

## Level 4: Finite Differences and the Heat Equation







**Pavni:** Acharya, how do we actually *approximate* derivatives on a computer?

**Acharya:** We use the **finite difference method**. A computer only works with discrete points, so we replace continuous derivatives with **difference quotients** on a **grid**. These come directly from the **Taylor series**, and because we truncate the series, each formula comes with a predictable **error term**.

**Pavni:** Can you give me an example?

**Acharya:** Suppose we have points  $x_0, x_1, \dots, x_N$  with spacing  $\Delta x$ . Around a point  $x_i$ , Taylor's theorem gives us expansions for  $u(x_{i+1})$  and  $u(x_{i-1})$ . By combining them, we obtain difference formulas:

- **Forward difference** for the first derivative:

$$u'(x_i) = \frac{u(x_{i+1}) - u(x_i)}{\Delta x} - \frac{1}{2}u''(\xi) \Delta x,$$

so the truncation error is  $\mathcal{O}(\Delta x)$ .

- **Backward difference:**

$$u'(x_i) = \frac{u(x_i) - u(x_{i-1})}{\Delta x} + \frac{1}{2}u''(\xi) \Delta x,$$

also error  $\mathcal{O}(\Delta x)$ .

- **Central difference:**

$$u'(x_i) = \frac{u(x_{i+1}) - u(x_{i-1}))}{2\Delta x} - \frac{1}{6}u^{(3)}(\xi) (\Delta x)^2,$$

so the error is  $\mathcal{O}((\Delta x)^2)$  — more accurate.

- **Second derivative** (central difference):

$$u''(x_i) = \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1}))}{(\Delta x)^2} - \frac{1}{12}u^{(4)}(\xi) (\Delta x)^2,$$

with error  $\mathcal{O}((\Delta x)^2)$ .

**Pavni:** So these formulas are not exact — they always come with an error term?

**Acharya:** Precisely. That error is called the **local truncation error**. When we build numerical schemes for PDEs, these errors accumulate step by step, and we'll have to balance them with stability.

**Pavni:** I see. So finite differences give us algebraic formulas for derivatives, but with a known accuracy.

**Acharya:** Exactly. That's the foundation of finite difference methods.



## Application: The Heat Equation

**Acharya:** Consider the 1D heat equation:

$$u_t = \alpha^2 u_{xx}, \quad 0 < x < 1, \quad t > 0$$

with boundary conditions  $u(0, t) = u(1, t) = 0$  and initial profile  $u(x, 0) = f(x)$ .

We set up a grid: - In space:  $x_i = i\Delta x$ ,  $i = 0, \dots, N$

- In time:  $t^n = n\Delta t$ ,  $n = 0, 1, 2, \dots$

At each point, let  $u_i^n \approx u(x_i, t^n)$ .

**Pavni:** And now we replace derivatives?

**Acharya:** Correct.

- Time derivative (forward difference):

$$u_t(x_i, t^n) \approx \frac{u_i^{n+1} - u_i^n}{\Delta t}$$

- Spatial second derivative (central difference):

$$u_{xx}(x_i, t^n) \approx \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{(\Delta x)^2}$$

Plugging these into the PDE, we get:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha^2 \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{(\Delta x)^2}.$$

Rearranging:

$$u_i^{n+1} = u_i^n + \lambda (u_{i-1}^n - 2u_i^n + u_{i+1}^n),$$

where  $\lambda = \frac{\alpha^2 \Delta t}{(\Delta x)^2}$ .

## Stability Condition

**Pavni:** So we can just keep applying this formula to march forward in time?

**Acharya:** Yes, but with a caveat. This scheme, called **FTCS (Forward Time, Central Space)**, is only stable if:

$$\lambda \leq \frac{1}{2}.$$

**Pavni:** So if  $\Delta t$  is too large, the scheme fails?

**Acharya:** Exactly. The numerical solution will blow up, even though the true solution is stable. Choosing  $\Delta t$  small enough ensures stability.



