Project report

Joakim Axnér, Jonas Lorenz

Part 1: Aphids

At the beginning of summer, there are 10 aphids on the balcony. As a start, consider how the aphids would perform in the absence of tomatoes over a time period of two months. Each day ten percent of the aphids disappear.

Exercise 1.1. *What is the analytical solution? Does it make sense?*

Solution. The second Lotka-Volterra equation with $\delta = 0$ is given by

$$\frac{dy}{dt} = -\gamma y.$$

We know from analysis already that the solution in our case is given by

$$y \colon \mathbb{R} \to \mathbb{R}, \quad t \mapsto 10 \cdot e^{-\frac{t}{10}}.$$

This only really makes sense as an approximation. The number of aphids can only ever be a natural number. However, we have for example that $y(60) \approx 0.0025$. Clearly there cannot be a number of aphids between zero and one. ∎

Exercise 1.2. *Plot the analytical solution from time $t = 0$ to $t = 60$ days.*

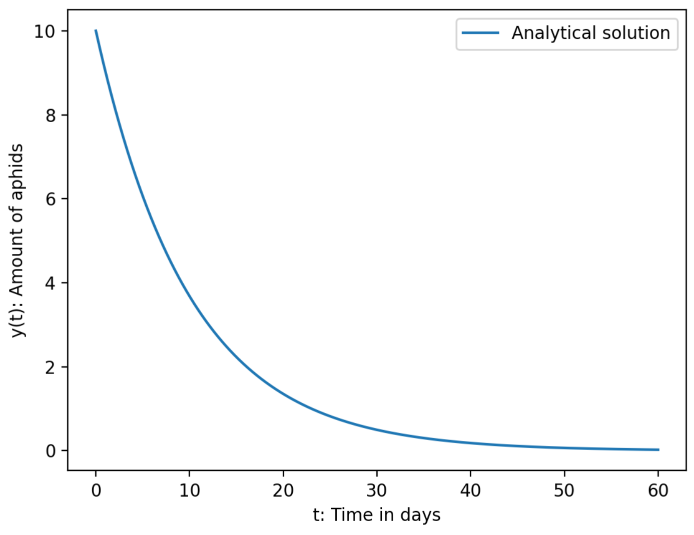

Figure 1: Analytical solution with $\delta = 0$

Exercise 1.3. *What is the theoretical maximum time step that yields a stable solution given a forward Euler discretisation?*

Solution. In our case, forward Euler computes the sequence $(y_n)_{n\in\mathbb{N}} \subset \mathbb{R}$ given by

$$y_{n+1} = y_n - \frac{t}{10} y_n, \quad y_0 = 10$$

where $t \in \mathbb{R}_{>0}$ is the step size we are using. The solution is stable if $(y_n)_n$ is bounded. Now we have

$$y_n = \left(1 - \frac{t}{10}\right)^n y_0$$

for all $n \in \mathbb{N}$. This is unbounded if and only if $\left|1 - \frac{t}{10}\right| > 1$. Therefore the solution is bounded if and only if $t \in (0, 20]$. Thus the maximal step size leading to a stable solution is $t = 20$. ■

Exercise 1.4. *Solve the problem numerically with forward Euler and plot it together with the analytical solution.*

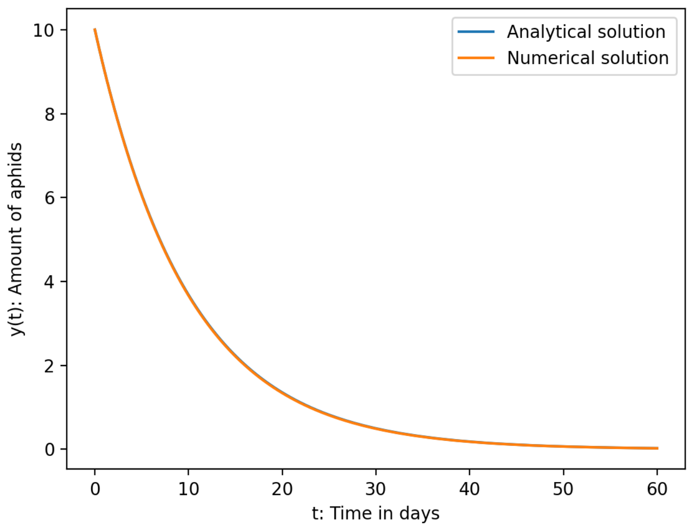

Figure 2: Analytical and numerical solution

Exercise 1.5. *Try a time step that is slightly larger than the maximum stable time step, for example by setting $\Delta t = 1.1\Delta t_{max}$. Is the numerical solution growing or decreasing, in other words stable? Does it look accurate? Include a plot.*

Solution. The following plot shows the numerical and analytical solutions on the interval $[0, 200]$.

