leftarrow y_0$
5: for i = 1 . . .N − 1 do // Main Integration Loop
6: $t_{curr} \leftarrow t_i$
7: $y_{curr} \leftarrow Y_{*,i}$
8: $k_1 \leftarrow \Delta t \cdot f(t_{curr}, y_{curr})$
9: $k_2 \leftarrow \Delta t \cdot f(t_{curr} + \Delta t/2, y_{curr} + k_1/2)$
10: $k_3 \leftarrow \Delta t \cdot f(t_{curr} + \Delta t/2, y_{curr} + k_2/2)$
11: $k_4 \leftarrow \Delta t \cdot f(t_{curr} + \Delta t, y_{curr} + k3)$
12: $Y_{*,i+1} \leftarrow y_{curr} + 1/6\ (k_1 + 2k_2 + 2k_3 + k_4)$
13: if $max(|Y_{*,i+1}|) > 10^7$ or $Y_{*,i+1}$ contains NaN then
14: $Y_{*,j} \leftarrow 10^7$ $\forall j \in \{i + 1 \ldots N\}$ // Apply penalty
15: break // Early stopping to save compute
16: end if
17: end for
18: return t,Y

Notation Legend:

- f(t, y): The derivative function returning dy/dt.
- y_0: The initial state vector of the system.
- Δt: The integration time step size.
- N: The total mathematically calculated number of discrete time steps.
- t: The vector of all linearly spaced discrete-time evaluation points.
- Y: The D × N matrix storing the simulated state vectors across all time steps.
- k_1, k_2, k_3, k_4: The four Runge-Kutta slope approximations evaluated within a single time step.

Notes:

- Divergence Protection: If an optimization algorithm guesses wild parameters, the physics can explode to infinity. If any state variable exceeds 10^7 or returns NaN, the integration loop instantly breaks to save CPU cycles.
- Penalty Padding: The remainder of an aborted trajectory matrix is filled with 10^7. This heavily penalizes the cost function while maintaining the exact matrix shape needed for vector math.

- Strict Discretization: np.linspace is used instead of np.arange to mathematically guarantee the time array matches the exact length of the target data, preventing floating-point shape mismatches.

2.7. Adams-Bashforth-Moulton in Solving ODE Systems

Algorithm 7: 4th-Order Adams-Bashforth-Moulton (ABM4) Predictor-Corrector

Require: Derivative function f(t, y), Initial state $y_0 \in R^D$
Require: Time span $[t_0, t_f]$, Step size Δt
Ensure: Time vector t, State matrix Y
1: $N \leftarrow \lfloor (t_f - t_0)/\Delta t \rfloor + 1$
2: $t \leftarrow linspace(t_0, t_f, N)$
3: $Y \leftarrow 0^{D\times N}$; $F \leftarrow 0^{D\times N}$
4: $Y_{*,1} \leftarrow y_0$; $F_{*,1} \leftarrow f(t_1, y_0)$
5: for i = 1 . . . 3 do // Bootstrap Phase
6: $Y_{*,i+1} \leftarrow$ RK4 Step(f, t_i,$Y_{*,i}$,Δt)
7: $F_{*,i+1} \leftarrow f(t_i+1, Y$)
8: end for
9: for i = 4 . . .N − 1 do // Main Integration Loop
10: // Predictor: Explicit Adams-Bashforth
11: $y_{pred} \leftarrow Y_{*,i} + \Delta t / 24\ (55F_{*,i} - 59F_{*,i-1} + 37_{F*,i-2} - 9F_{*,i-3})$
12: $f_{pred} \leftarrow f(t_{i+1}, y_{pred})$
13: // Corrector: Implicit Adams-Moulton
14: $Y_{*,i+1} \leftarrow Y_{*,i} + \Delta t/24\ (9f_{pred} + 19F_{*,i} - 5F_{*,i-1} + F_{*,i-2})$
15: $F_{*,i+1} \leftarrow f(t_{i+1}, Y_{*,i+1})$
16: if $\max(|Y_{*,i+1}|) > 106$ then // Divergence Check
17: $Y_{*,j} \leftarrow NaN\ \forall j \in \{i + 1 . . .N\}$
18: break
19: end if
20: end for
21: return t,Y

Notation Legend:

- F: A D×N matrix storing the historical evaluations of the true derivative function f(t, y) at previous time steps.
- y_{pred}: The explicitly predicted system state vector for the next time step (the Adams-Bashforth step).
- f_{pred}: The derivative evaluated specifically at the predicted state y_{pred}.

