Numerical Solutions of Ordinary Differential Equations

Perla Mallouk

perla.mallouk@math.cnrs.fr

MAP5, Université Paris Cité

September 30, 2025

Numerical approximation: general concepts

Consider any first-order ODE in explicit form, with an initial condition:

$$\begin{cases} x'(t) = f(t, x), \\ x(t_0) = x_0. \end{cases}$$

Main idea of the Euler scheme: approximate $x'(t)$ by a difference quotient:

$$x'(t) \approx \frac{x(t + \Delta t) - x(t)}{\Delta t},$$

which yields

$$x(t + \Delta t) \approx x(t) + \Delta t \; f(t, x).$$

The Forward Euler method (FE)

Set $t_n = t_0 + n\,\Delta t$ and write

$$\begin{cases} x_{n+1} = x_n + \Delta t\; f(t_n, x_n), \quad \text{for} \quad n = 0, 1, \ldots, \\ x_0 \quad \text{given by the initial condition.} \end{cases}$$

Consequence: starting from an initial position x_0 at time t_0, and fixing a time step $\Delta t > 0$, one can compute the approximations

$$x_1 \approx x(t_0 + \Delta t), \quad x_2 \approx x(t_0 + 2\,\Delta t), \quad \ldots$$

iteratively and in an **explicit** manner.

The Backward Euler method (BE)

Set $t_n = t_0 + n\,\Delta t$ and write

$$\begin{cases} x_{n+1} = x_n + \Delta t\; f(t_{n+1}, x_{n+1}), \quad \text{for} \quad n = 0, 1, \ldots, \\ x_0 \quad \text{given by the initial condition.} \end{cases}$$

Consequence: starting from an initial position x_0 at time t_0, and fixing a time step $\Delta t > 0$, one can compute the approximations

$$x_1 \approx x(t_0 + \Delta t), \quad x_2 \approx x(t_0 + 2\,\Delta t), \quad \ldots$$

iteratively, but only after solving an **implicit** equation.

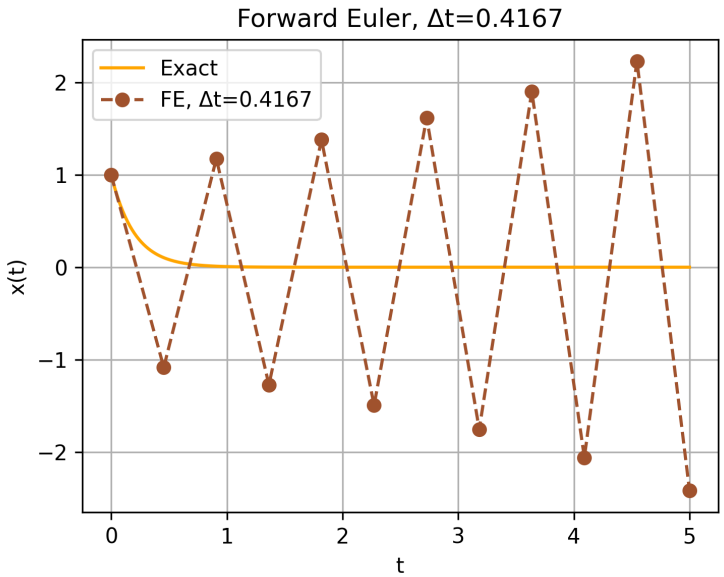
A simple example

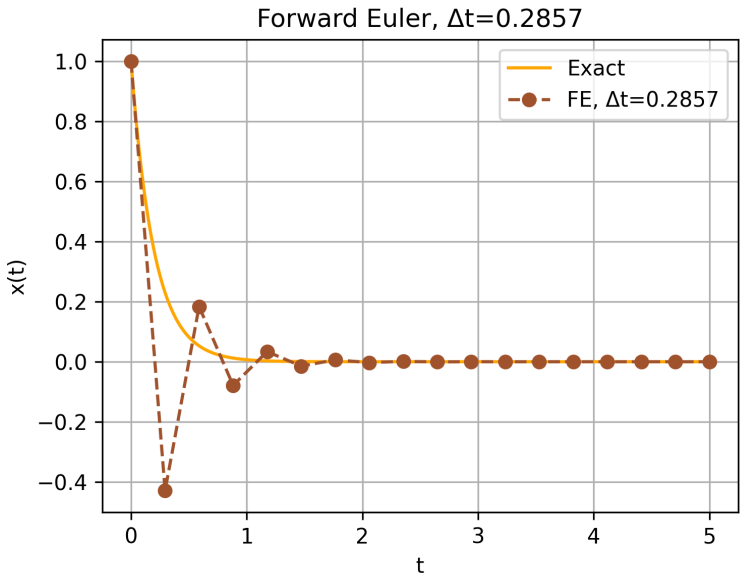
Consider the following ODE:

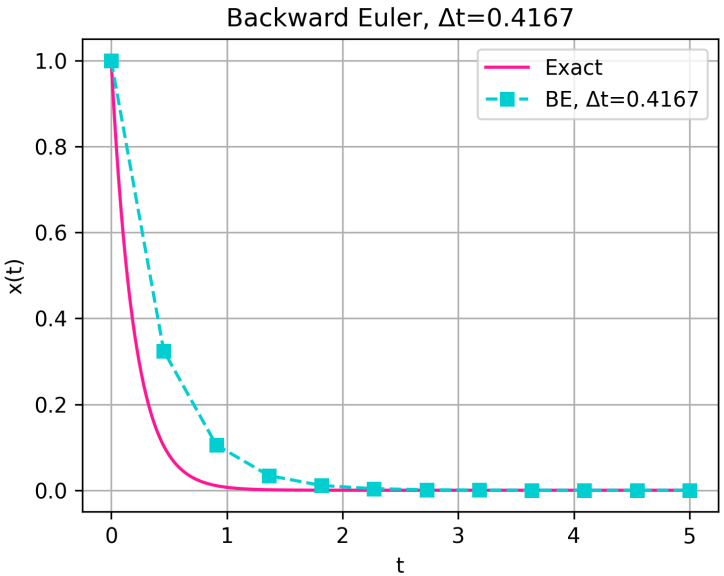
$$\begin{cases} x'(t) = -\lambda x(t), \\ x(0) = 1. \end{cases}$$

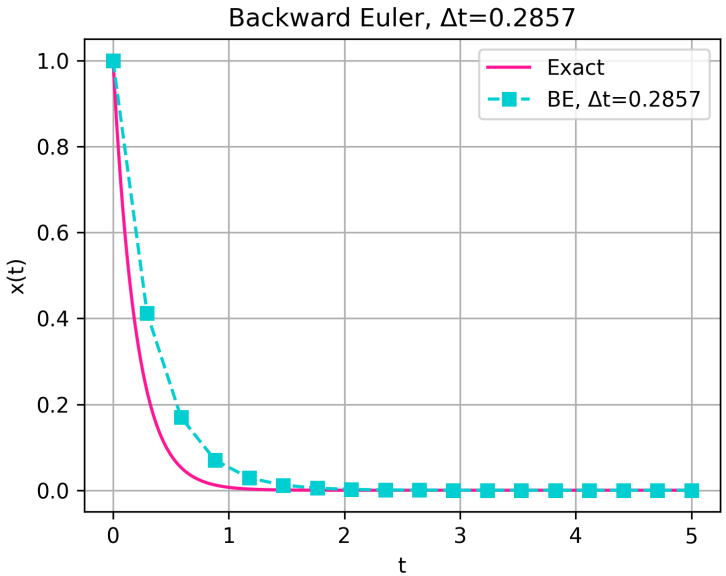
We will solve this equation with both methods: FE and BE, for $\lambda = 5$ and for two different time steps: $\Delta t = 0.4167$ and $\Delta t = 0.2857$.

A simple example









Comments

Questions:

- What are the **similarities** and the **differences** between the previous examples?
- What are the **difficulties**?

Remarks:

- **FE:** the numerical solution oscillates — need to decrease Δt;
  BE: no oscillations, better stability, but still not very accurate.
- **FE:** x_{n+1} depends exclusively on the value x_n previously computed;
  $\Rightarrow$ **explicit** method.
  BE: x_{n+1} depends also on itself through the value $f(t_{n+1}, x_{n+1})$;
  $\Rightarrow$ **implicit** method, more costly.
- To achieve a higher order of accuracy: use more sophisticated schemes, such as **the Runge–Kutta methods**.

Python built-in functions for ODE solving

- **Runge–Kutta methods** (RK) in Python: `solve_ivp` with method options like `"RK45"`, `"RK23"`, *etc.*
  `"RK45"` performs well with most ODE problems and is generally a good first choice of solver.
  However, solvers like `"RK45"` might be unable to efficiently solve ODEs with stiffness or widely varying scales, and can be extremely slow in those cases.

- **Alternative approach for stiff ODEs** in Python: use `"Radau"` or `"BDF"` in `solve_ivp`.
  `"BDF"` or `"Radau"` are better choices for stiff ODE problems or when evaluation is difficult.

Python ODE solver documentation

- **Python basic tutorial** available on Moodle.
- More details on how to choose a **SciPy ODE solver**: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html`
- Overview of **integration methods** available in `solve_ivp`: `https://docs.scipy.org/doc/scipy/reference/integrate.html`

Matlab/Octave built-in functions for ODE solving

- **Runge–Kutta methods** (RK) in Matlab/Octave: `ode45`, `ode23`, *etc.*
  `ode45` performs well with most ODE problems and is generally a good first choice of solver.
  However, solvers like `ode45` might be unable to efficiently solve ODEs with stiffness or widely varying scales, and can be extremely slow in those cases.

