# Crank-Nicolson-Julia-1

March 25, 2025

## 1 Derivation of the Crank–Nicolson Method

This notebook cell outlines a step-by-step derivation of the Crank–Nicolson scheme for the 1D heat equation:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < L, \quad t > 0.$$

---

### 1.1 1. The Heat Equation

We consider the one-dimensional heat (diffusion) equation on ([0, L]). We partition the spatial domain into (N) intervals of width $\Delta x = \frac{L}{N}$, so

$$x_i = i\,\Delta x, \quad i = 0, 1, \ldots, N.$$

We partition the time domain into steps of size $\Delta t$, so

$$t^n = n\,\Delta t, \quad n = 0, 1, 2, \ldots.$$

We denote

$$u_i^n \approx u(x_i, t^n).$$

---

### 1.2 2. Discretize the Time Derivative

Approximate the time derivative by a forward difference:

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

---

### 1.3 3. Discretize the Space Derivative

Approximate the second spatial derivative by a central difference:

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

---

## 1.4 4. Crank–Nicolson: Averaging Explicit and Implicit

Crank–Nicolson is formed by **averaging** the spatial derivatives at time (n) (explicit) and (n+1) (implicit). So,

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

---

## 1.5 5. Introduce the Mesh Ratio

Define the mesh ratio

$$r \;=\; \frac{\alpha\,\Delta t}{(\Delta x)^2}.$$

Multiply both sides of the Crank–Nicolson equation by $\Delta t$ and collect like terms:

$$u_i^{n+1} - u_i^n = \frac{r}{2}\Big[(u_{i+1}^n - 2\,u_i^n + u_{i-1}^n) + (u_{i+1}^{n+1} - 2\,u_i^{n+1} + u_{i-1}^{n+1})\Big].$$

After rearranging, we get:

$$-\frac{r}{2}\,u_{i-1}^{n+1} \;+\; \big(1+r\big)\,u_i^{n+1} \;-\; \frac{r}{2}\,u_{i+1}^{n+1} = \frac{r}{2}\,u_{i-1}^n \;+\; \big(1-r\big)\,u_i^n \;+\; \frac{r}{2}\,u_{i+1}^n.$$

---

## 1.6 6. Tridiagonal System Form

For $i = 1, 2, \ldots, N-1$, this defines a **tridiagonal linear system** in the unknowns $\{u_i^{n+1}$. Denote the left-hand side as $A\mathbf{u}^{n+1}$ and the right-hand side as $B\mathbf{u}^n$. We can solve it at each time step using the **Thomas algorithm**:

$$A\,\mathbf{u}^{n+1} = B\,\mathbf{u}^n.$$

---

## 1.7 7. Final Scheme

Hence, the Crank–Nicolson scheme for the 1D heat equation is:

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

subject to your chosen boundary conditions (Dirichlet, Neumann, etc.) at (i = 0) and (i = N).

### 1.8 Matrix Form

We can write this entire system for (i = 1, 2, . . . , N-1) in **matrix form** as:

$$A\,\mathbf{u}^{n+1} \;=\; B\,\mathbf{u}^{n},$$

where $\mathbf{u}^n$ is the vector $\left[u_1^n,\; u_2^n,\; \dots,\; u_{N-1}^n\right]^T$,
and (A) and (B) are **tridiagonal** matrices of size $(N-1) \times (N-1)$.

#### 1.8.1 The Matrix (A)

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

#### 1.8.2 The Matrix (B)

$$B = \begin{bmatrix} 1-r & \frac{r}{2} & 0 & \cdots & 0 \\ \frac{r}{2} & 1-r & \frac{r}{2} & \ddots & \vdots \\ 0 & \frac{r}{2} & 1-r & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \frac{r}{2} \\ 0 & \cdots & 0 & \frac{r}{2} & 1-r \end{bmatrix}.$$

Hence, the **Crank–Nicolson update** for each time step is:

$$\mathbf{u}^{n+1} = A^{-1} B\,\mathbf{u}^{n},$$

though in practice we do **not** invert (A) directly; rather, we solve the linear system

$$A\,\mathbf{u}^{n+1} = B\,\mathbf{u}^{n}$$

efficiently using the **Thomas algorithm** (a specialized (O(N)) Gaussian elimination for tridiagonal matrices).

---

**Key points:** - This is **second-order accurate** in both time and space. - **Unconditionally stable** for linear diffusion problems. - Solved via a tridiagonal linear system at each timestep.

## 2 Thomas Algorithm: Step-by-Step Derivation

We want to solve a **tridiagonal linear system** of $n$ equations:

$$
\begin{aligned}
b_1\,x_1 \;+\; c_1\,x_2 &= d_1,\\
a_2\,x_1 \;+\; b_2\,x_2 \;+\; c_2\,x_3 &= d_2,\\
a_3\,x_2 \;+\; b_3\,x_3 \;+\; c_3\,x_4 &= d_3,\\
&\;\vdots\\
a_n\,x_{n-1} \;+\; b_n\,x_n &= d_n.
\end{aligned}
$$

Here: - $a_i$ are the subdiagonal entries ($i = 2, \ldots, n$; often we take $a_1 = 0$ by convention), - $b_i$ are the main diagonal entries ($i = 1, \ldots, n$), - $c_i$ are the superdiagonal entries ($i = 1, \ldots, n-1$; often $c_n = 0$ by convention), - $d_i$ are the constants on the right-hand side.

