EOSC 213: Computational Methods in Geological Engineering

Lecture 4: Finite differences + Forward Euler (numerical IVPs)

Shadab Ahamed

Department of Earth, Ocean and Atmospheric Sciences,
The University of British Columbia

Thu, Jan 15, 2026

Lecture 3 recap: what we learned

- A first-order ODE: $\frac{dx}{dt} = f(x, t; \lambda)$
- Initial condition selects a trajectory: $x(t_0) = x_0$
- Parameters shape the model (e.g., λ, r, K)
- Some ODEs have analytic solutions:
  - separable ODEs (exponential decay, logistic growth)
  - integrating factors (linear first-order ODEs)

Today: from calculus to code

- On a computer, we do **discrete steps**
- So we need to approximate:
  - derivatives $\Rightarrow$ finite differences
  - time evolution $\Rightarrow$ time stepping (Forward Euler today)

Goal: turn an IVP into an algorithm:

$$\frac{dx(t)}{dt} = f(t, x), \quad x(t_0) = x_0 \quad \Rightarrow \quad \{x_0, x_1, \ldots, x_N\} \text{ at } \{t_0, t_1, \ldots, t_N\}$$

Goal: approximate derivatives

Why approximate derivatives?

- to solve ODEs on a computer
- to estimate slopes from data
- to support design / optimization workflows

We will start with the key tool:

$$x'(t) = \frac{dx(t)}{dt} \approx \frac{x(t+h) - x(t)}{h}$$

Taylor expansion idea

Start from Taylor series around t:

$$x(t+h) = x(t) + x'(t)h + \frac{1}{2}x''(t)h^2 + \frac{1}{6}x'''(t)h^3 + \cdots$$

Rearrange to isolate $x'(t)$:

$$\frac{x(t+h) - x(t)}{h} = x'(t) + \frac{1}{2}x''(t)h + \mathcal{O}(h^2)$$

Takeaway:

$$\frac{x(t+h) - x(t)}{h} \text{ is first-order accurate } \big(\mathcal{O}(h)\big)$$

Forward and backward differences (first-order)

Forward difference:

$$x'(t) \approx \frac{x(t+h) - x(t)}{h} \qquad \text{error } \mathcal{O}(h)$$

Backward difference:

$$x'(t) \approx \frac{x(t) - x(t-h)}{h} \qquad \text{error } \mathcal{O}(h)$$

Both use **one-sided** information.

Today we will use the forward difference to build Forward Euler.

Central difference (higher accuracy idea)

If we add the Taylor expansions at $t + h$ and $t - h$, we get:

$$\frac{x(t+h) - x(t-h)}{2h} = x'(t) + \mathcal{O}(h^2)$$

Central difference is second-order accurate.

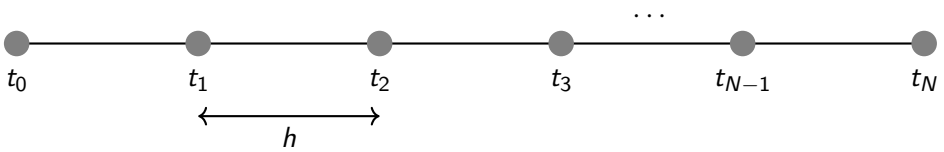
We will mostly focus on:

- using forward difference for time stepping (Forward Euler)
- understanding how accuracy depends on h

Finite differences on a mesh

On a computer, we store values on a grid:

$$t_0, t_1, \ldots, t_n \quad \text{with step } h = t_{j+1} - t_j$$



Function values become a vector:

$$x = [x(t_0), x(t_1), \ldots, x(t_n)]^\top$$

Forward difference at x_j:

$$x'(t_j) \approx \frac{x(t_{j+1}) - x(t_j)}{h}$$

Key idea: derivatives become *operations on arrays*.

From derivative to time stepping

We want to solve:

$$x'(t) = f(t, x), \qquad x(t_0) = x_0$$

Use forward difference for $x'(t)$:

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

Set it equal to $f(t, x)$:

$$\frac{x(t+h) - x(t)}{h} = f(t, x)$$

Rearrange:

$$x(t+h) = x(t) + h\, f(t, x)$$

Forward Euler (algorithm form)

Define a time grid:

$$t_0,\ t_1 = t_0 + h, t_2 = t_0 + 2h, t_3 = t_0 + 3h,\ \ldots,\ t_N = t_0 + Nh$$

Let $x_j \approx x(t_j)$, then Forward Euler is:

$$x_{j+1} = x_j + h\, f(t_j, x_j)$$

This is a complete algorithm:

- start with x_0
- repeat update for $j = 0, 1, \ldots, N-1$

Pseudocode (what we will implement in PyTorch)