**i** Why the FTCS stability condition is  $\lambda \leq \frac{1}{2}$

If the update is

$$u^{(n+1)} = Au^{(n)},$$

and the initial error is  $e^0$ , then after  $n$  steps the error is

$$e^{(n)} = A^n e^0.$$

Using an induced matrix norm:

$$\|e^{(n)}\| = \|A^n e^0\| \leq \|A^n\| \|e^0\| \leq \|A\|^n \|e^0\|.$$

- If  $\|A\| \leq 1$ , the error never grows.
- If  $\|A\| < 1$ , the error decays as  $n \rightarrow \infty$ .
- In general, the asymptotic condition is that the **spectral radius**  $\rho(A) < 1$ .

### Eigenvalues of the FTCS matrix

For the FTCS tridiagonal matrix  $A$  (symmetric Toeplitz), the eigenvalues are

$$\mu_k = 1 - 4\lambda \sin^2\left(\frac{k\pi}{2(N-1)}\right), \quad k = 1, 2, \dots, N-2,$$

where  $\lambda = \frac{\alpha^2 \Delta t}{(\Delta x)^2}$ .

Thus the spectral radius is

$$\rho(A) = \max_{1 \leq k \leq N-2} |\mu_k| = \max_{1 \leq k \leq N-2} \left| 1 - 4\lambda \sin^2\left(\frac{k\pi}{2(N-1)}\right) \right|.$$

### Stability condition

- As  $N \rightarrow \infty$  (i.e.  $\Delta x \rightarrow 0$ ), the maximum of  $\sin^2(\cdot)$  tends to 1.
- Therefore the most restrictive case is

$$|1 - 4\lambda| \leq 1.$$

- This simplifies to

$$0 \leq \lambda \leq \frac{1}{2}.$$

Thus, the **FTCS scheme is stable if and only if**

$$\lambda = \frac{\alpha \Delta t}{(\Delta x)^2} \leq \frac{1}{2}.$$



## Mini-Quizzes

### Quiz 1:

Let  $u(x) = x^2$ . With  $\Delta x = 0.1$ , approximate  $u'(1)$  using:

1. Forward difference
2. Central difference

Compare with the exact derivative  $u'(1) = 2$ . Which is more accurate?

#### 💡 Answer 1

- Forward difference:

$$\frac{u(1 + \Delta x) - u(1)}{\Delta x} = \frac{(1.1)^2 - 1^2}{0.1} = \frac{1.21 - 1}{0.1} = 2.1.$$

- Central difference:

$$\frac{u(1 + \Delta x) - u(1 - \Delta x)}{2\Delta x} = \frac{(1.1)^2 - (0.9)^2}{0.2} = \frac{1.21 - 0.81}{0.2} = 2.0.$$

- Exact derivative:  $u'(1) = 2$ .

**Conclusion:** The central difference gives the exact value here (error 0), while the forward difference has error 0.1. Central is more accurate (as expected — it is second-order).

### Quiz 2:

Suppose  $\alpha = 1$ ,  $\Delta x = 0.1$ . What is the maximum  $\Delta t$  for stability in the explicit scheme?

(Hint:  $\lambda = \frac{\alpha \Delta t}{(\Delta x)^2} \leq \frac{1}{2}$ .)

#### 💡 Answer 2

We need

$$\lambda = \frac{\alpha \Delta t}{(\Delta x)^2} \leq \frac{1}{2}.$$

With  $\alpha = 1$  and  $\Delta x = 0.1$ ,  $(\Delta x)^2 = 0.01$ . So

$$\frac{\Delta t}{0.01} \leq \frac{1}{2} \quad \Rightarrow \quad \Delta t \leq 0.01 \times \frac{1}{2} = 0.005.$$

**Maximum allowable  $\Delta t = 0.005$ .**



## Heat equation — Matrix Method with Non-zero Dirichlet Conditions

**Pavni:** We already saw how to discretize the heat equation with FTCS. But can we write the whole scheme in a more compact way?