Notes:

- Predictor-Corrector Efficiency: Unlike RK4 (which requires 4 function evaluations per step), ABM4 only requires 2 function evaluations per step once initialized, significantly reducing computational overhead during millions of metaheuristic iterations.
- RK4 Bootstrapping: ABM4 is a 4-step method, requiring 3 historical points to calculate the next state. Because t = 0 has no history, RK4 is used to strictly compute the first 3 starting steps.
- Divergence Catch: Similar to RK4, a spatial bound (106) is continuously monitored. If the predictor-corrector loop fails to track the curves and explodes, the trajectory is truncated and filled with NaN.

3. Implementation, Experimental Results and Discussions

In our experiments, hardware and software configuration were:

- Processor: AMD Ryzen 9 6900 HX with 32 Gb internal memory;
- Operating system Windows 11
- Programming language: Python 3.12.3;
- Libraries: numpy, matplotlib, panda.

3.1. Running Parameters

To ensure statistical consistency, each algorithm was run 20 times independently for each problem:

- Forward problems (Solving ODEs with Fourier series): Population = 150, Iteration maximum = 1000.
- Inverse problems (Parameter estimation): Population = 50, Iteration maximum = 100.

Fitness Function (Cost):

For the direct problem: It is calculated as the Mean Square Error (MSE) of the residuals of the physical equations at M discrete points, plus a massive penalty λ (e.g. 100 or 1000) for not respecting the initial conditions (x_{target}). For a generic system x(t) = f(t, x):

$$C_{forward} = \frac{1}{M}\sum_{i=1}^{M}(\dot{x}(t_i) - f(t_i, x(t_i)))^2 + \lambda\,(x(0) - x_{target})^2 \quad (1)$$

For the inverse problem*:* It is calculated as the MSE between the simulated trajectory (using the parameter guess and the RK4 integrator) and the empirically observed data (y_{obs}) at K time points:

$$C_{inverse} = \frac{1}{K}\sum_{k=1}^{K}(\mathbf{y}_{sim}(t_k) - \mathbf{y}_{obs}(t_k))^2 \quad (2)$$

If the simulation diverges numerically (reaches 10^7), a rejection penalty of 10^{15} is applied.

Specific Parameters of Each Metaheuristic in Part:

- PSO: An architecture based on the constriction coefficient (κ = 1, ϕ1 = 2.05, ϕ2 = 2.05) was used to guarantee mathematical convergence, abandoning the standard empirical parameters. The effective values resulted and applied in the velocity equation throughout the run were: the inertial coefficient (constriction) χ ≈ 0.7298, without damping (w_{damp} = 1.0), and the acceleration coefficient $c_1 = c_2 \approx 1.4962$.
- GA: Crossing rate r_{cross} = 0.9; mutated, i.e. r_{mut} = 1/D; tournament k = 3.

- ACO (Discrete): Buckets B = 500; Pheromone importance α = 1.0; Evaporation ρ = 0.1; Storage Q = 1.0.
- ACOR (Continuous): Archive size K = 50; Intensification factor q = 0.1; Deviation rate, i.e. distance, ζ = 0.8.
- DE: Scale factor limits shifted, i.e. β ∈ [0.5, 0.7]; Crossover probability p_{cr} = 0.9.