Algorithm 1: Forward Euler time stepping

Input: t_0, x_0, h, N, and function $f(t, x)$

$t[0] \leftarrow t_0$, $x[0] \leftarrow x_0$

for $j \leftarrow 0$ **to** $N - 1$ **do**

$t[j+1] \leftarrow t[j] + h$

$x[j+1] \leftarrow x[j] + h\, f\big(t[j], x[j]\big)$

Output: vectors $t[0:N]$, $x[0:N]$

Corresponding Python code

```python
def forward_euler(f, t0, x0, h, N):
    # Make x0 a tensor
    x0 = torch.as_tensor(x0, dtype=torch.float32)

    # Time grid: [t0, t0 + h, t0 + 2h, ..., t0 + Nh]
    t = t0 + h * torch.arange(N + 1, dtype=torch.float32)
    # Allocate solution array
    x = torch.zeros((N + 1,) + x0.shape, dtype=torch.float32)
    x[0] = x0
    # Loop (time stepping)
    for j in range(N):
        x[j + 1] = x[j] + h * f(t[j], x[j])

    return t, x
```

Finite difference example: exponential decay

Consider the exponential decay model:

$$\frac{dx}{dt} = -\lambda x, \qquad \lambda > 0$$

We approximate the derivative using a **forward difference**:

$$\frac{dx(t)}{dt} \approx \frac{x(t+h) - x(t)}{h}$$

Substitute into the ODE:

$$\frac{x(t+h) - x(t)}{h} = -\lambda x(t)$$

Rearrange:

$$x(t+h) = x(t) - \lambda h\, x(t)$$

Key idea: the continuous decay law becomes a *discrete update rule*.

Forward Euler update for exponential decay

On a time grid $t_j = t_0 + jh$, let $x_j \approx x(t_j)$.

From the finite difference derivation:

$$x_{j+1} = x_j - \lambda h\, x_j$$

Factor:

$$x_{j+1} = (1 - \lambda h)\, x_j$$

Interpretation:

- Each step multiplies the solution by $(1 - \lambda h)$
- If h is small, this mimics smooth exponential decay
- If h is too large, the method may become inaccurate or unstable (we will talk about *stability of numerical solutions* in the next class)

This is exactly what we will code next.

Unfolding the Forward Euler recursion

From Forward Euler for exponential decay:

$$x_{j+1} = (1 - \lambda h)\, x_j$$

Starting from the initial value x_0:

$$\begin{aligned} x_1 &= (1 - \lambda h)\, x_0 \\ x_2 &= (1 - \lambda h)\, x_1 = (1 - \lambda h)^2\, x_0 \\ x_3 &= (1 - \lambda h)\, x_2 = (1 - \lambda h)^3\, x_0 \\ &\vdots \end{aligned}$$

After j steps:

$$x_j = (1 - \lambda h)^j\, x_0$$

Unfolding the Forward Euler recursion: exact vs numerical

Key insight:

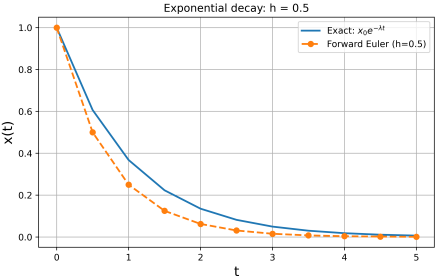
- Forward Euler produces a *discrete exponential*
- Compare with the exact solution:

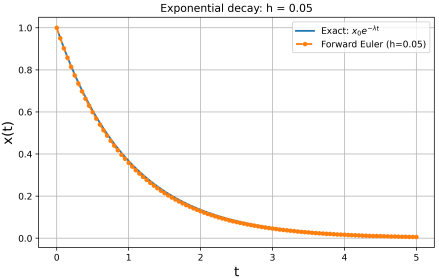
$$x(t_j) = x_0 e^{-\lambda t_j} = x_0 e^{-\lambda j h}$$

Forward Euler:

$$x_{j+1} = (1 - \lambda h)\, x_j$$

If h is smaller, the discrete curve better matches the smooth exponential.





How do we pick the step size h? (the main idea)

Smaller h usually means:

- more accurate approximation
- more computation (more steps)

A practical check (conceptual):

- solve with step H
- solve again with step $h = H/2$
- compare the two solutions on the coarse grid points

If the difference is small enough, you can trust the result more. We will discuss how to choose h in more detail in future.

What can go wrong?

Forward Euler is simple, but:

- it can be inaccurate if h is too large
- it can become unstable for some problems (even if the model is correct)

We will see this in code using examples like:

- exponential decay
- logistic growth
- a differential equation with no analytical solution

Main lesson: numerical methods need accuracy *and* stability. We will go into more details in these topics in future lectures.

Now we code

In the live notebook, we will:

- implement Forward Euler in PyTorch
- test on exponential decay, logistic growth, and a differential equation with non-analytical solution
- visualize solutions vs step size
- build intuition: accuracy vs cost

Tip: write small helper functions and label every plot axis.

Summary

- Taylor series $\Rightarrow$ finite differences approximate derivatives
- Forward difference leads to a simple time-stepping method
- Forward Euler update:

$$x_{j+1} = x_j + h\, f(t_j, x_j)$$

- Step size matters (accuracy, cost, stability)

Next: (in coding) implement, test, and visualize Forward Euler.