

Exercise 1: Initial Value Problems

Prof. Dr. Moritz Diehl and teaching assistant(s)

In this exercise we will have a first look at initial value problems (IVPs). We will analyze some of the conditions necessary for the existence and uniqueness of solutions to IVPs and we will implement and compare explicit integration methods for the solution of ordinary differential equations.

I Existence and uniqueness of solutions to IVPs: Lipschitz-continuity

In order to guarantee that a solution to an IVP exists and it is unique, we can rely on Theorem 1.2. Among the conditions that have to be satisfied for the results of the theorem to hold, the function $f(x(t), t)$ describing the dynamics of the system, has to be Lipschitz continuous with respect to x .

1. Indicate (without proof) which of the following functions are globally Lipschitz (GL), locally Lipschitz (LL) or not locally Lipschitz (NLL):

	(GL)	(LL)	(NLL)
(a) x^2	☐	☐	☐
(b) $ x ^{\frac{1}{2}}$	☐	☐	☐
(c) $\text{sign}(x) x ^{\frac{1}{2}}$	☐	☐	☐
(d) $\ x\ _2^2$	☐	☐	☐
(e) $\ x\ _2$	☐	☐	☐
(f) $\ x\ _2^{\frac{1}{2}}$	☐	☐	☐
(g) $\ Ax\ _2$	☐	☐	☐
(h) $\sin x^T Ax$	☐	☐	☐
(i) $\sin \ x\ _2$	☐	☐	☐

II Explicit integrators

In Section 1.2 we will introduce integration schemes for ordinary differential equations (ODEs). These schemes allow one to compute a numerical approximation of the true solution to a initial value problem (IVP) of the form

$$\begin{aligned}\dot{x}(t) &= f(x(t)), \quad t \in [0, T] \\ x(0) &= x_0.\end{aligned}\tag{1}$$

In this exercise we will compare three different explicit integration schemes.

1. The `scipy.integrate.solve_ivp` function provides a user-friendly interface to an implementation of the Runge-Kutta integrator of type Dormand-Prince. It is suitable for the integration of non-stiff ODEs of the form $\dot{x} = f(t, x)$ and it can provide reasonable accuracy in reasonable computation times in many practical cases. Complete the template to call `solve_ivp` in order to obtain a numerical approximation of the solution to the IVP for a damped oscillator

$$\begin{aligned}x(0) &= x_0, \\ \dot{x}_1(t) &= x_2(t), \\ \dot{x}_2(t) &= -0.2x_2 - x_1(t),\end{aligned}\tag{2}$$

with $x_0 = (1, 0)^\top$. As output time grid, use the grid defined by $t = kT_s$, where $k = 0, \dots, 20$ and $T_s = 0.5$. Plot the obtained trajectories.

2. We will now implement two other explicit integration schemes. First, implement the explicit Euler scheme

$$x_{n+1} = x_n + hf(x_n).\tag{3}$$

Second, implement the explicit Runge-Kutta scheme of order four (RK4):

$$\begin{aligned}k_1 &= f(x_n) & k_2 &= f(x_n + \frac{h}{2}k_1) & k_3 &= f(x_n + \frac{h}{2}k_2) & k_4 &= f(x_n + hk_3) \\ x_{n+1} &= x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).\end{aligned}\tag{4}$$

Using the two schemes ($h = 0.5$ for the RK4 scheme and $h = 0.125$ for explicit Euler), compute numerical approximations of the solution to the IVP in (2) and plot the obtained trajectories in the same figure as (a).

3. Implement an RK4 integrator for our IVP as a CasADi function and use it to perform the numerical simulation.

Hint: you can build an expression from CasADi variables and then use it to declare a function, as in the following example. Use `.full()` to convert the CasADi DM arrays resulting from the evaluation of a CasADi function to a NumPy ndarray.

```
import casadi as ca
x = ca.MX.sym('x', 2, 1)
expr = ca.sin(x[0]) * x[1]
f = ca.Function('f', [x], [expr])
y = f([ca.pi/2, 2])
y2 = y.full()
```

III The Lorenz attractor

We will now use the integrators implemented to simulate some interesting systems.

1. In 1963, Edward Lorenz developed a simplified mathematical model for atmospheric convection. The model is a system of three ordinary differential equations now known as the Lorenz equations:

$$\begin{aligned}\dot{x}_1 &= \sigma(x_2 - x_1) \\ \dot{x}_2 &= x_1(\rho - x_3) - x_2 \\ \dot{x}_3 &= x_1x_2 - \beta x_3\end{aligned}\tag{5}$$

with $\rho = 28$, $\sigma = 10$ and $\beta = \frac{8}{3}$.

- (a) Does the Lorenz attractor satisfy the conditions necessary for the Picard-Lindelöf theorem to be applicable?
- (b) Using the RK4 integrator implemented before, simulate the system for $x_0 = [1, 0, 0]^T$ and $t \in [0, 100]$ with step-size $h = 0.01$. Plot the resulting trajectory in 3D. You can use the provided template.