Student names: Honigmann Simon, Munier Louis & Plett Palomar Kilian Asterio

Instructions: Update this file (or recreate a similar one, e.g. in Word) to prepare your answers to the questions. Feel free to add text, equations and figures as needed. Hand-written notes, e.g. for the development of equations, can also be included e.g. as pictures (from your cell phone or from a scanner). ***This lab is not graded. However, the lab exercises are meant as a way to familiarise with dynamical systems and to study them using Python to prepare you for the final project.*** *This file does not need to be submitted and is provided for your own benefit. The graded exercises will have a similar format.*

In this exercise, you will familiarise with ODE integration methods, how to plot results and study integration error. The file `lab#.py` *is provided to run all exercises in Python. Each* `exercise#.py` *can be run to run an exercise individually. The list of exercises and their dependencies are shown in Figure 1. When a file is run, message logs will be printed to indicate information such as what is currently being run and and what is left to be implemented. All warning messages are only present to guide you in the implementation, and can be deleted whenever the corresponding code has been implemented correctly.*

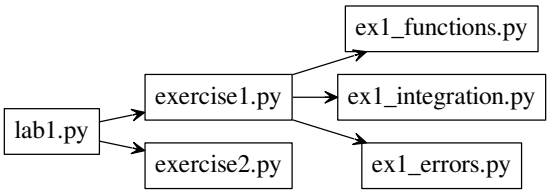

Figure 1: Exercise files dependencies. In this lab, you will be modifying `exercise1.py`, `ex1_functions.py`, `ex1_integration.py`,`ex1_errors.py` *and* `exercise2.py`. *It is recommended to check out* `exercise1.py` *before looking into the other* `ex1_*.py` *files.*

Installation of dependencies

In order to run the exercises for Lab1 make sure to update the dependencies. You can do this by executing the following commands:

- Open the terminal
- Navigate to the exercise repository (Make sure you are in the root of the folder)
- Execute,
  - $ git pull
  - $ pip install -r requirements.txt

Question 1: Numerical integration

1.a Compute the analytical solution x(t) for the following linear dynamical system. Provide here the calculation steps, then implement the solution in `ex1_functions.py::analytic_function()` and run `exercise1.py` to plot the result.

$$\dot{x} = 2 \cdot (5 - x), \quad x(t = 0) = 1 \tag{1}$$

Since this is a separable differential equation it can be directly solve.

$$\begin{aligned}
\frac{dx}{dt} &= 2 \cdot (5 - x) \\
\frac{1}{5 - x} dx &= 2dt \\
\int_{x_0}^{x} \frac{1}{5 - x} dx &= \int_{t_0}^{t} 2dt \\
-\ln(5 - x) + \ln(5 - x_0) &= 2t - 2t_0 + C
\end{aligned}$$

Since the condition at time $t = 0$ is $x = 1$ there is $C = 0$ and :

$$x(t) = 5 - 4e^{-2t}$$

The result is shown below.