**Acharya:** Yes. That's where the **matrix method** comes in. Let's recall the PDE:

$$u_t = \alpha^2 u_{xx}, \quad 0 < x < 1, \quad t > 0,$$

with Dirichlet conditions

$$u(0, t) = L, \quad u(1, t) = R,$$

and initial condition  $u(x, 0) = f(x)$ .

**Pavni:** So we still set up the grid in space and time?

**Acharya:** Exactly. The FTCS update at interior nodes is

$$u_i^{j+1} = u_i^j + \lambda (u_{i-1}^j - 2u_i^j + u_{i+1}^j), \quad i = 1, \dots, N_x - 2,$$

where  $\lambda = \frac{\alpha^2 \Delta t}{\Delta x^2}$ .

**Pavni:** That's a lot of coupled equations. How do we collect them?

**Acharya:** We put all interior values into a vector

$$u^{(j)} = \begin{bmatrix} u_1^j \\ u_2^j \\ \vdots \\ u_{N_x-2}^j \end{bmatrix}.$$

**Pavni:** And the update rule becomes a matrix multiplication?

**Acharya:** Yes. We can write

$$u^{(j+1)} = A u^{(j)} + b,$$

where  $A$  is tridiagonal and  $b$  accounts for the boundary conditions:

$$A = \begin{bmatrix} 1-2\lambda & \lambda & & & & \\ \lambda & 1-2\lambda & \lambda & & & \\ & \ddots & \ddots & \ddots & & \\ & & \lambda & 1-2\lambda & \lambda & \\ & & & \lambda & 1-2\lambda \end{bmatrix}, \quad b = \begin{bmatrix} \lambda L \\ 0 \\ \vdots \\ 0 \\ \lambda R \end{bmatrix}.$$

**Pavni:** So if  $L = R = 0$ , then  $b = 0$  and we just have  $u^{(j+1)} = A u^{(j)}$ .

**Acharya:** Precisely. That's the beauty of the matrix method: it organizes the scheme into a linear algebra update.

---



## Implementation in Python

Let us now implement the method in Python and compare with the exact solution for  $f(x) = \sin(\pi x)$ :

$$u(x, t) = e^{-\pi^2 \alpha t} \sin(\pi x).$$

```
import numpy as np
import matplotlib.pyplot as plt

alpha = 1.0
Nx     = 50
T      = 0.1
L, R   = 0.0, 0.0

dx = 1.0/(Nx-1)
dt_max = dx*dx/(2*alpha)
s      = 0.4
dt     = s*dt_max
Nt     = int(T/dt)
dt     = T/Nt
r      = alpha*dt/dx**2

x = np.linspace(0.0, 1.0, Nx)
u0 = np.sin(np.pi * x)

m = Nx - 2
main = (1 - 2*r) * np.ones(m)
off  = r * np.ones(m-1)
A = np.diag(main) + np.diag(off, 1) + np.diag(off, -1)

b = np.zeros(m)
b[0], b[-1] = r*L, r*R

u = u0.copy()
u_in = u[1:-1].copy()
snapshots = []
snap_times = np.linspace(0, T, Nt+1, dtype=int)

for j in range(Nt):
    u_in = A @ u_in + b
    u[1:-1] = u_in
    if j in snap_times:
        snapshots.append((j*dt, u.copy()))

plt.figure(figsize=(8,4))
first_exact = True
```