3.2. Use and Representation of ODE Systems

3.2.1. Used ODE Systems

The ODE systems used in our experiments come from real-life problems and were:

- Predator Prey (Lotka–Volterra equations): it is a first-order nonlinear system of differential equations used in biology to simulate the interaction between two species, called predator and prey [24], [25];
- Lorentz equations: Systems of deterministic ordinary nonlinear differential equations that represent forced dissipative hydrodynamic flow. The solutions of such a system can be identified with trajectories in phase space [26].
- Van der Pol oscillator: A second-order differential equation system is used to study a non-conservative, oscillating system with non-linear damping [27].
- Fitting Predator Prey: such a system of nonlinear equations was used for stochastic simulation and analysis of plankton time series from two lakes [28].
- The Three-Body problem is defined in general as the motion of three point masses under Newton's laws of motion and gravitation, which is described by nine coupled, second-order nonlinear differential equations. The advantage of the model lies in parameter estimation and forecasting [29].

3.2.2. Direct Problem (Solving ODEs with Fourier Series)

To approximate continuous trajectories without relying on a stepwise numerical integration, the state variable x(t) is parameterized as a finite sum of Fourier harmonics:

$$x(t) \approx a_0 + \sum_{n=1}^{N} (a_n \cos(n\omega t) + b_n \sin(n\omega t)) \quad (3)$$

- In metaheuristics, an “individual” (particle, chromosome) is directly represented as a one-dimensional vector of real numbers containing Fourier coefficients (a_0, a_n, b_n).
- The size of the vector is partitioned equally for each equation of the system. For example, for Lotka-Volterra (2 variables), the first half stores the coefficients for the prey u, and the second half stores the coefficients for the predator v.
- By evaluating the analytical derivatives of the Fourier series, the differential equations are transformed into residual algebraic equations, transferring the problem from numerical integration to continuous global optimization. All calculations are strictly vectorized with the NumPy library for computational efficiency.

Fundamental Frequency (ω) and Time Domain (t): To align the mathematical approximation with the natural physical oscillations of the system, but also to robustly evaluate the error over a relevant time horizon (uniformly discretized into M = 250 evaluation points), the following configurations were used:

- Lotka-Volterra: ω = 0.433 (natural frequency of the system oscillations, estimated analytically by the relation ω ≈√α · γ), evaluated on the interval t ∈ [0, 15].
- Lorenz attraction: ω = 3.0, evaluated on the interval t ∈ [0, 2]
- Restricted Three-Body Problem: ω = 1.0 (normalized orbital period), evaluated over the interval t ∈ [0, 2π]
- Van der Pol Oscillator: ω = 0.943, evaluated over the interval t ∈ [0, 6.663] (Note: For the inverse problems, the time interval and the discretization step Δt were automatically extracted to exactly match the experimental data from the .csv files).

Physical System Parameters:

- Lotka-Volterra: α = 1.1 (prey birth rate), β = 0.4 (predation rate), δ = 0.1 (predator reproduction rate), γ = 0.4 (predator mortality rate);
- Lorenz attraction: σ = 10.0 (Prandtl number), ρ = 28.0 (Rayleigh number), β = 8/3 (geometric ratio). This specific configuration guarantees the evaluation of the algorithms in a purely chaotic topological regime;
- Restricted Three-Body Problem (R3BP): μ = 0.01215 (mass ratio corresponding to the Earth-Moon system);
- Van der Pol oscillator: μ = 1.0 (nonlinear damping coefficient).

3.2.3. Inverse Problem (Parameter Estimation)

For fitting problems, the individual vector in the metaheuristic no longer represents the Fourier coefficient. Instead, the vector directly represents the unknown parameters of the ODE system.

- Lotka-Volterra example: The vector has dimension D = 4, representing the biological rate values: [α, β, γ, δ].
- Lorenz example: The vector has dimension D = 3, representing the chaotic parameters: [σ, ρ, β].

3.3. Experimental Results

In Table 1, Table 2, Table 3, Table 4, Table 5 and Table 6, the experimental results obtained with the five metaheuristic algorithms and the two classical mathematical methods are presented for the ODE systems discussed in the previous subsections.

The best results are marked in bold.