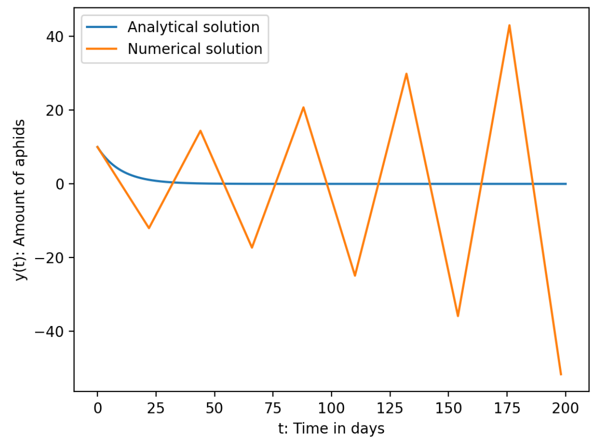

Figure 3: Step size slightly larger than Δt_{max}

We can clearly see that the solution oscillates and increases in terms of the absolute value. It is therefore unstable and is inaccurate, especially when comparing it to the analytical solution. ■

Exercise 1.6. *Try a time step that is slightly smaller than the maximum stable time step, for example by setting $\Delta t = 0.9\Delta t_{max}$. Is the numerical solution growing or decreasing, in other words stable? Does it look accurate? Include a plot.*

Solution. The following plot shows the numerical and analytical solution on the interval $[0, 200]$.

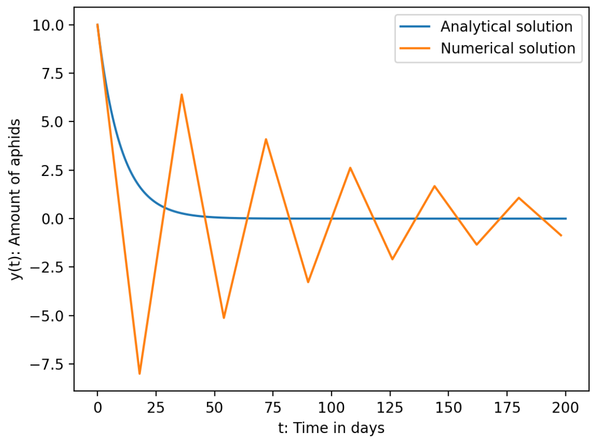

Figure 4: Step size slightly smaller than Δt_{max}

We can clearly see that the solution oscillates and decreases in terms of the absolute value. As expected, the solution is stable. However, it is not very accurate, especially when comparing with the analytical solution. ■

Exercise 1.7. *Decrease the time step until the oscillations disappear and the solution looks very similar to the analytical solution. Which time step did you choose? Include a plot.*

Solution. We knew from the previous exercises where we used $\Delta t = 0.1$ that this gave a very accurate solution. When decreasing the time steps we landed at $\Delta t = 0.1$ as the time step to go with. The follwing plot shows the plots for the different steps.

