EOSC 213: Computational Methods in Geological Engineering

Lecture 5: Beyond Forward Euler (Explicit Midpoint + Backward Euler)

Shadab Ahamed

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

Tue, Jan 20, 2026

Plan for today (slides + coding)

- **Slides:**
  - Why Forward Euler can fail (motivation)
  - Two alternatives: **Explicit Midpoint** and **Backward Euler**
  - What changes: accuracy vs stability vs computation cost
- **Live coding (last $\sim$30 minutes):**
  - Implement Explicit Midpoint and Backward Euler for simple IVPs
  - Compare with Forward Euler (plots + step size experiments)

Recap (Lecture 4): Forward Euler

We solved IVPs

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

by time stepping on a grid $t_j = t_0 + jh$.

Forward Euler:

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

Takeaway:

- Forward Euler is **simple** and easy to implement
- Step size h controls accuracy and computational cost
- For some problems, Forward Euler can behave **badly** if h is too large

Motivation for why Forward Euler is not enough: same ODE can have very different numerical behaviors

Consider exponential decay:

$$x'(t) = -\lambda x(t), \qquad \lambda > 0$$

Exact solution:

$$x(t) = x_0 e^{-\lambda t}$$

Forward Euler gives:

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

Problems: if h is too large, $(1 - \lambda h)$ can be negative or have magnitude > 1.

- what happens when $(1 - \lambda h)$ is negative?
- when happens when the magnitude $|1 - \lambda h| > 1$?

When $(1 - \lambda h)$ is negative: (non-physical) oscillating solution

Forward Euler update for exponential decay:

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

Case: $1 - \lambda h < 0$

- Each step multiplies the solution by a **negative factor**
- This causes **sign flips**:

$$x_0 > 0 \;\Rightarrow\; x_1 < 0 \;\Rightarrow\; x_2 > 0 \;\Rightarrow\; \ldots$$

- The numerical solution oscillates around zero

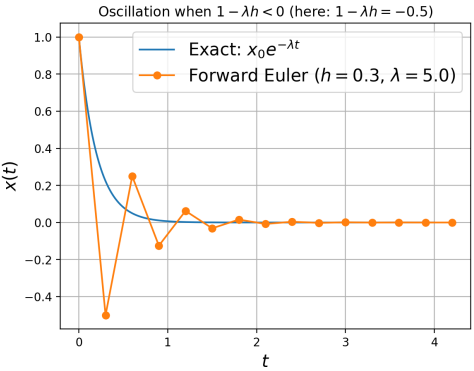


Figure: Forward Euler with $1 - \lambda h < 0$ (sign-changing oscillations)

When $(1 - \lambda h)$ is negative: (non-physical) oscillating solution

Why is this non-physical?

- Exact solution: $x(t) = x_0 e^{-\lambda t}$ (no sign changes)
- The oscillation is introduced purely by the numerical method

Important distinction:

- $|1 - \lambda h| < 1$: oscillatory but *numerically stable*
- Still *physically incorrect*

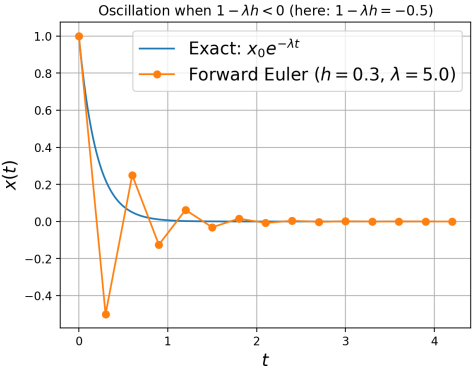


Figure: Forward Euler with $1 - \lambda h < 0$ (sign-changing oscillations)

When $|1 - \lambda h| > 1$: unstable, diverging solution

Forward Euler update for exponential decay:

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

Case: $|1 - \lambda h| > 1$

- Each step multiplies the solution by a factor with magnitude > 1
- Errors are **amplified** at every time step
- Instead of decaying, the numerical solution **grows**

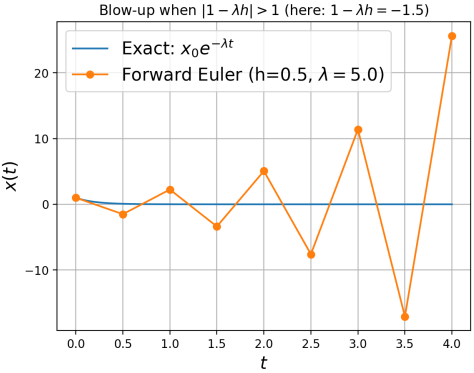


Figure: Forward Euler with $|1 - \lambda h| > 1$ (unstable and diverging)

When $|1 - \lambda h| > 1$: unstable, diverging solution

What happens mathematically?

$$x_N = (1 - \lambda h)^N x_0$$

If $|1 - \lambda h| > 1$, then: $|x_N| \to \infty$ as $N \to \infty$

Why is this catastrophic?