Table 1. Experimental results for the Predator-Prey system

Solver	Best Cost	Worst Cost	Mean Cost	Standard Deviation`	Average Time (s)
PSO	**6.73**	12.67	8.61	1.32	**46.58**
GA	9.66	15.32	13.13	1.18	**46.52**
ACO	27.17	68.61	43.15	11.81	163.29
ACOR	**6.44**	10.32	7.22	1.00	51.78
DE	**6.22**	**7.08**	**6.35**	**0.25**	61.72
RK4		0.00			0.07
ABM4		0.00			0.06

Table 2. Experimental results for the Three-Body system

Solver	Best Cost	Worst Cost	Mean Cost	Standard Deviation	Average Time (s)
PSO	0.74	51.84	13.43	15.66	90.33
GA	4.89	26.54	12.07	6.16	90.59
ACO	109.26	318.08	179.09	58.21	229.64
ACOR	1.57	33.26	10.73	8.77	**87.63**
DE	**0.00**	**23.26**	**1.42**	**5.01**	97.52
RK4	FAIL	FAIL	FAIL	FAIL	FAIL
ABM4	FAIL	FAIL	FAIL	FAIL	FAIL

Table 3. Experimental results for Fitting Predator-Prey system

Solver	Best Cost	Worst Cost	Mean Cost	Standard Deviation	Average Time (s)
PSO + RK4	**0.03**	**1.79**	0.77	**0.46**	**17.37**
GA + RK4	0.33	6.79	2.49	1.57	**17.61**
ACO + RK4	0.44	5.45	2.90	1.32	20.26
ACOR + RK4	0.25	3.93	1.52	0.95	19.31
DE + RK4	**0.03**	4.74	**0.26**	1.02	**17.44**

Table 4. Experimental results for the Lorenz system, Initial Value Problem

Solver	Best Cost	Worst Cost	Mean Cost	Standard Deviation	Average Time (s)
PSO	**77.89**	1796.66	811.91	408.47	**39.12**
GA	108.36	1418.26	735.71	456.45	**40.38**
ACO	2834.2	7240.16	5021.28	1185.20	111.80
ACOR	**77.89**	1451.82	**698.03**	381.35	39.98
DE	**77.89**	**1220.79**	762.95	**256.69**	46.58
RK4		0.00			0.09
ABM4		0.00			0.08

Table 5. Experimental results for the Van der Pol system, Boundary Value Problem

Solver	Best Cost	Worst Cost	Mean Cost	Standard Deviation	Average Time (s)
PSO	0.09	1.31	0.28	0.29	39.07
GA	0.31	1.33	0.67	0.22	42.54
ACO	1.97	6.61	3.55	1.23	115.98
ACOR	**0.06**	0.86	0.15	0.23	**36.69**
DE	**0.06**	**0.06**	**0.06**	**0.00**	40.92
RK4	FAIL	FAIL	FAIL	FAIL	FAIL
ABM4	FAIL	FAIL	FAIL	FAIL	FAIL

Table 6. Experimental results for Fitting Lorenz system

Solver	Best Cost	Worst Cost	Mean Cost	Standard Deviation	Average Time (s)
PSO + RK4	**0.09**	**0.09**	**0.09**	**0.00**	11.76
GA + RK4	**0.09**	**0.09**	**0.09**	**0.00**	11.60
ACO + RK4	0.11	1.52	0.39	0.34	11.94
ACOR + RK4	**0.09**	43.27	8.72	17.27	11.81
DE + RK4	**0.09**	49.50	15.51	21.06	11.92

4. Conclusion

Based on the experimental data shown in the earlier subsection, the following observations can be made:

- Concerning execution duration (acquiring the solution), classical mathematical methods (RK4, ABM4) are advised when ODE systems can be addressed with them. (refer to Table 1 and Table 4);
- When ODE systems fail to satisfy criteria for resolution through classical mathematical techniques, employing metaheuristic methods, serve as a viable alternative, although the execution time may be suboptimal. Under these circumstances, metaheuristic techniques consistently yield a reasonably good approximate solution (refer to Table 2 and Table 5);
- A conclusive ranking of the utilized heuristics based on execution time cannot be determined, as it significantly relies on the specific working parameters of each heuristic and particularly on the ODE system needing resolution (refer to Table 1, Table 2, Table 4, and Table 5, first five rows).
- Experimental data suggests that combining metaheuristics with classical methods offers a viable alternative, with execution times nearly comparable to those achieved with traditional methods alone (refer to Table 3 and Table 6). Table 7 presents the global average values, which are derived from Table 1, Table 2, Table 4, and Table 5 for all ODE systems examined, categorized by each metaheuristic included. Based on the average results, the DE metaheuristic appears to be a superior choice for addressing ODE systems.

Table 7. Average values for metaheuristics

Solver	Best Cost	Worst Cost	Mean Cost	Standard Deviation	Average Time (s)
PSO	21.37	465.62	208.56	106.44	**53.78**
GA	30.81	365.36	190.39	116.00	55.01
ACO	743.15	1908.37	1311.77	314.11	155.18
ACOR	21.49	374.06	**179.03**	97.84	54.02
DE	**21.04**	**312.80**	192.69	**65.49**	61.68

Regarding hybrid methods, it seems that PSO-RK4 hybridization gives the best average results, as can be seen in Table 8.

Table 8. Average values for hybridization

Solver	Best Cost	Worst Cost	Mean Cost	Standard Deviation	Average Time (s)
PSO + RK4	0.06	**0.94**	**0.43**	**0.23**	**14.56**
GA + RK4	0.21	3.44	1.29	0.78	14.60
ACO + RK4	0.27	3.49	1.64	0.83	16.10
ACOR + RK4	0.17	23.60	5.12	9.11	15.56
DE + RK4	**0.06**	27.12	7.89	11.04	14.68

It is widely recognized that the effectiveness of population-based metaheuristics is heavily influenced by the configuration of their parameters. For future research, we believe that Reinforcement Learning (RL) could be employed to dynamically modify the parameters of a metaheuristic throughout the optimization process.

Due to the fact that current RL methods are specialized and closely linked to a specific metaheuristic, our aim is to create and deploy a generalized RL agent for managing and fine-tuning the parameters of various metaheuristics.

References:

[1]. H. S. Jamwal, S. S. Thakur, and N. Singh, "Applications of Differential Equations in Chemistry," in *Proceedings of ICAMAS-2025*, pp. 1–4, 2025.

[2]. S. N. Busenberg, Ed., *Differential Equations and Applications in Ecology, Epidemics, and Population Problems*. 1st ed. London, U.K.: Academic Press, 2012.

[3]. R. Geroch, "Partial Differential Equations of Physics," in *General Relativity: Proceedings of the Forty Sixth Scottish Universities Summer School in Physics*, G. S. Hall and J. R. Pulham, Eds., pp. 19–60, Routledge, 2017. doi: 10.1201/9780203753804-2.

[4]. T. A. Burton, *Modeling and Differential Equations in Biology*, ser. Lecture Notes in Pure and Applied Mathematics, vol. 58. New York, NY, USA: Marcel Dekker/CRC Press, 1980.

[5]. B. Goodwine, *Engineering Differential Equations: Theory and Applications*. New York, NY, USA: Springer Science+Business Media, 2010. doi: 10.1007/978-1-4419-7919-3.

[6]. J. C. Butcher, "A history of Runge-Kutta methods," *Applied Numerical Mathematics*, vol. 20, no. 3, pp. 247–260, 1996. doi: 10.1016/0168-9274(95)00108-5.

[7]. J. C. Butcher, *Numerical Methods for Ordinary Differential Equations*, 3rd ed. Hoboken, NJ, USA: John Wiley & Sons, 2016. doi: 10.1002/9781119121534.

[8]. F. B. Hildebrand, *Introduction to Numerical Analysis*, 2nd ed. New York, NY, USA: Dover Publications, 1987.