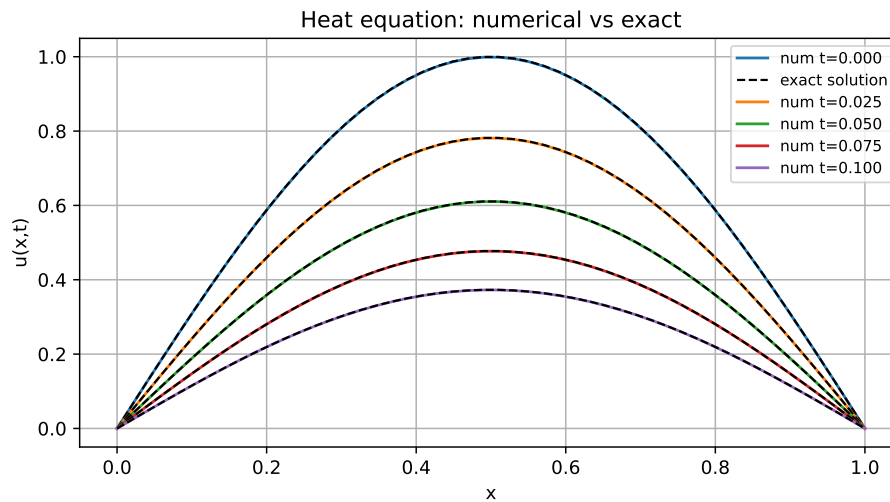


```

for t_here, u_snap in snapshots:
    plt.plot(x, u_snap, label=f"num t={t_here:.3f}")
    u_exact = np.exp(-np.pi**2*alpha*t_here) * np.sin(np.pi*x)
    if first_exact:
        plt.plot(x, u_exact, 'k--', linewidth=1.2, label='exact solution')
        first_exact = False
    else:
        plt.plot(x, u_exact, 'k--', linewidth=1.2, label="_nolegend_")

plt.xlabel("x"); plt.ylabel("u(x,t)")
plt.title("Heat equation: numerical vs exact")
plt.legend(fontsize="small")
plt.grid(True)
plt.show()

```



## Remarks

- The **linear algebra structure**  $u^{(j+1)} = Au^{(j)} + b$  makes the scheme compact and systematic.
- The stability restriction  $\lambda \leq \frac{1}{2}$  follows from analyzing the eigenvalues of  $A$ .
- For large  $N_x$ ,  $A$  is sparse and tridiagonal — in practice, use `scipy.sparse.diags` for efficiency.



## Backward Difference (Implicit) Scheme for the Heat Equation

**Pavni:** Acharya, the FTCS scheme works only if  $\lambda \leq \frac{1}{2}$ . Is there a method that avoids this restriction?

**Acharya:** Yes. We can use a **backward difference in time**, together with the same central difference in space. This gives the **Backward Euler scheme**, which is implicit but unconditionally stable.

**Pavni:** Implicit? What does that mean?

**Acharya:** It means that the new values  $u^{n+1}$  appear on both sides of the equation, so we must solve a system of equations at each time step.

### Derivation

Start from the PDE:

$$u_t = \alpha^2 u_{xx}.$$

- Approximate the time derivative with a **backward difference**:

$$u_t(x_i, t^{n+1}) \approx \frac{u_i^{n+1} - u_i^n}{\Delta t}.$$

- Approximate the spatial second derivative at the new time level:

$$u_{xx}(x_i, t^{n+1}) \approx \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{(\Delta x)^2}.$$

The scheme becomes:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha^2 \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{(\Delta x)^2}.$$

Rearrange:

$$-\lambda u_{i-1}^{n+1} + (1 + 2\lambda) u_i^{n+1} - \lambda u_{i+1}^{n+1} = u_i^n, \quad \lambda = \frac{\alpha^2 \Delta t}{(\Delta x)^2}.$$

### Matrix Form

Let  $u^{(n)}$  be the vector of interior values at time step  $n$ . Then

$$Bu^{(n+1)} = u^{(n)},$$



where

$$B = \begin{bmatrix} 1+2\lambda & -\lambda & & & & \\ -\lambda & 1+2\lambda & -\lambda & & & \\ & \ddots & \ddots & \ddots & & \\ & & -\lambda & 1+2\lambda & -\lambda & \\ & & & -\lambda & 1+2\lambda \end{bmatrix}_{(N_x-2) \times (N_x-2)}.$$

So each step requires solving the linear system

$$u^{(n+1)} = B^{-1}u^{(n)}.$$

## Stability

**Pavni:** Doesn't that make it more expensive than FTCS?

**Acharya:** It does, because we must solve a tridiagonal system at every time step.

But the reward is **unconditional stability**: for any  $\Delta t > 0$  and  $\Delta x > 0$ , the scheme does not blow up.