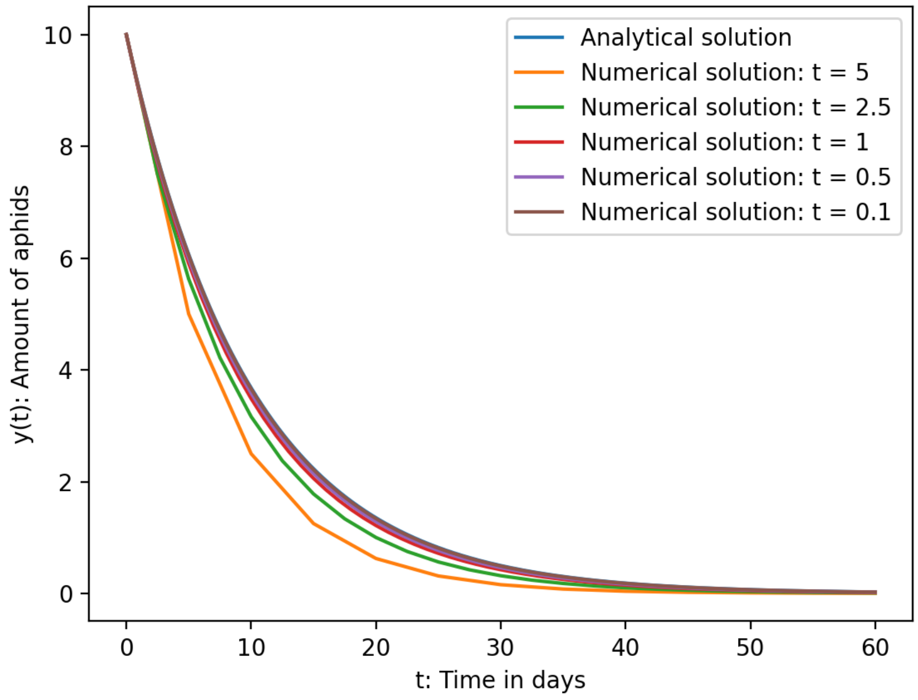

Figure 5: Plots for different time steps t

Here the analytical solution is not really visible anymore as the solution for $\Delta t = 0.1$ almost completely covers it. We can also see that all other solutions are visible, suggesting that they deviate from the analytical solution enough to deem them inaccurate. ■

Exercise 1.8. *Derive the exact maximum and minimum local truncation error.*

Solution. According to the book we have for the local truncation error that

$$|T_{k+1}| = \left| -\frac{1}{2} \cdot \frac{1}{10} e^{-\frac{\tau_k}{10}} \cdot \frac{1}{100} \right| = \frac{1}{2000} e^{-\frac{\tau_k}{10}}$$

where $\tau_k \in [t_k, t_{k+1}]$. We are interested in the maximum and minimum value of $|T_1|$. It is maximal for $\tau_0 = 0$. In this case $\frac{1}{2000}$ is the maximum truncation error. The minimum truncation error is given when $\tau_0 = \frac{1}{10}$ and is equal to $\frac{1}{2000}e^{-\frac{1}{100}}$. ■

Exercise 1.9. *Calculate the local truncation error numerically by comparing the analytical and numerical solution after one time step. Does the numerical local truncation error lie within the expected range?*

Solution. The numerically calculated local truncation error is given by `0.0004983374916811556`. This is slightly larger than the minimal truncation error from the previous exercise and thus lies in the expected range. ■

Exercise 1.10. *Take a twice as small time step. How did the truncation error change? Is it expected from theory.*

Solution. Analogously to *Exercise 1.8*, one gets a maximal truncation error of $\frac{1}{8000}$ and a minimal truncation error of $\frac{1}{8000}e^{-\frac{1}{200}}$. In this case we get a numerically calculated local truncation error of `0.00012479192682413043`. This is again slightly larger than the minimal theoretical truncation error. Further the truncation error decreased by about 75%. This can also be expected from theory as the local truncation error is given by

$$|T_{k+1}| = \left|-\frac{1}{2}y''(\tau_k)h^2\right|$$

where h is the time step and $\tau_k \in [t_k, t_{k+1}]$. If the time step gets reduced by half, the local truncation error will be about a fourth of the previous one. ■

Exercise 1.11. *How small a time step do you need to take to notice that the simulation is slow on your computer?*

Solution. The simulation becomes considerably slow when using the time step $\Delta t = 0.000001$. It then takes about 22 seconds. ■

Part 2: Aphids and Tomatoes

We have seen how the aphids behave on their own and will now add tomatoes to the system and model the interaction between aphids and tomatoes by solving the full system of the Lotka-Volterra equations.

