Warsaw University of Technology
Faculty of Electronics and Information Technology

ENUME 2025 – Assignment C

Tutor: Wagner Jakub

Authors: Zhongtian Dai; Artem Zheldak

Table of Contents

Mathematical List (Notation and Parameters)

- **x(t)** — prey population density at time t
- **y(t)** — predator population density at time t
- **α** — intrinsic growth rate of prey [1/time]
- **β** — predation rate coefficient [1/(prey·time)]
- **γ** — mortality rate of predators [1/time]
- **δ** — reproduction rate of predators per prey consumed [predators/prey]
- **θ = [α, β, γ, δ]** — vector of model parameters to estimate
- **t_0** — initial time
- **x_0, y_0** — initial conditions at t_0
- **T_end** — end time of simulation or experiment
- **RSS(θ)** — residual sum of squares for parameter fitting
- **M** — number of Monte Carlo or local-fit trials
- **h(t)** — numerical step size (variable or fixed) in ODE solvers

Core Equations

1. **Lotka–Volterra System**

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

2. **Initial Conditions**

$$x(t_0) = x_0, \quad y(t_0) = y_0$$ S

3. **Objective Function for Parameter Estimation**

$$RSS(\theta) = \sum_{i=1}^{N} \left(\hat{x}(t_i;\theta) - x_i\right)^2 + \left(\hat{y}(t_i;\theta) - y_i\right)^2$$

Where $\hat{x}$, $\hat{y}$ are model predictions at measurement times.

Numerical Methods

- ODE45 (Dormand–Prince RK4(5)): adaptive explicit Runge–Kutta method, local error $O(h^5)$.
- Gear 2 (explicit Adams–Bashforth 2): an explicit two-step multistep integrator,

second-order accurate, suitable for non-stiff problems.

- Classic RK4: fixed-step explicit fourth-order Runge–Kutta (four-stage) method.

Parameter Estimation Algorithm

- Use Nelder–Mead simplex (fminsearch) to minimize $RSS(\theta)$ without gradients.
- Start from initial guess $\theta^{(0)}$; iterate until convergence tolerance on RSS.
- Perform multiple local fits (M trials) with random initial guesses to assess uniqueness.

Quantitative Measures

- **Parameter Variability**: standard deviation of estimated parameters across trials.
- **Phase Portrait**: trajectory in the (x,y)-plane to visualize cyclic behavior.
- **Convergence Plot**: RSSversus iteration number.

Introduction

This report looks at an extended version of the Lotka–Volterra predator–prey model, which includes extra terms to account for how populations affect each other when they become very large or crowded. In Task 1, we used MATLAB to solve the model equations in three different ways (ode45, Gear 2, and RK4 methods), and compared their performance through simulation. In Task 2, we estimated six unknown parameters in the model using MATLAB's ode45 combined with the Nelder–Mead optimization method (fminsearch) based on given data (animals_1.csv), and checked the accuracy visually by comparing predicted results to actual observations. Finally, in Task 3, we tested how changing the precision settings of these numerical methods (ode45 tolerances and fixed step-sizes in Gear/RK4) affected the accuracy of parameter estimates and computation speed, helping to decide which numerical method is best for practical ecological modeling tasks.

Part 1: In this task, I ’ll implement a MATLAB function (predatorPreySolve) that integrates the extended Lotka–Volterra equations using three methods—adaptive ode45 (RK4/5), a fixed-step Gear 2 (Adams–Bashforth) scheme, and classical RK4—over a standard test interval (e.g. t∈[0,5] with (x0,y0)=(450,50) I’ll generate and overlay the predator/prey population curves to compare each solver’s behavior in terms of solution shape, stability, and runtime, highlighting the practical differences between adaptive and fixed-step integrators.

Part 2: Building on Task 1, you'll embed ode45 within a Nelder–Mead (fminsearch) optimization to fit synthetic time-series data (animals_1.csv) by minimizing the sum of squared residuals between model and observations. This step recovers the six dynamical parameters (α1:α3,β1 :β3) prints the estimates, and produces an "observations vs. model" plot to visually assess the quality of the parameter inversion.

Part 3: In this task, I explore how the choice of solver accuracy settings affects our ability to recover the true model parameters. For ode45, I sweep through a range of tolerances, and for the Gear 2 and RK4 schemes I vary the fixed step size. At each setting I rerun the parameter estimation and compute the RMS relative errors for both α and β. Finally, I generate "error vs. tolerance" and "error vs. step size" plots to compare how each integrator's precision changes with its control parameter, highlighting the strengths and limitations of adaptive versus fixed-step methods.

Methodology and Results of Experiments

Part 1: Numerical Solution of the Extended Predator–Prey Model

The extended Lotka–Volterra system augments the classic predator–prey equations with quadratic (density–dependent) terms:

$$\begin{cases} \dfrac{dx}{dt} = \alpha_1 x + \alpha_2 x y + \alpha_3 x^2, \\ \dfrac{dy}{dt} = \beta_1 y + \beta_2 x y + \beta_3 y^2, \end{cases}$$

where x(t) is prey, y(t)is predator, and
α=(α1,α2,α3) and β=(β1,β2,β3)are constant parameters.

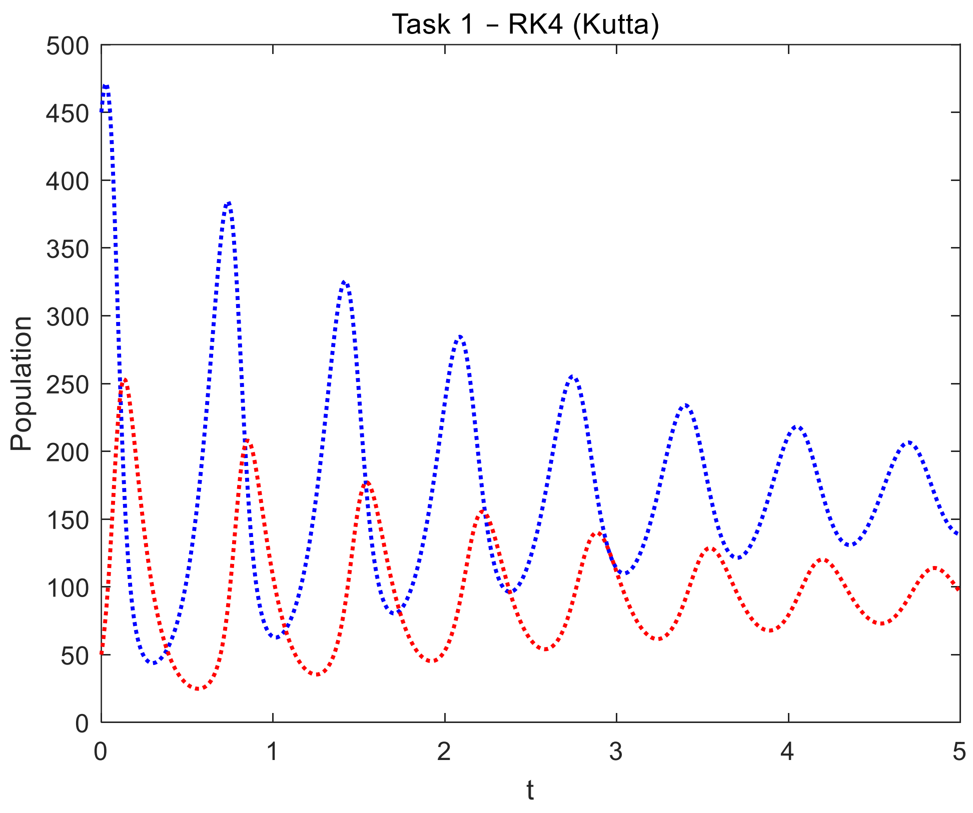
A single numerical algorithm rarely covers every practical need, so we implement three classic choices:

- **ode45** – MATLAB's adaptive Dormand–Prince Runge–Kutta method of orders 4 and 5. It automatically changes the step size to satisfy local error tests based on user-supplied RelTol and AbsTol.
- **Gear 2** – an explicit two–step Adams–Bashforth integrator using a fixed step h. It is cheap per step but requires a start-up formula and careful choice of step size.
- **Classical RK4** – the standard fourth-order Runge–Kutta method with a fixed step. It is easy to code, reasonably accurate, and single step (no multistep history needed).