**Pavni:** So no restriction like  $\lambda \leq \frac{1}{2}$ ?

**Acharya:** Exactly. Backward Euler is stable for all  $\lambda$ .

It is only first-order accurate in time (like FTCS), but still second-order in space.

## Remarks

- Backward Euler is more robust but requires solving a linear system at each step.
- For large systems, efficient algorithms like the **Thomas algorithm** (specialized Gaussian elimination for tridiagonal matrices) are used.
- In practice, one balances cost (explicit FTCS, cheap but conditionally stable) against robustness (implicit Backward Euler, unconditionally stable).

### **i** Why the Backward Euler scheme is unconditionally stable

If the update is

$$u^{(n+1)} = Au^{(n)},$$

and the initial error is  $e^0$ , then after  $n$  steps the error is



$$e^{(n)} = A^n e^0.$$

As before, the asymptotic condition is that the **spectral radius**  $\rho(A) < 1$ .

---

### Eigenvalues of the Backward Euler iteration matrix

The Backward Euler scheme for the heat equation is

$$\frac{U^{n+1} - U^n}{\Delta t} = \alpha^2 A U^{n+1},$$

which rearranges to

$$U^{n+1} = (I - \lambda A)^{-1} U^n, \quad \lambda = \frac{\alpha^2 \Delta t}{(\Delta x)^2}.$$

- If  $\lambda_i(A)$  are the eigenvalues of the discrete Laplacian  $A$ , then the eigenvalues of the iteration matrix are

$$\mu_i = \frac{1}{1 - \lambda_i(A)}.$$

- For the 1D Laplacian with Dirichlet BCs,

$$\lambda_i(A) = -4 \sin^2\left(\frac{i\pi}{2m}\right), \quad i = 1, 2, \dots, m-1.$$

Thus

$$\mu_i = \frac{1}{1 + 4\lambda \sin^2\left(\frac{i\pi}{2m}\right)}.$$


---

### Stability condition

- Since  $\lambda > 0$  and  $\sin^2(\cdot) \geq 0$ , the denominator is always greater than 1.
- Therefore

$$0 < \mu_i < 1, \quad \forall i.$$

This means all eigenvalues of the iteration matrix lie strictly inside the unit circle.

---

Thus, the **Backward Euler scheme is unconditionally stable**: - No restriction on  $\Delta t$ .

- Every mode decays monotonically.
- In contrast, FTCS required  $\lambda \leq \frac{1}{2}$  for stability.



## Interactive plots to see the eigenvalues of FTCS and Backward Difference methods

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

---

**How to interact with the plots** - Move the **slider** to change  $\lambda = \frac{\alpha^2 \Delta t}{(\Delta x)^2}$ .

- **FTCS plot:** notice that for  $\lambda > 0.5$  some eigenvalues leave  $[-1, 1]$ , indicating instability.
- **Backward Euler plot:** eigenvalues always remain in  $(0, 1)$ , showing unconditional stability.
- Hover over points to see their values, zoom by dragging, double-click to reset.







# Crank–Nicolson scheme

**Pavni:** We’ve studied FTCS and Backward Euler for the heat equation. FTCS was simple, but had that annoying stability restriction  $\lambda \leq \frac{1}{2}$ .