Exercise 2.1. *Add the full Lotka-Volterra equations to your code. To check that it is working correctly, remodel the example from the section "A simple example" on the wikipedia page about the Lotka-Volterra equations.*

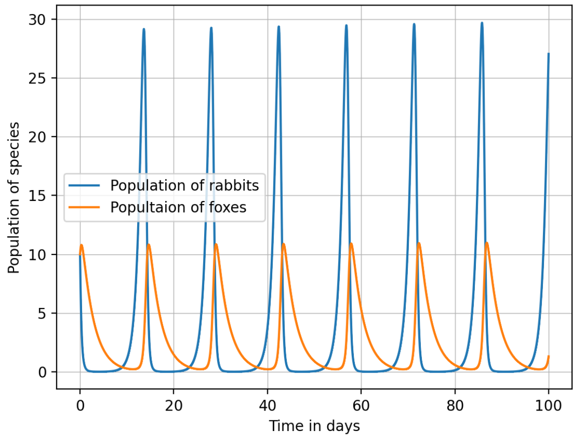

Figure 6: Wikipedia example

Exercise 2.2. *Now model the tomato and aphid populations where tomato leaves grow by $\alpha = 0.3$ every day while their death rate due to aphid munching is $\beta = 0.1$. Further aphids die at a rate of $\gamma = 0.1$ per day and increase by $\delta = 0.05$ due to eating tomato leaves.*
Solution. A time step of $\Delta t = 0.01$ works very well.

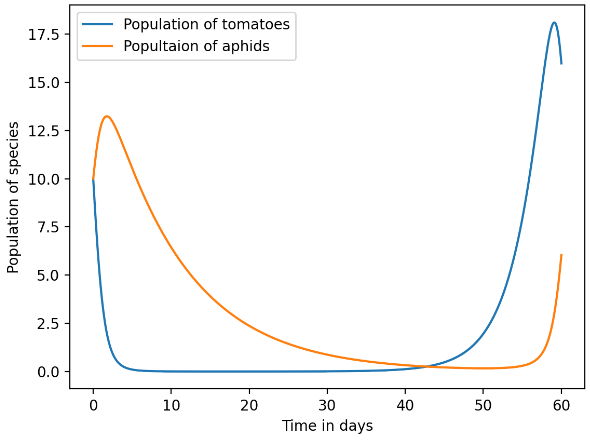

Figure 7: Simulation of tomatoes and aphids

■

Exercise 2.3. *The success rate of the tomato plants can be measured as the amount of tomato leaves integrated over the whole summer $S = \int x(t)\,dt$. You can compute the integral by using the* numpy*-function* `trapz`*, which is based on the numerical integration method called the trapezoid method.*

Solution. When using the numpy-function to calculate the success rate of the tomatoes we get $S =$ 110.23731163495529. One can for example interpret this number in the way that about $\frac{110}{60} \approx 1.83$ tomato leaves are alive each day on average during the two months. ■

Exercise 2.4. *How much does S decrease if you start with* 11 *aphids instead of* 10.

Solution. The success rate S decreases to 62.14385500602905. This is approximately a 43.63% decrease. ■

Exercise 2.5. *What would happen if the aphids mutate to a more resistant form so that their death rate is just slightly lower, namely* 0.09 *instead of* 0.1*? How much does S decrease?*

Solution. The success rate S decreases to 32.136281184058745. This is approximately a 70.85% decrease. We can thereby say that, especially in the short time span of two months, the aphids being more resistant is more detrimental to the tomato leaves than a mere increase of numbers. This can be explained due to the aphid numbers needing more time to hit their low point, causing the tomatoes to thrive. ■

Exercise 2.6. *What would happen if I get better at watering the tomato plants so that $\alpha = 0.35$ instead of 0.3? How much does S increase?*

Solution. In this case $S =$ 118.60788436790797. This constitutes an increase of roughly 7.59%. When considering the previous exercises, the numbers of the aphids seem to be more impactful side of the lever. ■

Exercise 2.7. *Why do you think I suggested using Euler forward in this project rather than Euler backward.*

Solution. First of all, Euler forward is a fairly simple and explicit method. This makes computation relatively simple. Given the short time interval and the good results Euler forward was able to achieve, for example in the first exercises, it simply seems to be enough for the task. The complications of an implicit method would not be worth dealing with considering the already good results. In our case Euler backward would result in the equations

$$x_{k+1} = x_k + h(\alpha x_{k+1} - \beta x_{k+1} y_{k+1}) \quad \text{and} \quad y_{k+1} = y_k + h(\delta x_{k+1} y_{k+1} - \gamma y_{k+1}).$$

Since their is no way to rewrite the equations such that we could explicitly compute, one would need to use costly methods such as fix point iterations. However this is far beyond the scope of what the relatively simple Lotka-Volterra equations demand. ■

Part 3: Adding Ladybugs

We would now like to get rid of the aphids, but without the use of synthetic pesticides by using biological control, which relies on introducing a hyper-predator that eats the pests. We will now simulate what happens if ten ladybugs are introduced into the system. The ladybugs grow with a rate of 0.1 and disappear or die with a rate of 0.5. The aphids further get eaten by ladybugs at a rate of 0.3.