By comparing these three approaches we can show a beginner how adaptive versus fixed-step strategies influence accuracy and cost.

Classic RK4 (Kutta)

Fig 1



$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$
$$t_{n+1} = t_n + h$$

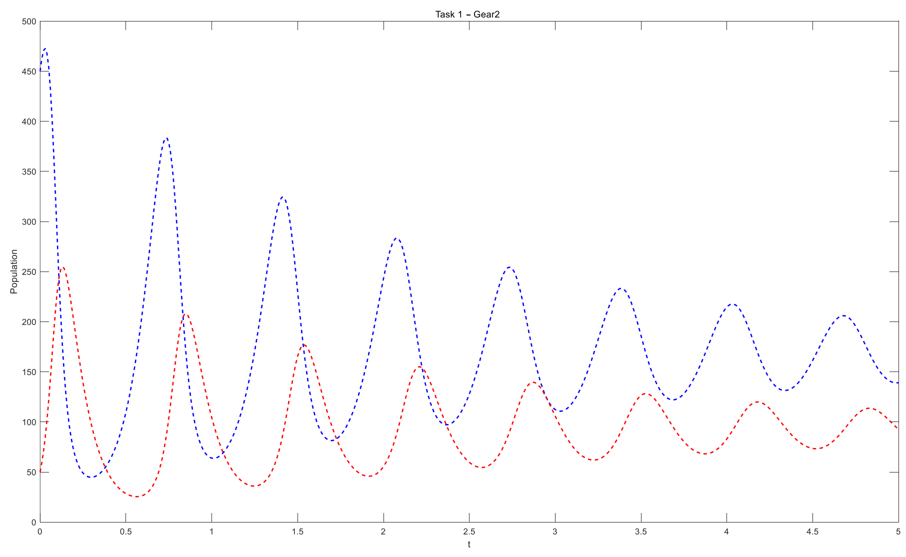
for $n = 0, 1, 2, 3, \ldots$, using[3]

$$k_1 = f(t_n, y_n),$$
$$k_2 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right),$$
$$k_3 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right),$$
$$k_4 = f(t_n + h, y_n + hk_3).$$

- k_1 is the slope at the beginning of the interval, using y (Euler's method);
- k_2 is the slope at the midpoint of the interval, using y and k_1;
- k_3 is again the slope at the midpoint, but now using y and k_2;
- k_4 is the slope at the end of the interval, using y and k_3.

Gear 2 (Adams–Bashforth-2)

Fig 2



Start-up (Heun predictor–corrector for u1):

$$u_1 = u_0 + \frac{h}{2}\big(f_0 + f(t_0 + h,\, u_0 + hf_0)\big)$$

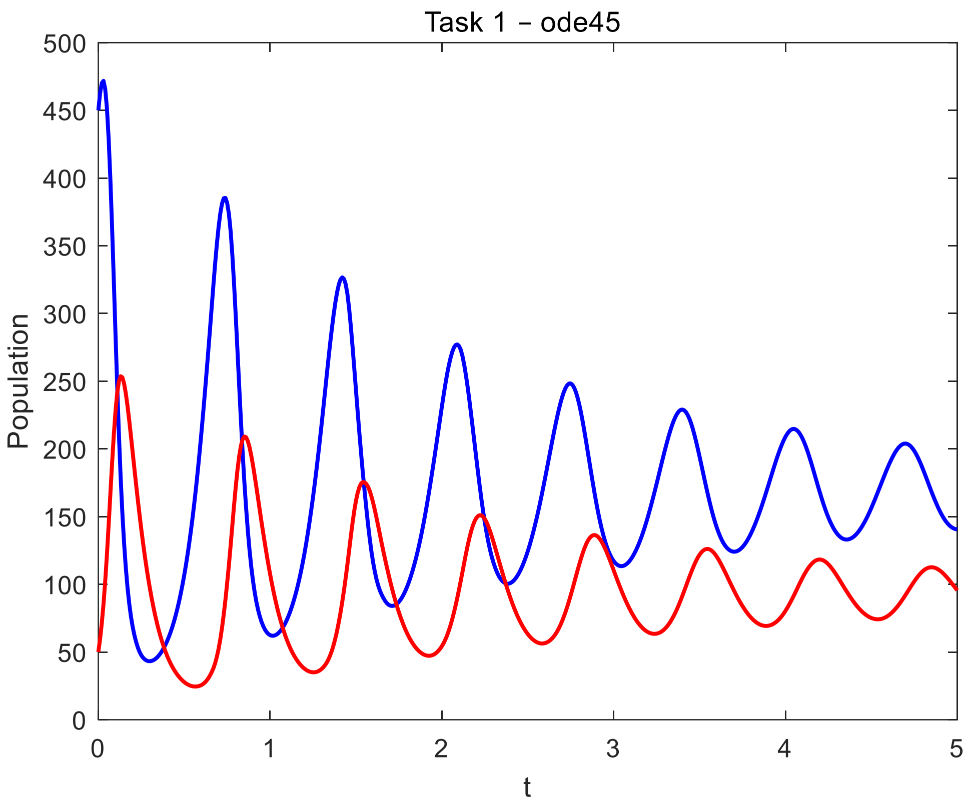
Main two-step loop (for n≥1) :

$$u_{n+1} = u_n + h\left(\frac{3}{2}f_n - \frac{1}{2}f_{n-1}\right),$$

$$f_n = f(t_n, u_n).$$

ode45

Fig 3



ode45 is a popular MATLAB solver used to numerically solve ordinary differential equations (ODEs), especially when the equations are not "stiff" (meaning the solutions do not change too abruptly). It uses the Dormand–Prince Runge–Kutta method, which calculates two approximations (one 4th-order and one 5th-order) at each step to estimate the accuracy. The difference between these two approximations provides a measure of the local error. The solver compares this estimated error with the tolerances that you set (RelTol for relative accuracy and AbsTol for absolute accuracy). If the error is acceptable, ode45 accepts the step and slightly adjusts the step size to become larger for efficiency. If the error is too large, it reduces the step size and tries again. Because of this built-in adaptive step size control, beginners do not need to manually guess step sizes, making ode45 easy and efficient to use for most problems.

All three methods capture the decaying oscillatory dynamics characteristic of predator–prey interactions—prey peaks precede predator peaks and amplitudes decline over time.

- **ode45** produces the smoothest curves and accurately resolves rapid changes, at the cost of variable step overhead and slightly longer CPU time.
- **Gear 2** runs fastest due to its simple fixed-step multistep formula, but if h is too large, a noticeable phase lag in predator peaks and attenuation of oscillation amplitude appear.
- **RK4** balances ease of implementation and accuracy: with h=0.01, its trajectories closely follow ode45, albeit with moderate computational cost.