**Acharya:** Yes. And Backward Euler solved that problem — it was unconditionally stable.

**Pavni:** But both of them are only *first-order accurate in time*, right?

**Acharya:** Exactly. And that’s a problem if we want more accuracy without shrinking  $\Delta t$  too much.

**Pavni:** So FTCS is unstable unless  $\lambda \leq \frac{1}{2}$ , and Backward Euler is stable but too diffusive. We’re still missing something better.

**Acharya:** That’s where the **Crank–Nicolson scheme** comes in. The idea is to apply the trapezoidal rule in time — using the *average* of the spatial derivative at times  $n$  and  $n + 1$ .

**Pavni:** So instead of evaluating only at the old time (FTCS) or the new time (Backward Euler), we take a balance of both?

**Acharya:** Exactly. That trick makes the scheme **second-order accurate in time** and **unconditionally stable**. It’s like combining the strengths of both FTCS and Backward Euler.

**Pavni:** Ah, so Crank–Nicolson is really about accuracy as well as stability. That sounds worth deriving!

**Pavni:** Let’s derive Crank–Nicolson starting from the heat equation  $u_t = \alpha^2 u_{xx}$ . How do we get the time-centered scheme?

**Acharya:** Start by integrating the PDE in time over one step, from  $t^n$  to  $t^{n+1}$  at a fixed  $x$ :

$$u(x, t^{n+1}) - u(x, t^n) = \alpha^2 \int_{t^n}^{t^{n+1}} u_{xx}(x, t) dt.$$

**Pavni:** Now approximate that time integral — I remember the trapezoidal (averaging) rule.



**Acharya:** Exactly. Apply the trapezoidal rule to the integral on the right:

$$\int_{t^n}^{t^{n+1}} u_{xx}(x, t) dt \approx \frac{\Delta t}{2} (u_{xx}(x, t^n) + u_{xx}(x, t^{n+1}))$$

with pointwise quadrature error  $O(\Delta t^3)$  (so after division by  $\Delta t$  the time-discretization error is  $O(\Delta t^2)$ ).

**Pavni:** Then substitute that into the integrated PDE.

**Acharya:** We get

$$u(x, t^{n+1}) - u(x, t^n) = \alpha^2 \frac{\Delta t}{2} (u_{xx}(x, t^n) + u_{xx}(x, t^{n+1})) + O(\Delta t^3).$$

**Pavni:** Next we discretize  $u_{xx}$  in space with the centered second difference.

**Acharya:** Right. At grid point  $x_j$ ,

$$u_{xx}(x_j, t^\ell) \approx \frac{U_{j-1}^\ell - 2U_j^\ell + U_{j+1}^\ell}{(\Delta x)^2},$$

which in vector form for interior nodes is  $\frac{1}{(\Delta x)^2} AU^\ell$ , where  $A$  is the usual tridiagonal Laplacian (stencil  $[1, -2, 1]$ ).

Substituting gives (collecting interior points into  $U^n$ )

$$U^{n+1} - U^n = \frac{\alpha^2 \Delta t}{2(\Delta x)^2} (AU^n + AU^{n+1}) + (\text{truncation terms}).$$

**Pavni:** Introduce the nondimensional parameter  $\lambda$ ?

**Acharya:** Yes, set  $\lambda = \frac{\alpha^2 \Delta t}{(\Delta x)^2}$ . Then

$$U^{n+1} - U^n = \frac{\lambda}{2} (AU^n + AU^{n+1}) + (\text{truncation terms}).$$

Bring  $U^{n+1}$  terms to the left to obtain

$$\left(I - \frac{\lambda}{2} A\right) U^{n+1} = \left(I + \frac{\lambda}{2} A\right) U^n.$$

Now, let's write these matrices explicitly. The left-hand side matrix is

$$L = I - \frac{\lambda}{2} A = \begin{bmatrix} 1 + \lambda & -\frac{\lambda}{2} & 0 & \cdots & 0 \\ -\frac{\lambda}{2} & 1 + \lambda & -\frac{\lambda}{2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\frac{\lambda}{2} & 1 + \lambda & -\frac{\lambda}{2} \\ 0 & \cdots & 0 & -\frac{\lambda}{2} & 1 + \lambda \end{bmatrix}.$$