- Exact solution always decays to zero
- Numerical solution diverges to infinity

Conclusion:

- Forward Euler is a genuinely **unstable numerical method**
- The simulation is completely unreliable unless the value of h is smartly chosen

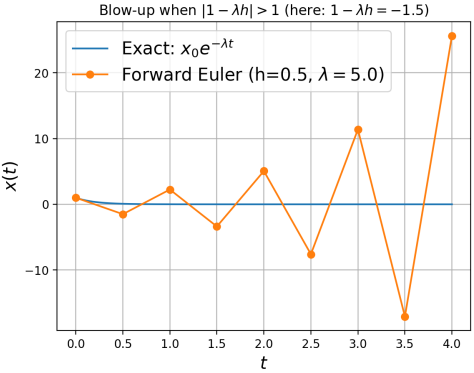
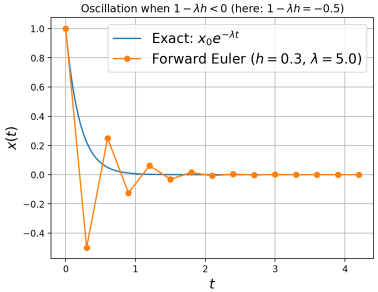


Figure: Forward Euler with $|1 - \lambda h| > 1$ (unstable and diverging)

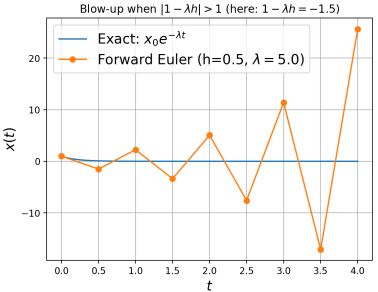
Same ODE, very different numerical outcomes



Oscillations but convergence

$1 - \lambda h < 0$

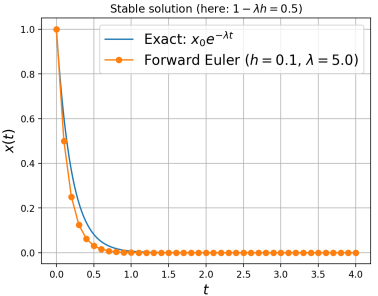
Sign flips (non-physical)



Oscillations and blow-up

$|1 - \lambda h| > 1$

Numerical instability



Stable solution

$0 < (1 - \lambda h) < 1$

Smooth, monotone decay

Key lesson: the ODE is stable - the numerical method may not be.

Two ways to improve (big picture)

Forward Euler for exponential decay:

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

When Forward Euler struggles, two common fixes are:

(1) Use more information per step $\Rightarrow$ better accuracy $\mathcal{O}(h^2)$

- Example: **Explicit Midpoint** (a 2nd-order method)

(2) Change where you evaluate the slope $\Rightarrow$ better stability

- Example: **Backward Euler** (implicit, very stable for decay)

Today we will see both.

Explicit Midpoint: idea

Forward Euler uses the slope at the start of the interval:

$$x_{j+1} = x_j + hf(t_j, x_j)$$

Explicit Midpoint uses a **midpoint slope** instead:

$$x_{j+1} = x_j + h\, f\left(t_j + \frac{h}{2},\ x_{j+\frac{1}{2}}\right)$$

where $x_{j+\frac{1}{2}}$ is an estimate of the state halfway through the step.

This often gives better accuracy for the same step size h.

Explicit Midpoint: the actual update

Step 1: take a half-step with Forward Euler:

$$x_{j+\frac{1}{2}} = x_j + \frac{h}{2} f(t_j, x_j)$$

Step 2: use the midpoint slope for a full step:

$$x_{j+1} = x_j + h\, f\left(t_j + \frac{h}{2},\ x_{j+\frac{1}{2}}\right)$$

Key facts (for now):

- typically **more accurate** than Forward Euler
- can still have stability issues (we'll formalize this in the stability analysis lecture in the next class)

Forward Euler vs Explicit Midpoint (pseudocode)

Algorithm 1: Forward Euler

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**

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

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

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

Algorithm 2: Explicit Midpoint

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**

$\quad x_{\text{half}} \leftarrow x[j] + \frac{h}{2} f\big(t[j], x[j]\big)$

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

$\quad x[j+1] \leftarrow x[j] + h\, f\big(t[j] + \frac{h}{2}, x_{\text{half}}\big)$

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

Key difference: Explicit Midpoint evaluates the slope at a predicted half-step.