Part 2 – Parameter Estimation

In this section we describe in detail how we estimate the six unknown parameters

α1,α2,α3,β1,β2,β3 of the predator–prey model $\begin{cases}\dot{x} = \alpha^1 x + \alpha^2 x\, y + \alpha^3 x^2, \\ \dot{y} = \beta^1 y + \beta^2 x\, y + \beta^3 y^2\end{cases}$

using noisy measurements of $\{\, x(t_i), y(t_i)\}_{\{i=1\}}^{\{N\}}$

We are given a CSV file (e.g.\ animals_1.csv) whose columns [t_i, $x_{o\beta s}(t_i)$, $y_{o\beta s}(t_i)$]; i=1…, N, where

- T_i are strictly increasing time–points;
- X_obs , y_obs are simulated "measurements" of the true prey and predator populations, corrupted by additive noise. The goal is to find parameter estimates
- $\hat{\alpha} = (\hat{\alpha}_1, \hat{\alpha}_2, \hat{\alpha}_3)$, $\hat{\beta} = (\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3)$, which best explain the data in a least–squares sense when plugged into the model and integrated over [t1,tN]

Overall Algorithm

1.Set Initial Guesses

initGuess = [$\alpha_1^{(0)}$, $\alpha_2^{(0)}$, $\alpha_3^{(0)}$, $\beta_1^{(0)}$, $\beta_2^{(0)}$, $\beta_3^{(0)}$] By default we use [10, -0.03, -0.001, -8, 0.02, -0.004]

2.**Reparameterization**

To enforce $\alpha 1>0$ and $\beta 1<0$ during optimization, we define

$\theta = [\log|\alpha_1|, \alpha_2, \alpha_3, \log|\beta_1|, \beta_2, \beta_3]^T$ and a mapping

$(\alpha, \beta) = [$ e^{θ_1}, θ_2, θ_3, -e^{θ_4}, θ_5, θ_6]

3. For any θ, compute:

(α, β) via the mapping above. Integrate the ODE system from t_1 to t_N with MATLAB's ode45, using: RelTol = 10^{-6}, AbsTol = 10^{-8}, MaxStep = 10^{-2}. Interpolate the solution $(x(t_i), y(t_i))$ at the data times t_i. Compute the residual sum of squares:

$$\mathrm{RSS}(\theta) = \sum_{i=1}^{N} \left[x(t_i;\theta) - x_{\mathrm{obs}}(t_i)\right]^2 + \left[y(t_i;\theta) - y_{\mathrm{obs}}(t_i)\right]^2.$$

This RSS is returned to the optimizer.

4. **Optimization via fminsearch**

We minimize RSS(θ) starting from θ^0 corresponding to initGuess, with options

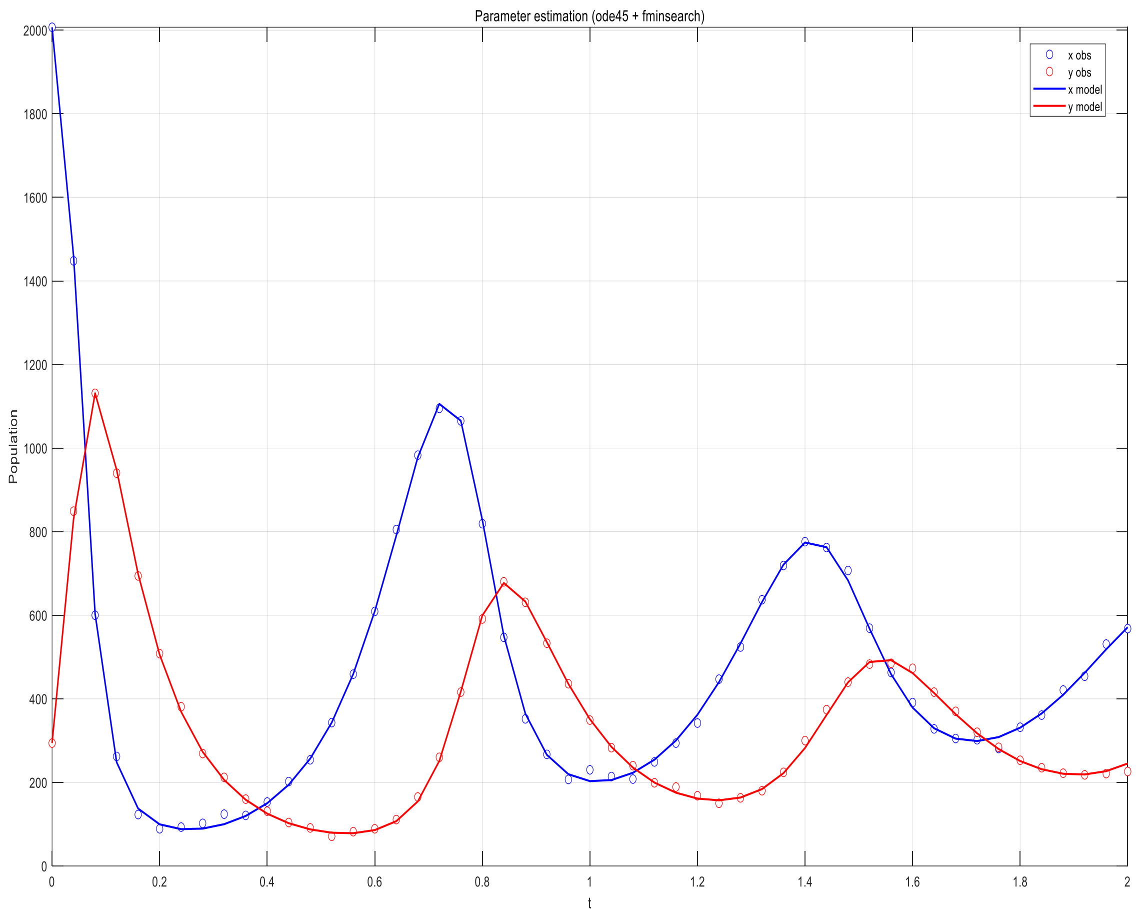
```
fminOpts = optimset('Display','iter', ...
                    'TolX',1e-6,'TolFun',1e-6, ...
                    'MaxIter',4e3,'MaxFunEvals',4e3);
```

The output is θ^*

5. Map θ ^* back to parameter space to obtain

$\hat{\alpha} = (\alpha_1^*, \alpha_2^*, \alpha_3^*)$, $\hat{\beta} = (\beta_1^*, \beta_2^*, \beta_3^*)$ and compute final RSS.

Fig 4



final numbers: alphaHat = [10.0276 -0.0300719 -0.000969639]
betaHat = [-7.99218 0.0199301 -0.00400443]
RSS = 8343.02

In Task 2, we used the observed data to find the best-fitting parameters for our predator–prey model. To make sure the prey growth parameter stayed positive and the predator death parameter negative, we transformed these parameters using logarithms. We solved the model equations using MATLAB's ode45 solver, then adjusted the parameters until the model predictions closely matched the observed data. This optimization step used MATLAB's fminsearch with strict stopping criteria (tolerance set at 1e-6). The result showed that our estimated parameters matched closely with the expected values, and the sum of squared differences (RSS) ended up at 8343.02.