- **Alternative approach for stiff ODEs** in Matlab/Octave: use `ode15s`.
  `ode15s` is a better choice for stiff ODE problems or when evaluation is difficult.

Matlab/Octave ODE solver documentation

- **Matlab/Octave basic tutorial** available on Moodle.
- More details on how to choose a **Matlab ODE solver**:
  `https://www.mathworks.com/help/matlab/math/choose-an-ode-solver.html`
- Description of **Matlab-compatible solvers** available in Octave:
  `https://octave.org/doc/v4.4.1/Matlab_002dcompatible-solvers.html`

Practice using Python

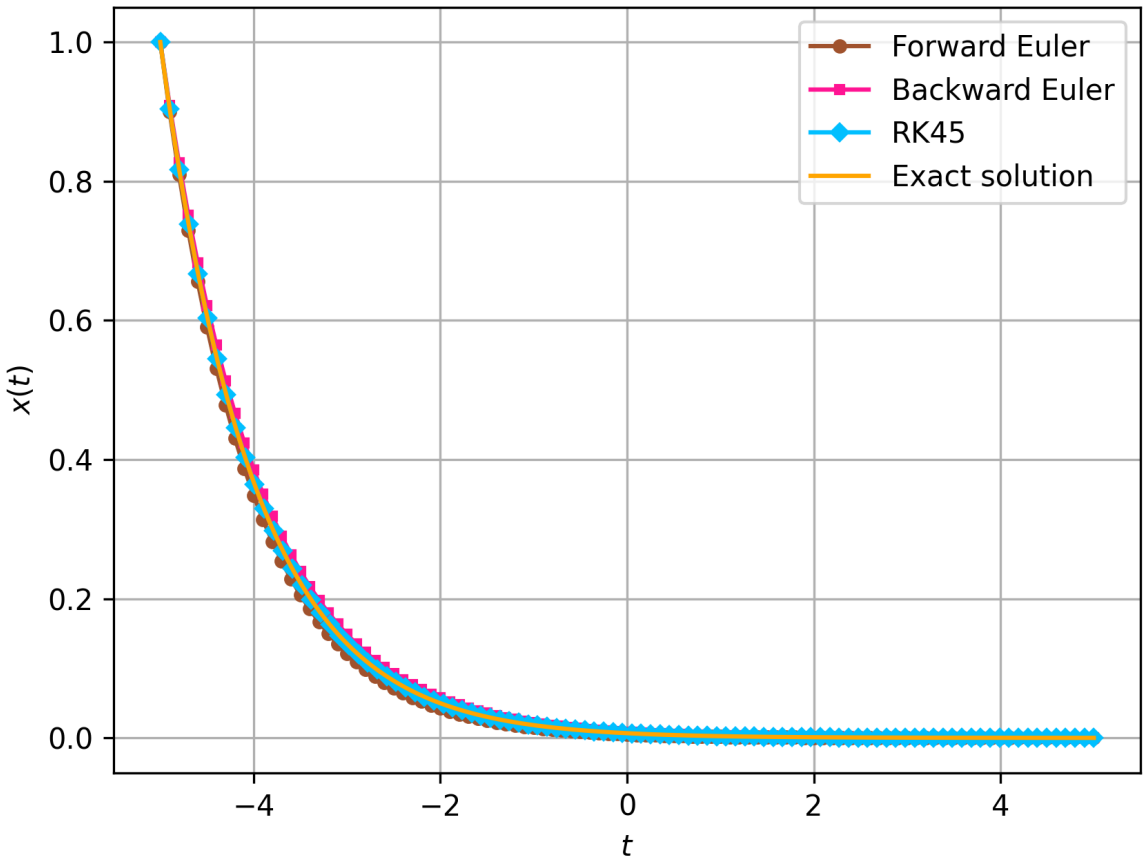
- Write a Python code using `NumPy` to solve numerically the equation $x' = -x$, with initial condition $x(-5) = 1$, by performing 100 steps of the Euler scheme with stepsize $\Delta t = 0.1$. Draw the graph of this numerical solution. On the same graph, also plot the exact solution and compare.
- Use the function `solve_ivp` with `method='RK45'` to solve the same equation with the same initial condition, over the range $-5 \leq t \leq 5$. On the same graph, draw this new numerical solution and comment.
- Use `solve_ivp` to solve numerically the equation $x' = t\cos(x)$ for the initial condition $x(0) = 1$ over the range $0 \leq t \leq 10$.
- Still for the equation $x' = t\cos(x)$, draw on the same graph the numerical solutions obtained with the different initial conditions $x(0) = 0$, $x(0) = 0.1$, $x(0) = 0.2$, ... $x(0) = 1$.