Exponential decay: Forward Euler vs Explicit Midpoint

We solve the same IVP for exponential decay with $\lambda = 5.0$, $h = 0.3 \implies (1 - \lambda h) < 0$:

Forward Euler: $\mathcal{O}(h)$ or 1^{st}-order accurate

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

Explicit Midpoint: $\mathcal{O}(h^2)$ or 2^{nd}-order accurate

$$x_{j+1} = x_j + h\, f\left(t_j + \frac{h}{2},\, x_{j+\frac{1}{2}}\right)$$

It uses a **midpoint slope**, which is typically more accurate.

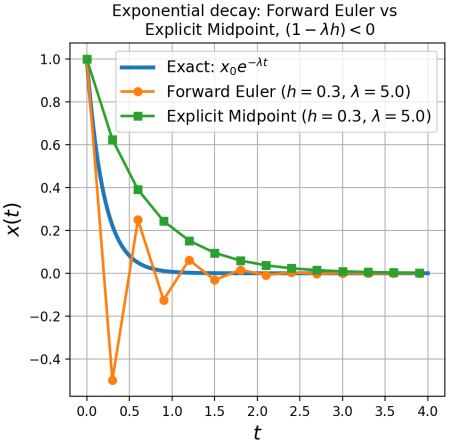


Figure: Forward Euler vs. Explicit Midpoint: Same h, same ODE, different numerical method $\Rightarrow$ different accuracy.

Exponential decay: Forward Euler vs Explicit Midpoint

What you should notice in the plot:

- Midpoint method tracks the exact decay more closely for the same h
- Forward Euler has larger step-to-step error (more numerical damping / distortion)
- **Accuracy vs. Cost**: midpoint does more work per step, but gives better accuracy

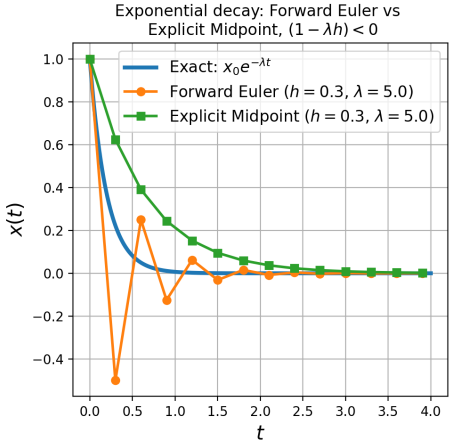


Figure: Forward Euler vs. Explicit Midpoint: Same h, same ODE, different numerical method $\Rightarrow$ different accuracy.

Backward Euler: idea

Forward Euler evaluates the slope at (t_j, x_j).

Backward Euler evaluates the slope at the **end** of the step:

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

Notice x_{j+1} appears on **both sides**.

That means:

- Backward Euler is **implicit**
- Each step may require solving an equation for x_{j+1}

Backward Euler for exponential decay: solvable algebraically

Consider:

$$x'(t) = -\lambda x(t) \quad \Rightarrow \quad f(t, x) = -\lambda x$$

Backward Euler:

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

Bring terms together:

$$x_{j+1}(1 + \lambda h) = x_j \quad \Rightarrow \quad x_{j+1} = \frac{1}{1 + \lambda h} x_j$$

Compare to Forward Euler:

$$\text{Forward Euler: } x_{j+1} = (1 - \lambda h)x_j, \qquad \text{Backward Euler: } x_{j+1} = \frac{1}{1 + \lambda h} x_j$$

- For Forward Euler: $|1 - \lambda h| \gtrless 1$, so the solution may be stable or unstable
- For Backward Euler: $0 < \left|\frac{1}{1+\lambda h}\right| < 1$ for any value of h, so the solution is always decaying (hence always stable)

Backward Euler on general nonlinear ODEs (preview)

For a general $f(t, x)$, Backward Euler step is:

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

This can be written as a root-finding problem:

$$g(x_{j+1}) \equiv x_{j+1} - x_j - h\, f(t_{j+1}, x_{j+1}) = 0$$

In this course:

- today, we will use Backward Euler for cases where we can solve it easily (like exponential decay)
- later, we will learn systematic tools (Newton's method / fixed-point iteration)

Forward Euler vs Backward Euler: pseudocode

Algorithm 3: Forward Euler (explicit)

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

$t[0] \leftarrow t_0$

$x[0] \leftarrow x_0$

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

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

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

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

Algorithm 4: Backward Euler (implicit)

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

$t[0] \leftarrow t_0$

$x[0] \leftarrow x_0$

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

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

$\quad$ `// Solve implicit equation for` $x[j+1]$

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

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

Key difference: Forward Euler computes x_{j+1} directly, Backward Euler requires solving an equation at each step.