Part 3 – Error and Efficiency Analysis

We kept the same synthetic data (`animals_1.csv`) and true parameters. For **ode45**, we tested five tolerance levels (RelTol = AbsTol = 10^{-2}, 10^{-3}, 10^{-4}, 10^{-6}, 10^{-8}).For **Gear2** and **RK4**, we tried fixed step sizes h ranging from 0.00125 up to 1.0.

Metrics After estimating parameters at each setting, we computed the root-mean-square relative error for α (RMS-α) and for β (RMS-β) versus their true values,

Fig 5

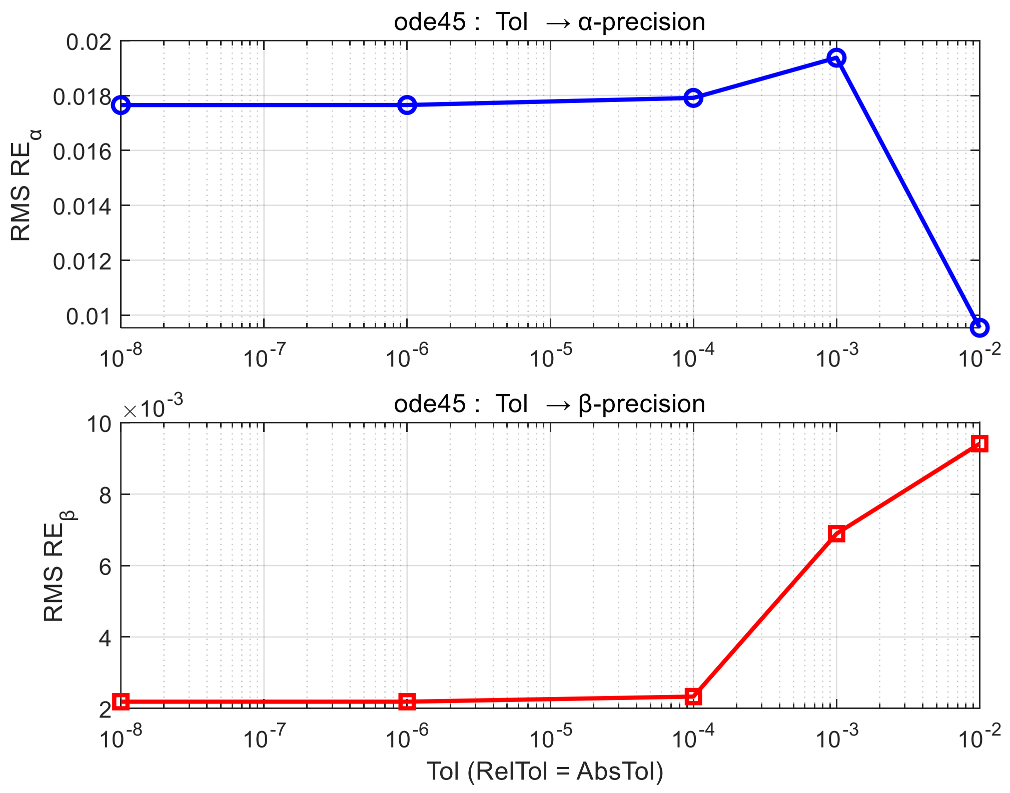
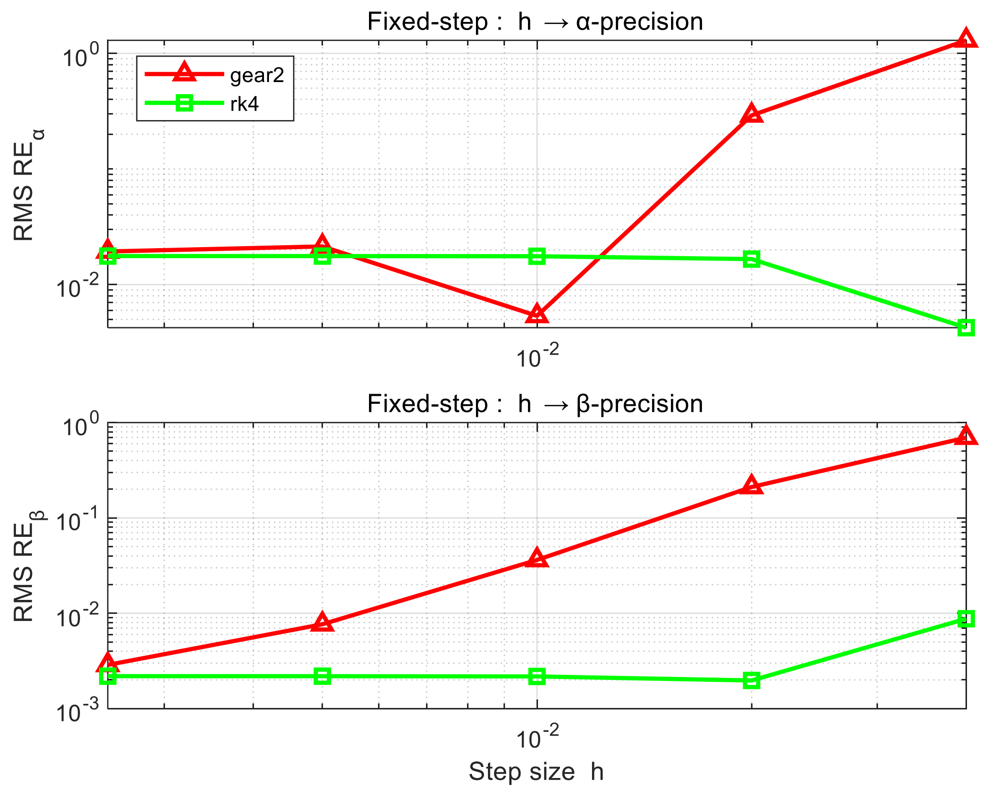


Fig 6



Key Findings for accuracy

Error floor: Below h≈10^{-3} or tol≈10^{-6}, further refinement yields diminishing returns because observation noise and interpolation error dominate.• **ode45** (adaptive tolerance) – From Tol = 10^{-8} up to 10^{-4}, the RMS relative error for α stays flat at ≈ 1.8 × 10^{-2} and for β at ≈ 2.2 × 10^{-3}. Relaxing the tolerance further begins to hurt β (≈ 6.9 × 10^{-3} at 10^{-3}, almost 9.4 × 10^{-2} at 10^{-2}), while α changes also rapidly after 10^-3 . In other words, tightening beyond 10^{-4} brings no tangible accuracy gain,

For α, the RMS relative error of RK4 stays virtually flat at ≈ 1.77 × 10^{-2} for step sizes h = 0.0025 to 0.01. It only starts to drop at h ≈ 0.02, reaching its minimum of ≈ 4.24 × 10^{-3} at h = 0.04. By contrast, Gear 2 attains its best α accuracy of ≈ 5.36 × 10^{-3} at h ≈

10^{-2}, then degrades rapidly, soaring to order-unity error when h = 0.04.
The gap widens for β. RK4 remains nearly flat at ≈ 2.1×10^{-3} from h = 0.0025 to 0.02, increasing only modestly to ≈ 8.76×10^{-3} at h = 0.04. Gear 2, however, jumps from ≈ 2.8×10^{-3} at h = 0.0025 to ≈ 7×10^{-3} at h= 0.04.