The right-hand side matrix is

$$R = I + \frac{\lambda}{2}A = \begin{bmatrix} 1-\lambda & \frac{\lambda}{2} & 0 & \cdots & 0 \\ \frac{\lambda}{2} & 1-\lambda & \frac{\lambda}{2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \frac{\lambda}{2} & 1-\lambda & \frac{\lambda}{2} \\ 0 & \cdots & 0 & \frac{\lambda}{2} & 1-\lambda \end{bmatrix}.$$

So the Crank–Nicolson scheme can be written as

$$\boxed{L U^{n+1} = R U^n}.$$

**Pavni:** Great — and the eigenvalue form and stability follow from the spectral map of  $A$ .

**Acharya:** Right — each eigenvalue  $\alpha_k$  of  $A$  is mapped to

$$\mu_k = \frac{1 + \frac{\lambda}{2}\alpha_k}{1 - \frac{\lambda}{2}\alpha_k},$$

and substituting  $\alpha_k = -4 \sin^2(\frac{k\pi}{2(N+1)})$  yields

$$\mu_k = \frac{1 - 2\lambda \sin^2(\frac{k\pi}{2(N+1)})}{1 + 2\lambda \sin^2(\frac{k\pi}{2(N+1)})}.$$

This shows  $|\mu_k| \leq 1$ , so CN is unconditionally stable.

**i** Crank–Nicolson: derivation summary & stability

**Starting point (integrated PDE):**

$$u(x, t^{n+1}) - u(x, t^n) = \alpha^2 \int_{t^n}^{t^{n+1}} u_{xx}(x, t) dt.$$

**Trapezoidal rule (time) + centered difference (space):**

$$\int_{t^n}^{t^{n+1}} u_{xx} dt \approx \frac{\Delta t}{2} (u_{xx}(t^n) + u_{xx}(t^{n+1})), \quad u_{xx}(x_j, t^\ell) \approx \frac{(AU^\ell)_j}{(\Delta x)^2}.$$

**Matrix form (with  $\lambda = \frac{\alpha^2 \Delta t}{(\Delta x)^2}$ ):**

$$\boxed{\left(I - \frac{\lambda}{2}A\right)U^{n+1} = \left(I + \frac{\lambda}{2}A\right)U^n}.$$



**Eigenvalues (mode-wise):** if  $s_k = \sin(\frac{k\pi}{2(N+1)})$  then

$$\mu_k = \frac{1 - 2\lambda s_k^2}{1 + 2\lambda s_k^2}, \quad k = 1, \dots, N.$$

**Stability check (short):** - For  $\lambda \geq 0$  and  $0 \leq s_k^2 \leq 1$  the denominator  $1 + 2\lambda s_k^2 > 0$ .

- Hence

$$|\mu_k| = \frac{|1 - 2\lambda s_k^2|}{1 + 2\lambda s_k^2} \leq 1,$$

and for nontrivial modes with  $\lambda > 0$  strict inequality holds.

**Conclusion:** Crank–Nicolson is **unconditionally stable** and second-order accurate in time and space (global error  $O(\Delta t^2 + \Delta x^2)$ ).

Unable to display output for mime type(s): text/html

**How to interact:**

- Move the slider to vary  $\lambda = \frac{\alpha^2 \Delta t}{(\Delta x)^2}$ .
- Hover on a marker to see the exact  $(k, \mu_k)$ .
- Zoom by dragging, double-click to reset.

**Observe:** all  $\mu_k$  satisfy  $|\mu_k| \leq 1$  for every  $\lambda \geq 0$ , confirming unconditional stability; for small  $\lambda$  the modes are near 1, for large  $\lambda$  high-frequency modes move nearer to  $-1$  but remain bounded in magnitude by 1.