Exponential decay: Forward Euler vs Backward Euler

We solve the same IVP for exponential decay with $\lambda = 5.0$, $h = 0.3$:

Forward Euler (explicit):

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

Backward Euler (implicit):

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

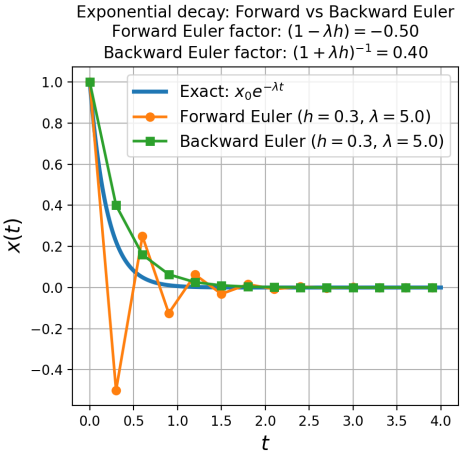


Figure: Forward Euler vs. Backward Euler: Same h, same ODE, different numerical method $\Rightarrow$ different stability.

Exponential decay: Forward Euler vs Backward Euler

Key comparison (per-step factor):

$$\underbrace{(1-\lambda h)}_{\text{Forward Euler}} \quad \text{vs} \quad \underbrace{\frac{1}{1+\lambda h}}_{\text{Backward Euler}}$$

- Forward Euler can oscillate / blow up if h is too large
- Backward Euler always decays for any $h > 0$ (more damping)

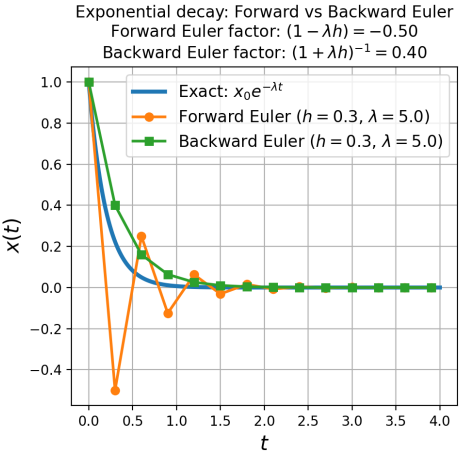


Figure: Forward Euler vs. Backward Euler: Same h, same ODE, different numerical method $\Rightarrow$ different stability.

Comparison between different methods

Method	Stability	Accuracy (order)	Computational cost	Typical strength
Forward Euler	Conditional	1st	Very low	simplest baseline
Explicit Midpoint	Conditional	2nd	Low–moderate	better accuracy per step
Backward Euler	Unconditionally stable	1st	High	strong stability for decay

Takeaway: trade off

- accuracy (how close)
- stability (whether it behaves sensibly)
- computational cost (time per step)

What you should takeaway today

By the end of Lecture 5, you should be able to:

- write down the update rule for:
  - Forward Euler
  - Explicit Midpoint
  - Backward Euler (and recognize when it's implicit)
- explain (in words) why different methods behave differently
- run basic experiments:
  - vary step size h
  - compare numerical vs analytical solutions when available
  - interpret what "bad behavior" looks like (oscillation, blow-up, wrong limit)

Live coding (last $\sim$30 mins): what we will do

In the notebook, we will:

- implement **Forward Euler** from scratch
- implement **Explicit Midpoint** from scratch
- implement **Backward Euler** from scratch (closed-form step)
- compare all three methods on:
  - exponential decay (analytical solution known)
- run a step-size study:
  - small h vs large h to see accuracy + stability effects

Summary

- Forward Euler is simple, but can behave badly for large step sizes h
- Explicit Midpoint improves accuracy by using midpoint information
- Backward Euler evaluates the slope at the end of the step (implicit)
- Different methods = different tradeoffs (accuracy, stability, cost)

Next lecture: stability analysis (systematic way to predict when things go wrong)