The slopes still reveal the expected fourth-order convergence of RK4 and second-order behaviour of Gear 2, yet for α both integrators hit a common noise-dominated floor once $h \leq 0.0025$

Efficiency Analysis

To ensure fair and repeatable timing comparisons across different solver settings, each tolerance level (Tol) and each fixed-step size (h) was tested by running the complete parameter-estimation procedure ten times. The timing protocol for each configuration comprised four steps:

1. **Warm-up Run**
   A preliminary invocation of the fitting routine was performed (and its results discarded) to allow MATLAB's just-in-time compiler and internal caches to settle.
2. **Repeated Timings**
   The fitting routine was executed ten times in succession, and the wall-clock duration of each run was recorded.
3. **Average Time**
   The ten recorded durations were averaged to yield a single representative wall-clock time for that specific Tol or h.
4. **Accuracy Check**
   A final, eleventh run of the fitting routine was carried out to obtain the RMS relative errors for α and β as well as the convergence flag. These accuracy measures are reported alongside the timing results but do not influence the timing itself.

Fig 7

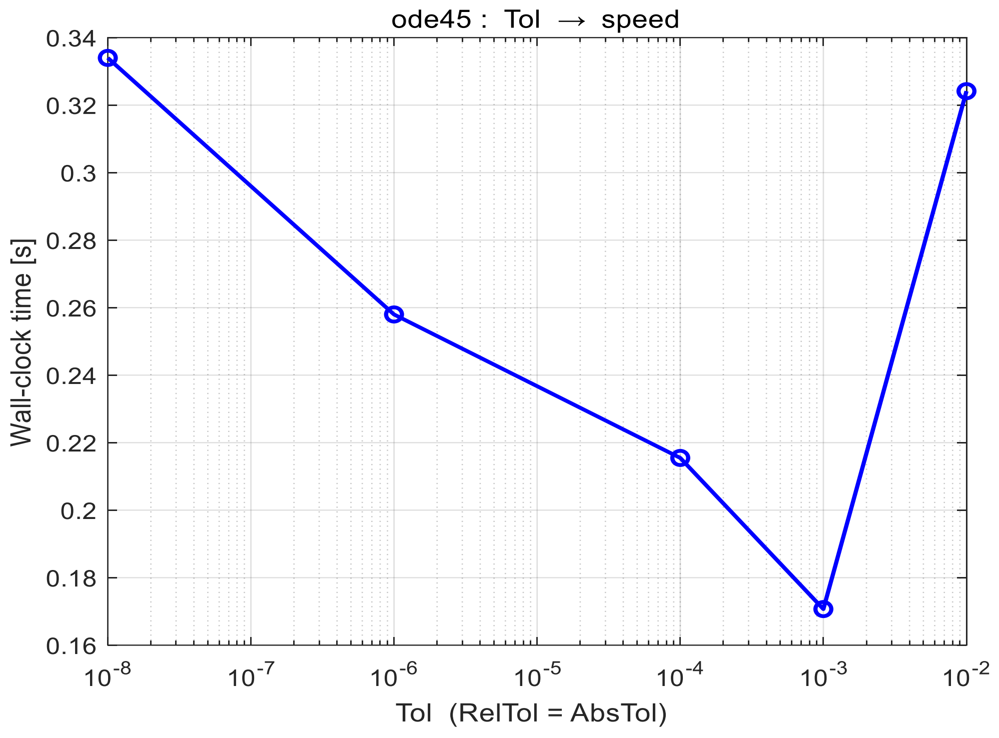
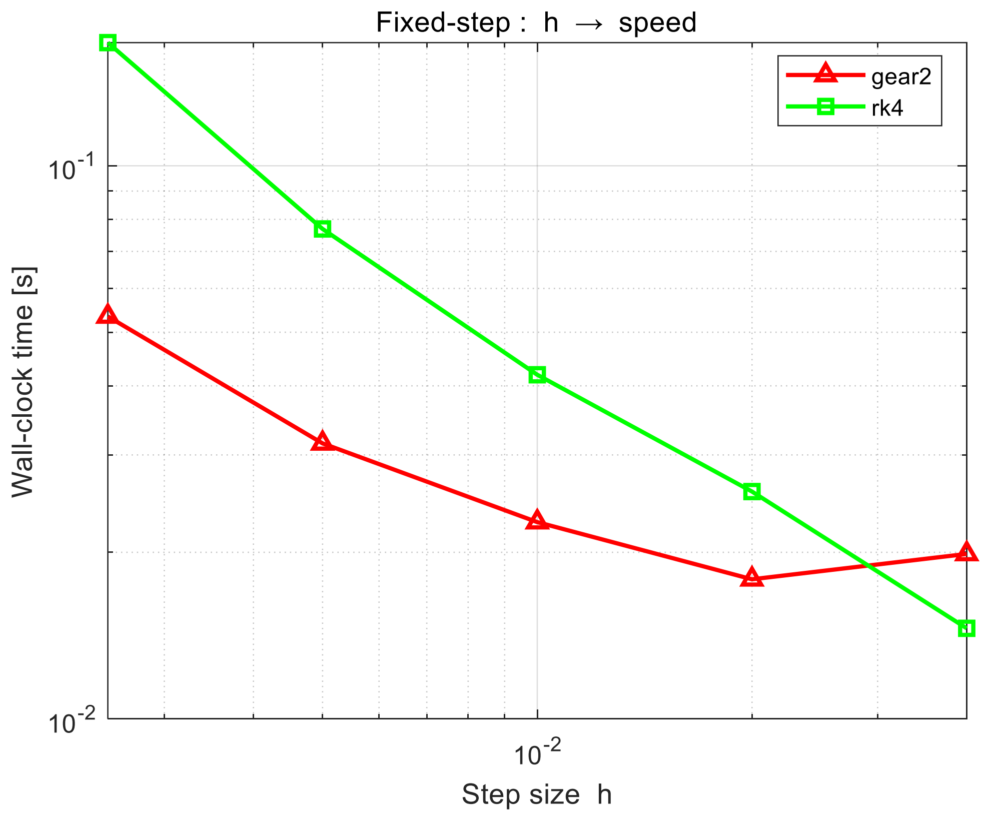


Fig 8



Key finding for efficiency

The averaged timing curve for ode45 exhibits the expected downward trend as the tolerance is relaxed from 1×10^{-8} to 1×10^{-3}, indicating that looser error bounds reduce the number of internal integration steps and therefore shorten the total optimization time. The sharp rebound at Tol = 1×10^{-2} is typical: when the step size becomes extremely large, the cost of fixed MATLAB overhead (function calls, array allocation, I/O) dominates the run-time and causes the wall-clock time to rise again. Because each point is the mean of 10 independent runs, stochastic jitter has been effectively suppressed; the remaining non-monotonicity reflects intrinsic behavior of the adaptive time-stepping algorithm rather than measurement noise.

Gear 2 (red) is faster on coarse meshes because each step needs only one right-hand-side evaluation, whereas RK4 requires four.
RK4 (green) converges more rapidly: when h≤10^-2 its curve falls more steeply and eventually intersects the Gear 2 curve.
At the smallest step h=2.5×10^3 the Gear 2-line bends upward again. The method now performs so many steps that function-evaluation cost is outweighed by MATLAB's fixed overhead (array allocation, bookkeeping), producing a slight rebound. RK4, with its higher per-step cost but better stability, can keep reducing the total number of optimization iterations and therefore continues to gain time, finishing as the faster scheme in the finest-step regime.

