Freie Universität Berlin
Prof. Dr. Max von Kleist
Dr. Alexia Raharinirina

4. Assignment
Complex Systems for Bioinformaticians
SS 2025

Deadline: May 13, 12:00 (**before** the lecture)

The homework should be worked out individually, or in groups of 2 students. Pen & paper exercises should be handed at the designated deadline. Each solution sheet must contain the names and 'Matrikulationnummer' of all group members and the name of the group. The name of the group must include the last names of the group members, in alphabetic order, e.g. "AlbertRamakrishnanRastapopoulos", for group members Mandy Albert, Mike Ramakrishnan, and Marcus Rastapopoulos. Please staple all sheets.
Programming exercises must be submitted via Whiteboard.

Homework 1 (Analytical Solution (pen & paper), 3 points)
You are given the following model

$$\frac{\mathrm{d}}{\mathrm{d}t}x = k \cdot t \cdot x + t^2 \tag{1}$$

with $k > 0, t \geq 0$. Analytically solve this model using the method of 'integrating factors' with initial condition: $x_0(t_0) > 0$ and $t_0 = 0$.

Homework 2 (Programming & Analysis, 2+2+1+2 points)
You are given the following model to describe the pharmacokinetics (concentration-time profile) of a pharmaceutic drug in the 'dosing compartment' x_0 as well as in the 'bloodstream' x_1 after a single dose intake:

$$x_0 \xrightarrow{r_0} x_1 \ , \ x_1 \xrightarrow{r_1} \varnothing \tag{2}$$

with reaction rates

$$r_0 = x_0 \cdot k_a \ , \ r_1 = x_1 \cdot k_e. \tag{3}$$
$$\tag{4}$$

where k_a and k_e are parameters describing the uptake and elimination of the drug from the body. In our example, let $k_a = 0.5$ and $k_e = 0.3$.
a) (**to be uploaded via Whiteboard**) Write a program implementing this model. Numerically integrate the resulting ODEs with the implicit Euler method using a step size $\Delta t = 0.5$ up to time $t_{final} = 24$. The program reads the input file ("Input.txt") provided on Whiteboard, where the two numbers in the input file denote the initial condition of the system, i.e. the initial values of the state variables $x_0(t_0), x_1(t_0)$. After simulation, write the time and the values of x_1 into a file "Task2aTraj.txt". The output text-file should be in the comma-separated text format using two digits after the comma (format '%1.2f'), e.g.

$$0.00, 0.50, 1.00, 1.50, ... \tag{5}$$
$$0.00, 50.00, 80.00, 96.12, ... \tag{6}$$

where the first row denotes the times and the second row the corresponding states of x_1. Call this program "Ex2a.py" and submit it via the Whiteboard system.

Hint: (i) Solve the equation for x_0, (ii) plug it into the ODE for x_1. (iii) Write down the implicit Euler method, (iv) Re-arrange that equation to get an evolution equation for $x_1(t_{n+1})$ as a function of $x_1(t_n)$.
Careful: Chat-GPT will NOT be able to help with this one ...!
b) (**to be uploaded via Whiteboard**) Numerically integrate the ODEs with the 4th-order Runge-Kutta scheme using a step size $\Delta t = 0.5$ up to time $t_{final} = 24$. The program reads the input file ("Input.txt") provided on Whiteboard, where the two numbers in the input file denote the initial condition of the system, i.e. the initial values of the state variables $x_0(t_0), x_1(t_0)$. After simulation, write the time and the values of x_1 into a file "Task2bTraj.txt". The output text-file should be in the comma-separated text format using two digits after the comma (format '%1.2f') as before. Call this program "Ex2b.py" and submit it via the Whiteboard system.
c) (**plot and discuss on paper**) Extend your program above by solving the ODEs with the built-in ODE-solver `scipy.integrate.solve_ivp` using 'RK45' for integration. Use the solver, such that it provides the solution of the ODE at the same times as your numerical integration methods (e.g. $t_i \in$(0.5, 1.0, 1.5, ..., 24); in Python: t_points = np.arange(0,t_final+1e-6,stepsize)). For the ODE solver you can use:
from scipy.integrate import solve_ivp
sol = solve_ivp(RHS,[0, t_final],X0,method='RK45',vectorized=True, t_eval = list(t_points))
Make a plot that superimposes the solutions from a) and b) with the in-built ODE-solver solution (a bit like Fig. 1), print the plot, hand it in and discuss (on paper):

- Describe in your words: Why does 'the wrong' trajectory over- and undershoot the apparent 'correct' trajectory?

d) (**plot and discuss on paper**) Modify your programs above in the following way: Use different time steps $\Delta t = 0.05, 0.1, 0.5, 1, 2$.
(i) Compute the *average* error between the solution with python's in-built solver $y(t)$ vs. your numerical solution(s) from a-b) with different time-steps. I.e. compute,

$$\bar{\varepsilon}(\Delta t) = \frac{1}{N_t}\sum_{i=0}^{N_t} |x_1(i \cdot \Delta t) - y(t_i)|, \tag{7}$$

where $N_t = t_{final}/(\Delta t)$ denotes the number of time steps in your simulation and $x_1(t_i)$ with $t_i \in (0, \Delta t, 2\cdot\Delta t, \ldots, N_t \cdot \Delta t)$ denotes your numerical solution with the method used in a) or b). Plot the error vs. the step size. How does the error behave with regards to the step size?

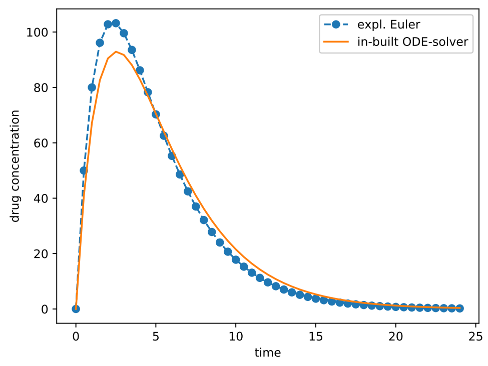

Figure 1: An example how the result from homework 1c) could look like (note that this just shows one solver solution [instead of two as in 2a & 2b.]).

Good luck!