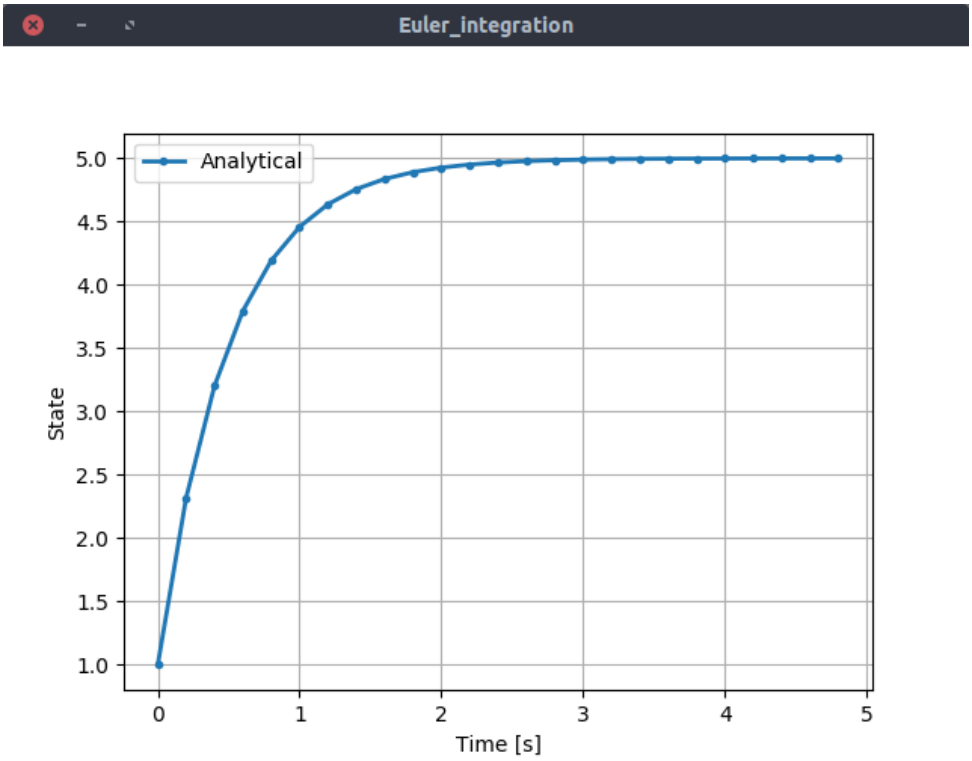

Figure 2: Euler analytical result.

1.b In some cases, an ODE system may not have an analytical solution or it may be difficult to compute. Implement Euler integration in `ex1_integration.py::euler_integrate()`, then run `exercise1.py` again to compare the solution of `euler_integrate()` (with 0.2 timestep) to the analytical solution obtained previously and include a figure of the result here. Make sure to also implement `ex1_functions.py::function()` so that the code may be run correctly.
As a code template, check out `ex1_integration.py::euler_example()`.

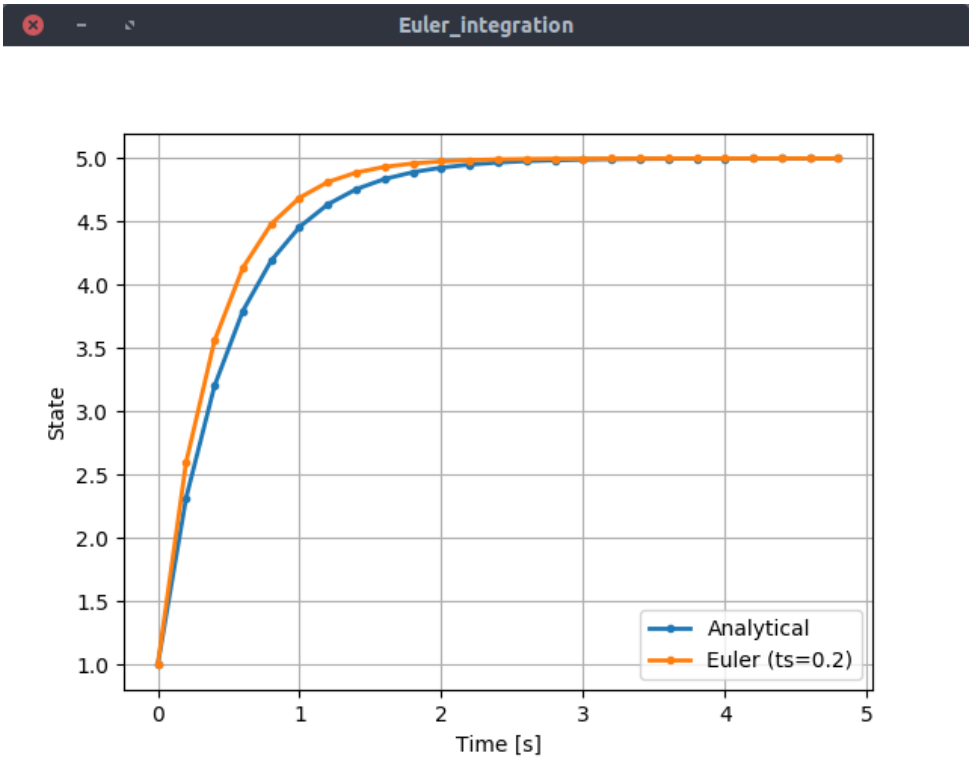

Figure 3: Euler integration results.