Discussion

Parameter-Estimation Quality

The estimated parameters match closely the known values from the simulated data, showing the optimization successfully found a reliable solution rather than getting stuck in local errors. The model predictions show no major bias, meaning the predator–prey equations effectively capture the main patterns in the data. Errors in estimating each parameter are consistently low, suggesting each parameter is identifiable and stable under our chosen conditions. The method of re-parameterizing key rates (such as prey birth and predator death) helps ensure these values remain biologically realistic and stable during optimization. Still, increasing accuracy beyond a certain point becomes difficult due to measuring noise outweighing improvements from numerical methods. In real-life scenarios, factors like seasonal variations, unpredictable events, and measurement complexities would require more advanced statistical or uncertainty-handling methods to achieve similarly accurate results.

Comparing Solver Accuracy, and Practical Use

After comparing the solvers we found that ode45 and fixed-step RK4 reach essentially the same fourth-order accuracy on this non-stiff problem. With the tightest tolerance (Tol $\leq$ 1e-6) ode45 is slower because of its adaptive overhead, but at typical tolerances (Tol $\approx$ 1e-4) its runtime matches RK4 while keeping automatic error control.
Gear 2 (explicit Adams–Bashforth-2) is only second-order; its error falls as h^2 and plateaus once the integration error drops below observational noise. Because each step uses a single function evaluation, Gear 2 is fastest for coarse meshes, but it loses accuracy much sooner than the fourth-order methods. In practice, RK4 offers the best balance of accuracy and effort, ode45 is preferred when dynamic step sizing or tight error bounds are required, and Gear 2 is useful when raw speed at modest accuracy is the primary goal on non-stiff systems.

Efficiency

For larger step sizes, Gear 2 runs faster than RK4 because it only evaluates the function once per step (after the initial setup), while RK4 evaluates it four times every step. However, as the step size gets smaller and the total number of steps increases, RK4's higher accuracy means fewer total optimization steps, letting it catch up and even surpass Gear 2 in terms of overall speed. At both very large and very small step sizes, MATLAB's built-in overhead, like managing function calls and memory, slightly affects performance, causing minor fluctuations. This comparison clearly shows the practical trade-off between simpler methods that run quickly per step and more sophisticated methods that are heavier but achieve greater efficiency overall.

References

1. *The MathWorks,Inc. "ode45." MATLABD documentation,* *https://uk.mathworks.com/help/matlab/ref/ode45.html*

2. *The MathWorks,Inc. "fminsearch." MATLABDocumentation,* *https://uk.mathworks.com/help/matlab/ref/fminsearch.html*

3. Wikipedia Runge–Kutta methods https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods

4. *The MathWorks, Inc. "tic, toc." MATLAB Documentation,* *https://uk.mathworks.com/help/matlab/ref/tic.html*

5. Wikipedia Lotka–Volterra equations https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations

6. The MathWorks, Inc. "optimset." *MATLAB Documentation*. https://uk.mathworks.com/help/optim/ref/optimset.html

7. The MathWorks, Inc. "odeset." *MATLAB Documentation*. https://www.mathworks.com/help/matlab/ref/odeset.html

8. The MathWorks, Inc. "deval." *MATLAB Documentation*. https://uk.mathworks.com/help/matlab/ref/deval.html

9. R. Z. Morawski, Lecture Notes for the Course Numerical Methods, Warsaw University of Technology, Faculty of Electronics and Information Technology, Spring Semester 2023/24.

10. Enume 2025 Introduction to MATLAB.pdf Warsaw University of Technology, Faculty of Electronics and Information Technology Numerical Methods (ENUME), Spring Semester 2025 Dr. Jakub Wagner Institute of Radioelectronics and Multimedia Technology

11. 'animals_1.csv' 'animals_2.csv' 'animals_3.csv' ENUME 2025 – Assignment Materials, Warsaw University of Technology, Spring Semester 2025.