Practice using Matlab/Octave

- Write a Matlab/Octave code to solve numerically the equation $x' = -x$, with initial condition $x(-5) = 1$, by performing 100 steps of the Euler scheme with stepsize $\Delta t = 0.1$. Draw the graph of this numerical solution. On the same graph, also plot the exact solution and compare.
- Use the function `ode45` to solve the same equation with the same initial condition, over the range $-5 \leq t \leq 5$. On the same graph, draw this new numerical solution and comment.
- Use `ode45` to solve numerically the equation $x' = t\cos(x)$ for the initial condition $x(0) = 1$ over the range $0 \leq t \leq 10$.
- Still for the equation $x' = t\cos(x)$, draw on the same graph the numerical solutions obtained with the different initial conditions $x(0) = 0$, $x(0) = 0.1$, $x(0) = 0.2$, ... $x(0) = 1$.

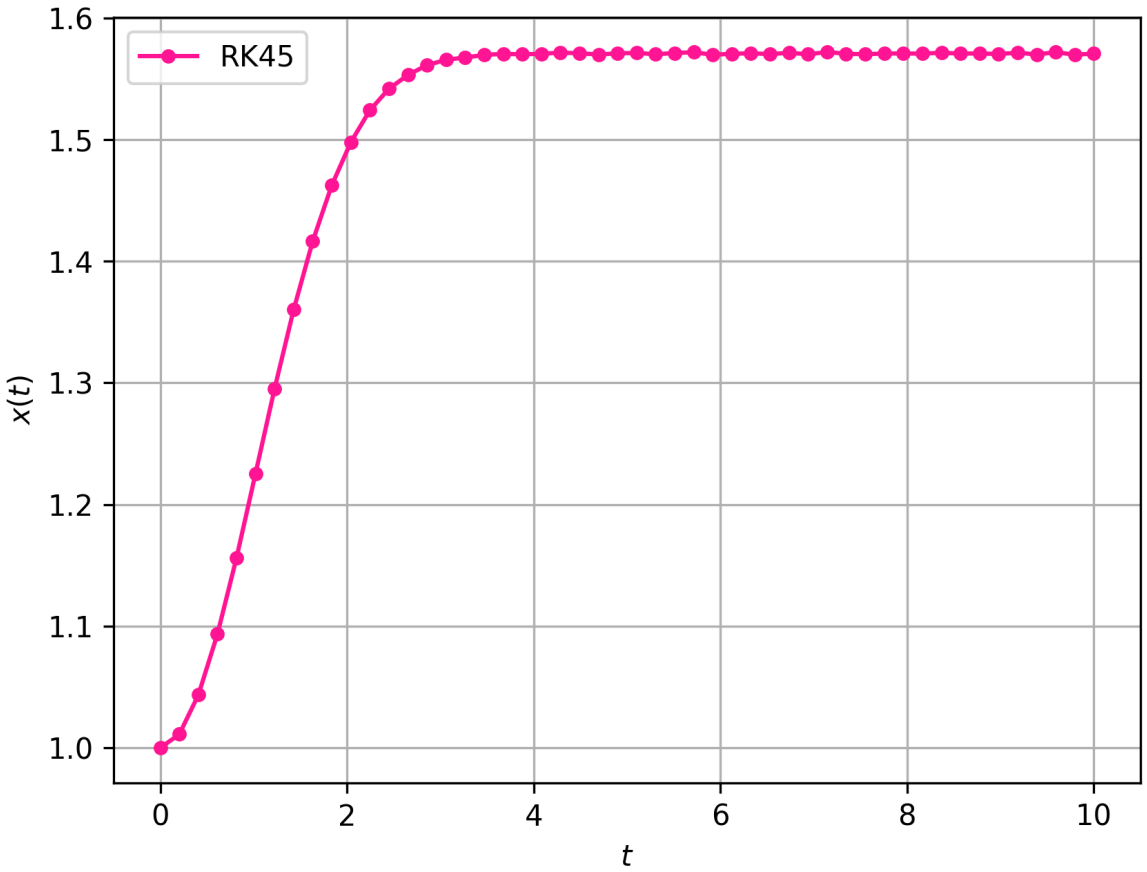
Practice using Python - correction

- Solving numerically the equation $x' = -x$, with initial condition $x(-5) = 1$: Forward Euler, Backward Euler and RK45 methods.



Practice using Python - correction

- Solving numerically the equation $x' = t\cos(x)$, with initial condition $x(0) = 1$: RK45 method.



Practice using Python - correction

- Solving numerically the equation $x' = t\cos(x)$, with different initial conditions: RK45 method.