As shown in figure 3 the euler integration method gives a good approximation of the real result. By computing the average abosulute error using :

$$err = \frac{1}{N}\sum_{i=1}^{N}|f_{method}(i) - f_{analytical}(i)|$$

with N being the number of state samples, f_{method} is the euler one and $f_{analytical}$ is the analytical function.

1.c Various efficient libraries are available to facilitate ODE integration. Compare the Euler method with Lsoda (`ex1_integration.py::ode_intgrate()`) and Runge-Kutta 4th order (`ex1_integration.py::ode_intgrate_rk()`) integration methods by completing the corresponding functions using the scipy library in Python. See `exercise1.py::exercise1()` for the function calls. Provide the error values (there are multiple ways you could do this, choose an appropriate method and explain why), number of time steps, and include a figure comparing the integration methods.

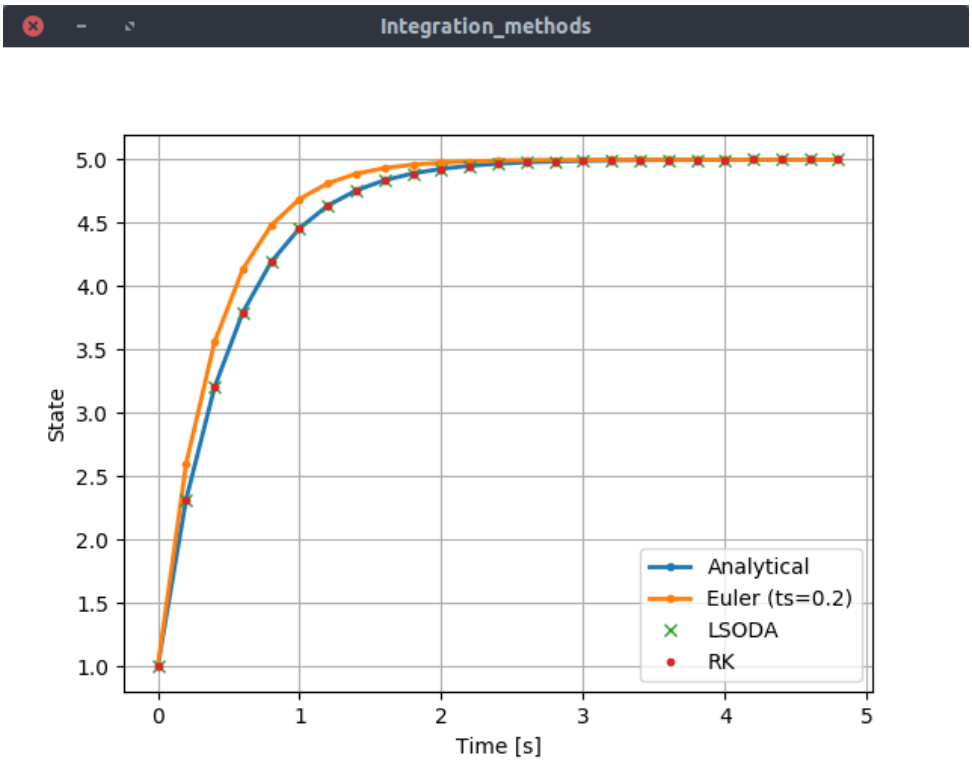

Figure 4: Compare different solver results.

Errors results :

Table 1: Error comparison between each methods.

Method	Average error	Error max	# timesteps
Euler	~ 0.085	~ 0.357	25
LSODA	$\sim 1.17e-8$	$\sim 5.79e-8$	Adaptative
Runge-Kutta	$\sim 6.41e-8$	$\sim 1.43e-7$	25

1.d As mentioned in the course, the comparison of question 1.c is not fair for the Euler method. Briefly say why. Choose another number of time steps for the Euler method that is fairer when compared to Runge-Kutta. Provide the error values, number of time steps, and include a figure comparing the two integration methods. Briefly discuss which integration method is best.

The Runge-Kutta integration method computes the derivatives 4 times at each iteration. A fair comparison is therefore to make 4 Euler steps for each Runge-Kutta step. So for the next comparison, the timestep for Euler method is of 0.05 s.