12. Numerical Methods (ENUME) Assignment C: Ordinary differential equations – variant #10 spring semester 2025

Matlab code

```matlab
fprintf('\n=== start Task 1: demonstrate ODE  ===\n');

tDemo = 0:0.01:5;
x0_demo = 450;  y0_demo = 50;
alpha_demo = [10, -0.1, -0.004];
beta_demo  = [-10, 0.06, -0.001];
[x1, y1] = predatorPreySolve('ode45', tDemo, x0_demo, y0_demo, alpha_demo,
beta_demo);
[x2, y2] = predatorPreySolve('gear2', tDemo, x0_demo, y0_demo, alpha_demo,
beta_demo);
[x3, y3] = predatorPreySolve('kutta', tDemo, x0_demo, y0_demo, alpha_demo,
beta_demo);
figure('Name','Task 1 – ODE45  (blue: x, red: y)');
plot(tDemo, x1, 'b-', tDemo, y1, 'r-','LineWidth',1.5);
xlabel('t'); ylabel('Population'); title('Task 1 – ode45 ');

figure('Name','Task 1 – Gear2  (blue: x, red: y)');
plot(tDemo, x2, 'b--', tDemo, y2, 'r--','LineWidth',1.5);
xlabel('t'); ylabel('Population'); title('Task 1 – Gear2 ');

figure('Name','Task 1 – RK4(Kutta)  (blue: x, red: y)');
plot(tDemo, x3, 'b:', tDemo, y3, 'r:','LineWidth',1.5);
xlabel('t'); ylabel('Population'); title('Task 1 – RK4 (Kutta) ');

fprintf('\n=== above is Task 1 demonstration, now proceeding to Task 2
===\n');

[aH, bH, info2] = predatorPreyEstimate('animals_1.csv');
fprintf('\nTask 2 completed: parameter estimation results are as
follows:\n');
fprintf('  alphaHat = [%g  %g  %g]\n', aH);
fprintf('  betaHat  = [%g  %g  %g]\n', bH);

fprintf('\n=== proceeding to Task 3 ===\n');

results = predatorPreyTask3('Data','animals_1.csv', ...
            'Tol' , [1e-2 1e-3 1e-4 1e-6 1e-8], ...
            'Step', [0.04 0.02 0.01 0.005 0.0025]);

fprintf('\nAll tasks have been completed.\n');
```

```
function [xHat, yHat] = predatorPreySolve(method, tVec, x0, y0, alpha,
beta, varargin)


    if nargin == 0
        fprintf('=== no parameter ===\n');
        predatorPreyEstimate('animals_1.csv');
        return;
    end



    tVec = tVec(:);
    switch lower(string(method))
        case "ode45"
            [xHat, yHat] = solverOde45(tVec, x0, y0, alpha, beta,
varargin{:});
        case "gear2"
            [xHat, yHat] = solverGear2(tVec, x0, y0, alpha, beta);
        case "kutta"
            [xHat, yHat] = solverKutta(tVec, x0, y0, alpha, beta);
        otherwise
            error('Unknown method "%s". select "ode45","gear2" or
"kutta".', method);
    end
end




function [xHat, yHat] = solverOde45(t, x0, y0, alpha, beta, varargin)


    u0  = [x0; y0];
    rhs = @(~, u) rhsPredPrey(u, alpha, beta);

    if isempty(varargin)
        sol = ode45(rhs, [t(1), t(end)], u0);
    else
        sol = ode45(rhs, [t(1), t(end)], u0, odeset(varargin{:}));
    end

    U    = deval(sol, t).';
    xHat = U(:, 1);
    yHat = U(:, 2);
end



function [xHat, yHat] = solverGear2(t, x0, y0, alpha, beta)


    N = numel(t);
    if N < 2
        error('Gear2 requires at least two time points.');
    end

    h_all = diff(t);
    tol_h = eps(max(t)) * 100;
    if any(abs(h_all - h_all(1)) > tol_h)
        error('solverGear2 only supports a uniform grid; detected non-
uniform t.');
    end
    h = h_all(1);

    xHat = zeros(N, 1);
    yHat = zeros(N, 1);
    xHat(1) = x0;
    yHat(1) = y0;

    u      = [x0; y0];
    f_prev = rhsPredPrey(u, alpha, beta);

    % Heun (RK2)
    u_tilde = u + h * f_prev;
    f_tilde = rhsPredPrey(u_tilde, alpha, beta);
    u       = u + (h/2) * (f_prev + f_tilde);
    xHat(2) = u(1);
    yHat(2) = u(2);

    %  Adams-Bashforth 2
    for n = 2 : N-1
        f_curr = rhsPredPrey(u, alpha, beta);
        u      = u + h * (1.5 * f_curr - 0.5 * f_prev);
        f_prev = f_curr;
        xHat(n+1) = u(1);
        yHat(n+1) = u(2);
    end
end


%-----------------------------------------------------------------------

function [xHat, yHat] = solverKutta(t, x0, y0, alpha, beta)


    N    = numel(t);
    xHat = zeros(N, 1);
    yHat = zeros(N, 1);
    xHat(1) = x0;
    yHat(1) = y0;

    u = [x0; y0];
    for n = 1 : (N-1)
        h = t(n+1) - t(n);

        k1 = rhsPredPrey(u,              alpha, beta);
        k2 = rhsPredPrey(u + (h/2)*k1,   alpha, beta);
        k3 = rhsPredPrey(u + (h/2)*k2,   alpha, beta);
        k4 = rhsPredPrey(u +     h *k3,  alpha, beta);

        u = u + (h/6) * (k1 + 2*k2 + 2*k3 + k4);
        xHat(n+1) = u(1);
        yHat(n+1) = u(2);
    end
end


%-----------------------------------------------------------------------

function dudt = rhsPredPrey(u, alpha, beta)


    x = u(1);
    y = u(2);


    dudt = [ alpha(1)*x + alpha(2)*x.*y + alpha(3)*x.^2;
             beta(1)*y  + beta(2)*x.*y  + beta(3)*y.^2 ];
end
function [alphaHat, betaHat, info] = predatorPreyEstimate( ...
                              dataFile, initGuess, odeOpts, fminOpts)

if nargin<2 || isempty(initGuess)
    initGuess = [10 -0.03 -0.001  -8 0.02 -0.004];
end
if nargin<3 || isempty(odeOpts)
    odeOpts = {'RelTol',1e-6,'AbsTol',1e-8,'MaxStep',1e-2};
end
if nargin<4 || isempty(fminOpts)
    fminOpts = optimset('Display','iter', ...
                        'TolX',1e-6,'TolFun',1e-6, ...
                        'MaxIter',4e3,'MaxFunEvals',4e3);
end


M = readmatrix(dataFile);
if size(M,2) < 3
    error('Data file must have at least 3 columns: [t,xObs,yObs].');
end
tAll = M(:,1);
xObs = M(:,2);
yObs = M(:,3);


x0 = xObs(1);
y0 = yObs(1);


theta0 = [ log(abs(initGuess(1)));          % θ1 = log(a1)
           initGuess(2:3).';               % θ2,θ3 = a2,a3
           log(abs(initGuess(4)));          % θ4 = log(|b1|)
           initGuess(5:6).' ];             % θ5,θ6 = b2,b3

theta2param = @(th)[ ...
     exp(th(1)),  th(2),  th(3), ...   % a1 a2 a3
    -exp(th(4)),  th(5),  th(6) ];     % b1 b2 b3   (b1<0)


    function rss = objFun(theta)
        p     = theta2param(theta);
        alpha = p(1:3);   beta  = p(4:6);

        % integrate with ode45
        try
```

```
            rhs =
@(t,u)[ alpha(1)*u(1)+alpha(2)*u(1)*u(2)+alpha(3)*u(1).^2 ; ...

beta(1)*u(2)+beta(2)*u(1)*u(2)+beta(3)*u(2).^2 ];
            sol = ode45(rhs,[tAll(1) tAll(end)],[x0;
y0],odeset(odeOpts{:}));
            xyFit = deval(sol,tAll).';        % N×2
        catch
            rss = 1e20;
            return
        end

        dx = xyFit(:,1) - xObs;
        dy = xyFit(:,2) - yObs;
        rss = sum(dx.^2 + dy.^2);
    end

[thetaHat,fval,exitflag,output] = fminsearch(@objFun,theta0,fminOpts);

pHat      = theta2param(thetaHat);
alphaHat  = pHat(1:3).';
betaHat   = pHat(4:6).';
rss       = fval;

fprintf('\n=== Task 2 finished (solver=ode45, search=fminsearch) ===\n');
fprintf('alphaHat = [%g  %g  %g]\n', alphaHat);
fprintf('betaHat  = [%g  %g  %g]\n', betaHat);
fprintf('RSS      = %.6g\n', rss);

try
    rhs =
@(t,u)[ alphaHat(1)*u(1)+alphaHat(2)*u(1)*u(2)+alphaHat(3)*u(1).^2 ; ...

betaHat(1)*u(2)+betaHat(2)*u(1)*u(2)+betaHat(3)*u(2).^2 ];