[9]. H. M. Baskonus and H. Bulut, "On the numerical solutions of some fractional ordinary differential equations by fractional Adams–Bashforth–Moulton method," *Open Mathematics*, vol. 13, no. 1, pp. 547–556, 2015. doi: 10.1515/math-2015-0052.

[10]. J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Networks (ICNN'95)*, Perth, WA, Australia, 1995, vol. 4, pp. 1942–1948. doi: 10.1109/ICNN.1995.488968.

[11]. Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC'98)*, Anchorage, AK, USA, pp. 69–73, 1998. doi: 10.1109/ICEC.1998.699146.

[12]. M. Clerc and J. Kennedy, "The particle swarm—explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002. doi: 10.1109/4235.985692.

[13]. J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA, USA: The MIT Press, 1992. doi: 10.7551/mitpress/1090.001.0001.

[14]. M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996. doi: 10.1109/3477.484436.

[15]. K. Socha and M. Dorigo, "Ant colony optimization for continuous domains: adaptation of discrete metaheuristics to continuous optimization," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1155–1173, 2008. doi: 10.1016/j.ejor.2006.06.046.

[16]. R. Storn and K. Price, "Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997. doi: 10.1023/A:1008202821328.

[17]. S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 4–31, 2011. doi: 10.1109/TEVC.2010.2059031.

[18]. M. Babaei, "A general approach to approximate solutions of nonlinear differential equations using particle swarm optimization," *Applied Soft Computing*, vol. 13, no. 7, pp. 3354–3365, 2013. doi: 10.1016/j.asoc.2013.02.005.

[19]. A. T. Abed and A. S. Y. Aladool, "Applying particle swarm optimization based on Padé approximant to solve ordinary differential equation," *Numerical Algebra, Control and Optimization*, vol. 12, no. 2, pp. 321–337, 2022. doi: 10.3934/naco.2021008.

[20]. M. Z. M. Kamali, N. Kumaresan, and K. Ratnavelu, "Solving differential equations with ant colony programming," *Applied Mathematical Modelling*, vol. 39, no. 10–11, pp. 3150–3163, 2015. doi: 10.1016/j.apm.2014.11.003.

[21]. A. Sboui, "On some results for stochastic differential equations via ant colony programming," *International Journal of Analysis and Applications*, vol. 22, 2024. doi: 10.28924/2291-8639-22-2024-105.

[22]. E. A. Hussain and Y. M. Abdul-Abbass, "Solving differential equation by modified genetic algorithms," *Journal of University of Babylon for Pure and Applied Sciences*, vol. 26, no. 10, pp. 233–241, 2018. doi: 10.29196/jubpas.v26i10.1877.

[23]. Z. Zhang, Y. Cai, and D. Zhang, "Solving ordinary differential equations with adaptive differential evolution," *IEEE Access*, vol. 8, pp. 128908–128922, 2020. doi: 10.1109/ACCESS.2020.3008823.

[24]. A. J. Lotka, *Elements of Physical Biology*. Baltimore, MD, USA: Williams & Wilkins, 1925.

[25]. V. Volterra, *Variazioni e fluttuazioni del numero d'individui in specie animali conviventi*. Roma, Italy: Società Anonima Tipografica "Leonardo da Vinci", 1926.

[26]. E. N. Lorenz, "Deterministic nonperiodic flow," in *Universality in Chaos*, 2nd ed., C. H. Campbell, Ed. London, U.K.: Routledge, 2017, pp. 367–378.

[27]. B. van der Pol, "On 'relaxation-oscillations'," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 978–992, 1926. doi: 10.1080/14786442608564127.

[28]. S. R. Carpenter, K. L. Cottingham, and C. A. Stow, "Fitting predator–prey models to time series with observation errors," *Ecology*, vol. 75, no. 5, pp. 1254–1264, 1994. doi: 10.2307/1937451.

[29]. Z. C. Kao, "Classical mechanics: The three-body problem," 2011. Available: https://www.math.uchicago.edu/~may/VIGRE/VIGRE2011/REUPapers/KaoZ.pdf [Accessed: May. 3, 2026]