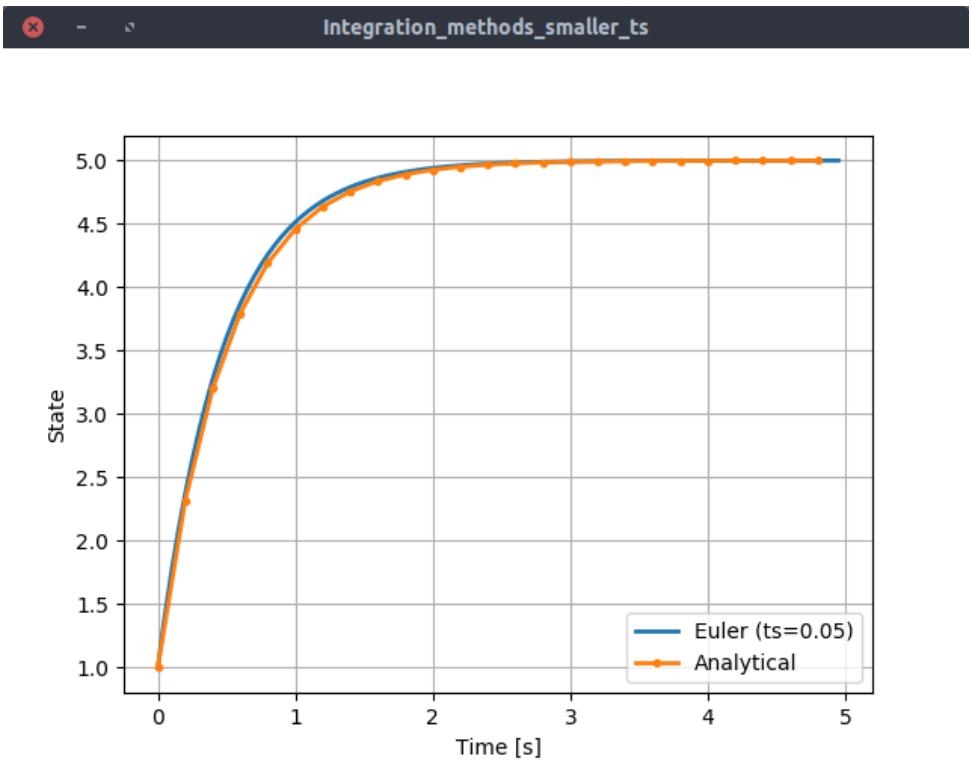

Figure 5: Compare different solver results.

Errors results :

Table 2: Error comparison between each methods.

Method	**Average error**	**Error max**	**# timesteps**
Euler	~ 0.085	~ 0.357	25
LSODA	$\sim 1.17e-8$	$\sim 5.79e-8$	Adaptative
Runge-Kutta	$\sim 6.41e-8$	$\sim 1.43e-7$	25 (100)
Euler small	~ 0.020	~ 0.077	100

The error is still much smaller using Runge-Kutta time steps (ode45), even if approximately the same number of derivatives are computed with both methods (100 times for Euler, $25 \cdot 4 = 100$ times for Runge-Kutta).

1.e Test the role of the step size by plotting the integration error as a function of step size. You can use `ex1_errors.py::compute_error()` to do this by completing the code in `ex1_errors.py::error()`. How accurate is the solution compared to the analytical solution for different step sizes? Include here a graph showing the error against the step size. Explain which error measure you used (there are several options)

Question 2: Stability analysis

2.a Find the fixed points of the following linear dynamical system, and analyze their stability (briefly describe the calculation steps).

$$\dot{x} = Ax, \qquad A = \begin{pmatrix} 1 & 4 \\ -4 & -2 \end{pmatrix} \tag{2}$$

2.b Perform numerical integration from different initial conditions to verify the stability properties. See `exercise2.py::exercise2()` for implementation. Include some figures with these different time evolutions and their corresponding phase portrait and explain their roles.

2.c Change one value in matrix A such that the time evolution becomes periodic for some initial conditions. Say which value and include a time evolution figure.