Exercise 3.1. *Write down the new system of three equations.*

Solution. In this case we get the set of equations

$$\begin{cases} x' = 0.3x - 0.1xy \\ y' = 0.05xy - 0.1y - 0.3yz \\ z' = 0.1yz - 0.5z. \end{cases}$$

■

Exercise 3.2. *Edit the python script and plot the solution.*

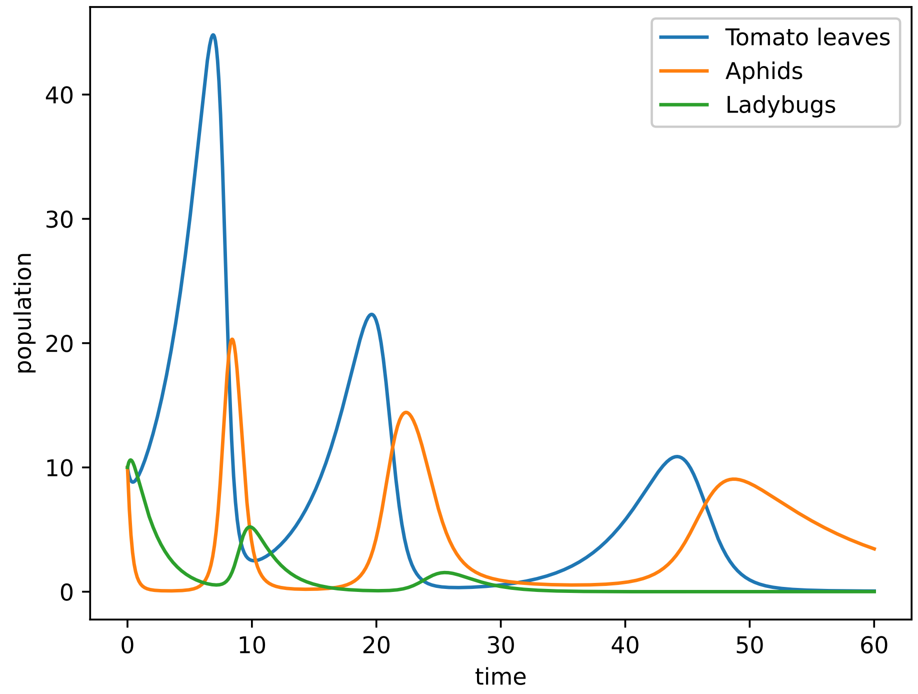

Figure 8: A Lotka-Volterra Three-Species model

Exercise 3.3. *How much does S increase?*

Solution. With ladybugs introduced, we get $S = 437.3172482800995$. This is a 269% increase compared to the old value $S = 118.60788436790797$. ■

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

def main(var=[0.3,0.1,0.1,0.05,0.1,0.3,0.5],mode=2,step=[0.1],\
         T=60,num=[10,10,10]):
    '''
    mode: 1 to 3 for respective exercise
    var: growth and decline parameters (4 or 7 parameters needed),
    their use can be found in the function "euler"
    step: step sizes to be plotted
    T: end time
    num: initial population numbers
    '''

    # boilerplate
    if mode not in [1, 2, 3]:
        return None

    if len(var) not in [4, 7]:
        return None

    if len(step) == 0:
        return None

    if len(num) != 3:
        return None

    if len(var) == 4 and mode == 3:
        return None

    if mode == 1:

        x = np.linspace(0, T, 20*T)
        y = 10*np.exp(-0.1*x)

        plt.figure()
        plt.plot(x, y, label='Analytical solution')

        for i in range(len(step)):

            dt = step[i]

            # forward Euler for given time step
            returned_population_numbers = euler([0,0,var[2],0,0,0,0], dt, T, num)
            time_vector = returned_population_numbers[0]
            aphid_numbers = returned_population_numbers[1]

            plt.plot(time_vector, aphid_numbers, '-o', markersize = 0.01, \
                     label=f'Numerical solution: t = {dt}')

