Student names: . . . (please update)

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.

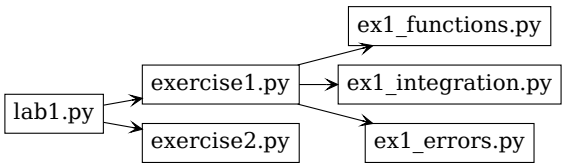

Figure 1: Exercise files dependencies. In this lab, you will be modifying `exercise1.py`, `ex1_functions.py`, `ex1_integration.py`,`ex1_errors.py` and `exercise2.py`. It is recommended to check out `exercise1.py` before looking into the other `ex1_*.py` files.

Running the exercises

In order to run the exercises for Lab1 make sure to update your git packages. 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`

NOTE : Make sure to activate your virtualenv before running the exercise codes.

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.

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

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()`.

1.c Various efficient libraries are available to facilitate ODE integration, such as Scipy. In this exercise first implement your own Runge-Kutta 4th order method (in `ex1_integration.py::ode_intgrate_rk()`). Then use scipy.odeint Lsoda method (in `ex1_integration.py::ode_intgrate()`) and scipy.ode adaptive Runge-Kutta Dopri method of order 4(5) (in `ex1_integration.py::ode_intgrate_dopri()`) and solve the ode (overimpose plots of the solution using different methods and markers). Compare all these methods (euler, runge-kutta, lsoda and dopri) by defining an error function (there are multiple ways you could do this, choose an appropriate method and explain why) and number of time steps.

1.d The error comparison between Euler and Runge-Kutta 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.

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). The error of the adaptive solvers can be reduced by setting a smaller value of the relative tolerance of the solver (rtol). Reduce rtol and check the solver's accuracy.

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.

2.d Compute the eigenvalues and eigenvectors of the following system. What can you say about the stability of its fixed point? What is the meaning of its eigenvectors? Try to plot the flow of the system to get an intuition of its behavior.

$$\dot{x} = Ax, \qquad A = \begin{pmatrix} 1 & 1 \\ 4 & -2 \end{pmatrix} \tag{3}$$