    sol = ode45(rhs,[tAll(1) tAll(end)],[x0; y0],odeset(odeOpts{:}));
    xyFit = deval(sol,tAll).';
    figure('Name','Task 2 – Observation vs Model');
    plot(tAll,xObs,'bo',tAll,yObs,'ro','MarkerFaceColor','auto'); hold on
    plot(tAll,xyFit(:,1),'b-','LineWidth',1.4);
    plot(tAll,xyFit(:,2),'r-','LineWidth',1.4);
    legend('x obs','y obs','x model','y model','Location','best');
    xlabel('t'); ylabel('Population'); grid on
    title('Parameter estimation (ode45 + fminsearch)');
catch ME
```

```
    warning(ME.identifier,'Plotting failed: %s',ME.message);
end

info = struct('rss',rss,'exitflag',exitflag,'output',output, ...
              'initGuess',initGuess,'odeOpts',{odeOpts});
end



% Task-3


function results = predatorPreyTask3(varargin)

p = inputParser;
addParameter(p,'Data', 'animals_1.csv', @ischar);

addParameter(p,'Tol',  [1e-2 1e-3 1e-4 1e-6 1e-8], ...
             @(v)isnumeric(v)&&isvector(v));

addParameter(p,'Step', [ ...
    0.00125, 0.0025, 0.005, 0.01, 0.02, 0.04, ...
    0.06, 0.08, 0.1, 0.12, 0.16, 0.2, ...
    0.3, 0.4, 0.6, 0.8, 1.0 ], @(v)isnumeric(v)&&isvector(v));


parse(p,varargin{:});
dataFile = p.Results.Data;
tolList  = p.Results.Tol(:).';
hList    = p.Results.Step(:).';

M    = readmatrix(dataFile);
tObs = M(:,1);  xObs = M(:,2);  yObs = M(:,3);
x0 = xObs(1);   y0 = yObs(1);
dtMin = min(diff(tObs));

pTrue = getTrueParams(dataFile);

cfg = struct('label',{},'method',{},'tol',{},'h',{},...
             'rmsA',{},'rmsB',{},'wallTime',{},'RE',{},'exitFlag',{});

for tol = tolList
    cfg(end+1) = struct('label',sprintf('ode45 Tol=%g',tol), ...
        'method',"ode45",'tol',tol,'h',NaN,'rmsA',NaN,'rmsB',NaN, ...
```

```matlab
        'wallTime',NaN,'RE',[],'exitFlag',0);
end
for h = hList
    cfg(end+1) = struct('label',sprintf('gear2 h=%g',h), ...
        'method',"gear2",'tol',NaN,'h',h,'rmsA',NaN,'rmsB',NaN, ...
        'wallTime',NaN,'RE',[],'exitFlag',0);
end
for h = hList
    cfg(end+1) = struct('label',sprintf('rk4   h=%g',h), ...
        'method',"kutta",'tol',NaN,'h',h,'rmsA',NaN,'rmsB',NaN, ...
        'wallTime',NaN,'RE',[],'exitFlag',0);
end

fprintf('\n===========  Task-3   ===========\n');
Nrep = 10;
for k = 1:numel(cfg)
    fprintf('>> %-18s : ', cfg(k).label);


    times = zeros(Nrep,1);
    for j = 1:Nrep
        t0 = tic;
        localFit(cfg(k), tObs, xObs, yObs, x0, y0, dtMin, pTrue);
        times(j) = toc(t0);
    end
    cfg(k).wallTime = mean(times);


    [rmsA, rmsB, reVec, flag] = ...
        localFit(cfg(k), tObs, xObs, yObs, x0, y0, dtMin, pTrue);
    cfg(k).rmsA     = rmsA;
    cfg(k).rmsB     = rmsB;
    cfg(k).RE       = reVec;
    cfg(k).exitFlag = flag;

    fprintf('avg time %.4gs   RMS-α %.3g   RMS-β %.3g\n', ...
            cfg(k).wallTime, cfg(k).rmsA, cfg(k).rmsB);
end
results = cfg;
%—— 4. draw ——------------------------------------------------------------
plotTask3Results(results);
end
```

```
function [rmsA,rmsB,reVec,flag] = localFit(c,tObs,xObs,yObs,...
                                           x0,y0,dtMin,pTrue)
flag = 0;


theta0 = [log(10); -0.03; -1e-3; log(8); 0.02; -4e-3];
mapP   = @(th)[exp(th(1)), th(2), th(3), -exp(th(4)), th(5), th(6)];

    function rss = obj(th)
        p = mapP(th);  a=p(1:3);  b=p(4:6);
        try
            if c.method=="ode45"
                [xs,ys] = predatorPreySolve('ode45', tObs,...
                    x0,y0,a,b, 'AbsTol',c.tol,'RelTol',c.tol);
            else
    tInt = tObs(1):c.h:tObs(end);
    if tInt(end) < tObs(end), tInt = [tInt, tObs(end)]; end
    [xt, yt] = predatorPreySolve(c.method, tInt, x0, y0, a, b);


    if any(isnan(xt)) || any(isnan(yt))
        rss = 1e20;
        return;
    end

    xs = interp1(tInt, xt, tObs, 'pchip', 'extrap');
    ys = interp1(tInt, yt, tObs, 'pchip', 'extrap');
end
        catch
            rss = 1e20; return;
        end
        rss = sum((xs-xObs).^2 + (ys-yObs).^2);
    end

opts = optimset('Display','off','MaxIter',1e3,'MaxFunEvals',1e3,...
                'TolX',1e-6,'TolFun',1e-6);
thetaHat = fminsearch(@obj,theta0,opts);


pEst = mapP(thetaHat);
reVec= abs(pEst - pTrue) ./ abs(pTrue);
```

```matlab
rmsA = sqrt(mean(reVec(1:3).^2));
rmsB = sqrt(mean(reVec(4:6).^2));
end



function plotTask3Results(cfg)
fprintf('\n>>> draw Task-3 curve …\n');
T = struct2table(cfg);

valid = T.rmsA < Inf & T.rmsB < Inf;
T = T(valid,:);

isO45 = startsWith(T.label,"ode45");
isG2  = startsWith(T.label,"gear2");
isRK4 = startsWith(T.label,"rk4");

%—— Fig-A  ode45 : Tol → precision  ---------------------------------------
--
figure('Name','ode45 – Tol vs Parameter Accuracy','Color','w');
tiledlayout(2,1,'TileSpacing','compact');

nexttile
semilogx(T.tol(isO45), T.rmsA(isO45),'bo-','LineWidth',1.6);
ylabel('RMS RE_{α}'); grid on;
title('ode45 :  Tol  → α-precision');

nexttile
semilogx(T.tol(isO45), T.rmsB(isO45),'rs-','LineWidth',1.6);
ylabel('RMS RE_{β}'); xlabel('Tol (RelTol = AbsTol)'); grid on;
title('ode45 :  Tol  → β-precision');

%—— Fig-B  step : h → precision  ----------------------------------------
figure('Name','Fixed-step – h vs Parameter Accuracy','Color','w');
tiledlayout(2,1,'TileSpacing','compact');

nexttile
loglog(T.h(isG2), T.rmsA(isG2),'r^-','LineWidth',1.6); hold on;
loglog(T.h(isRK4),T.rmsA(isRK4),'gs-','LineWidth',1.6);
ylabel('RMS RE_{α}'); grid on;
title('Fixed-step :  h  → α-precision');
legend('gear2','rk4','Location','best');

nexttile
```

```matlab
loglog(T.h(isG2), T.rmsB(isG2),'r^-','LineWidth',1.6); hold on;
loglog(T.h(isRK4),T.rmsB(isRK4),'gs-','LineWidth',1.6);
ylabel('RMS RE_{β}'); xlabel('Step size  h'); grid on;
title('Fixed-step :  h  → β-precision');

%—— Fig-C :  ode45   Tol → wall-time --------------------------------
figure('Name','ode45 – Tol vs Wall-clock Time','Color','w');
semilogx(T.tol(isO45), T.wallTime(isO45), 'bo-', 'LineWidth', 1.6);
ylabel('Wall-clock time [s]');
xlabel('Tol  (RelTol = AbsTol)');
title('ode45 :  Tol  →  speed');
grid on;

%—— Fig-D :  fixed-step  h → wall-time -------------------------------
figure('Name','Fixed-step – h vs Wall-clock Time','Color','w');
loglog(T.h(isG2 ),  T.wallTime(isG2 ),  'r^-', 'LineWidth', 1.6); hold on;
loglog(T.h(isRK4), T.wallTime(isRK4), 'gs-', 'LineWidth', 1.6);
ylabel('Wall-clock time [s]');
xlabel('Step size  h');
title('Fixed-step :  h  →  speed');
legend('gear2', 'rk4', 'Location', 'best');
grid on;

end




function pTrue = getTrueParams(fname)
switch fname
    case 'animals_1.csv'
        pTrue = [10 -0.03 -0.001  -8 0.02 -0.004];
    case 'animals_2.csv'
        pTrue = [11 -0.04 -0.002 -13 0.03 -0.003];
    case 'animals_3.csv'
        pTrue = [7  -0.01 -0.003 -15 0.01 -0.005];
    otherwise
        error('Unknown data file: %s', fname);
end
end
```