            # calculate the truncation error
            tr_err = max(aphid_numbers[1] - 10*np.exp(-0.1*dt), -aphid_numbers[1]\
                          + 10*np.exp(-0.1*dt))
            trunc = f'The numerically computed local truncation error \
for a time step of t = {dt} is given by {tr_err}.'
            print(trunc)

        # plot the data
        plt.xlabel('t: Time in days')
        plt.ylabel('y(t): Amount of aphids')
        plt.legend()
        plt.show()
    elif mode == 2:

        plt.figure()

        for i in range(len(step)):

            dt = step[i]

            # forward Euler for given time step
            returned_population_numbers = euler(var[0:4] + [0, 0, 0], dt, T, num)
            time_vector = returned_population_numbers[0]
            aphid_numbers = returned_population_numbers[1]
            tomato_numbers = returned_population_numbers[2]

            # calculating the success rate of the tomatoes
            S = np.trapz(tomato_numbers, time_vector)
            s_rate = f'The success rate of the tomatoes under the given \
circumstances and with a time step of t = {dt} is S = {S}.'
            print(s_rate)

            plt.plot(time_vector, tomato_numbers, '-o', markersize = 0.01, \
                     label=f'Population of tomatoes: t = {dt}')
            plt.plot(time_vector, aphid_numbers, '-o', markersize = 0.01, \
                     label=f'Population of aphids: t = {dt}')

        # plot the data
        plt.xlabel('Time in days')
        plt.ylabel('Population of species')
        plt.legend()
        plt.show()

    else:

        plt.figure()

        for i in range(len(step)):

            dt = step[i]

            # forward Euler for given time step
            returned_population_numbers = euler(var, dt, T, num)
            time_vector = returned_population_numbers[0]
            aphid_numbers = returned_population_numbers[1]
            tomato_numbers = returned_population_numbers[2]
            ladybug_numbers = returned_population_numbers[3]

            # calculating the success rate of the tomatoes
            S = np.trapz(tomato_numbers, time_vector)
            s_rate = f'The success rate of the tomatoes under the given \
circumstances and with a time step of t = {dt} is S = {S}.'
            print(s_rate)

            plt.plot(time_vector, tomato_numbers, '-o', markersize = 0.01, \
                     label=f'Population of tomatoes: t = {dt}')
            plt.plot(time_vector, aphid_numbers, '-o', markersize = 0.01, \
                     label=f'Population of aphids: t = {dt}')
            plt.plot(time_vector, ladybug_numbers, '-o', markersize = 0.01, \
                     label=f'Population of ladybugs: t = {dt}')

        # plot the data
        plt.xlabel('Time in days')
        plt.ylabel('Population of species')
        plt.legend()
        plt.show()
```

```
def euler(var,dt,T,num):

    # initial time and end time
    t = 0
    T = 60

    # assigning initial population numbers
    x_0 = num[0]
    y_0 = num[1]
    z_0 = num[2]

    # lists for plotting
    tomato_numbers = []
    tomato_numbers.append(x_0)
    aphid_numbers = []
    aphid_numbers.append(y_0)
    ladybug_numbers = []
    ladybug_numbers.append(z_0)
    time_vector = []
    time_vector.append(t)

    # constants, use of population growth and decline parameters
    tomato_growth = var[0]
    tomato_decline = var[1]
    aphid_decline = var[2]
    aphid_growth = var[3]
    ladybug_growth = var[4]
    aphid_being_eaten = var[5]
    ladybug_decline = var[6]

    # forward Euler for given time step
    while t < T:

        x_t = x_0
        y_t = y_0
        z_t = z_0
        x_0 = x_t + dt*(tomato_growth*x_t - tomato_decline*x_t*y_t)
        y_0 = y_t + dt*(aphid_growth*x_t*y_t - aphid_decline*y_t - aphid_being_eaten*y_t*z_t)
        z_0 = z_t + dt*(ladybug_growth*z_t*y_t - ladybug_decline*z_t)
        tomato_numbers.append(x_0)
        aphid_numbers.append(y_0)
        ladybug_numbers.append(z_0)
        t = t + dt
        time_vector.append(t)

    return (time_vector, aphid_numbers, tomato_numbers, ladybug_numbers)
```