Our goal is to find $x_1, x_2, \ldots, x_n$ in $\mathcal{O}(n)$ time. The procedure involves two phases:

1. **Forward Elimination** – to remove the subdiagonal $a_i$.
2. **Back Substitution** – to solve the resulting upper-triangular system.

---

## 2.1 1. Forward Elimination

We sequentially eliminate $a_2, a_3, \ldots, a_n$ by modifying each row in turn.

### 2.1.1 Step 1 (Row 1)

The first equation is

$$b_1\,x_1 \;+\; c_1\,x_2 \;=\; d_1.$$

We **normalize** this row by dividing through by $b_1$ (assuming $b_1 \neq 0$):

$$x_1 \;+\; \frac{c_1}{b_1}\,x_2 \;=\; \frac{d_1}{b_1}.$$

Define the **modified** superdiagonal and right-hand side for row 1:

$$c'_1 \;=\; \frac{c_1}{b_1}, \quad d'_1 \;=\; \frac{d_1}{b_1}.$$

Hence the first row (in **modified** form) is:

$$x_1 \;+\; c'_1\,x_2 \;=\; d'_1.$$

### 2.1.2 Step 2 (Row 2)

The second equation in the original system is

$$a_2\,x_1 \;+\; b_2\,x_2 \;+\; c_2\,x_3 \;=\; d_2.$$

We want to **eliminate** $x_1$ from this equation using the (already normalized) Row 1.

1. Multiply the (modified) Row 1 by $a_2$ and **subtract** from Row 2:

$$a_2\Big(x_1 + c_1'\,x_2\Big) = a_2\,d_1'.$$

   Subtracting, we get:

$$(b_2 - a_2\,c_1')\,x_2 \;+\; c_2\,x_3 \;=\; d_2 - a_2\,d_1'.$$

2. Define

$$\begin{aligned} b_2' \;&=\; b_2 \;-\; a_2\,c_1', \\ d_2' \;&=\; d_2 \;-\; a_2\,d_1'. \end{aligned}$$

   So the second equation becomes

$$b_2'\,x_2 \;+\; c_2\,x_3 \;=\; d_2'.$$

3. **Normalize** this row by dividing through by $b_2'$:

$$x_2 \;+\; \frac{c_2}{b_2'}\,x_3 \;=\; \frac{d_2'}{b_2'}.$$

   Define:

$$c_2' = \frac{c_2}{b_2'}, \quad d_2' = \frac{d_2'}{b_2'}.$$

Hence Row 2 is now:

$$x_2 \;+\; c_2'\,x_3 = d_2'.$$

#### 2.1.3 Step 3 (General Row $i$)

For $i = 3, 4, \ldots, n$, we proceed **similarly**:

1. The $i$-th equation (before modification) is

$$a_i\,x_{i-1} \;+\; b_i\,x_i \;+\; c_i\,x_{i+1} = d_i.$$

2. We have already normalized Row $i-1$, so we can subtract $a_i$ times that row to eliminate $x_{i-1}$. Symbolically:

$$b_i' \;=\; b_i \;-\; a_i\,c_{i-1}', \quad d_i' \;=\; d_i \;-\; a_i\,d_{i-1}'.$$

3. **Normalize** Row $i$ by dividing through by $b'_i$:

$$c'_i = \frac{c_i}{b'_i}, \quad d'_i = \frac{d'_i}{b'_i}.$$

After processing row $n$, the subdiagonal entries are effectively **eliminated**. The system is now **upper-triangular**, represented by:

$$x_i + c'_i \, x_{i+1} = d'_i, \quad (i = 1, 2, \ldots, n-1).$$

For the last equation ($i = n$), it simply becomes:

$$x_n = d'_n.$$

---

## 2.2 2. Back Substitution

Now we solve for the $x_i$ **from the bottom up**:

1. **Initialize** the solution at the last row:

$$x_n = d'_n.$$

2. **For** $i = n-1, n-2, \ldots, 1$:

$$x_i = d'_i - c'_i \, x_{i+1}.$$

This recovers all the unknowns $x_1, \ldots, x_n$ in a single **backward** pass.

---

## 2.3 3. Summary of the Thomas Algorithm

- **Forward Elimination**:
  For $i = 1$ to $n$:
  1. Normalize the current row by its diagonal ($b'_i$).
  2. Use the normalized row to eliminate $a_{i+1}$ in the next row.
- **Back Substitution**:
  Start from $x_n = d'_n$ and move backward using $x_i = d'_i - c'_i \, x_{i+1}$.

### 2.3.1 Complexity

Because each step uses only a few arithmetic operations per row, the **Thomas algorithm** runs in $\mathcal{O}(n)$ time, much faster than $\mathcal{O}(n^3)$ for generic Gaussian elimination.

---

## 2.4 4. Final Formulas

Putting it all together, if we define:

$$
\begin{aligned}
&\text{(Initialization)} && c_1' = \frac{c_1}{b_1}, && d_1' = \frac{d_1}{b_1}, \\
&\text{(Forward sweep for } i = 2, \dots, n) && b_i' = b_i - a_i\, c_{i-1}', && d_i' = d_i - a_i\, d_{i-1}', \\
& && c_i' = \frac{c_i}{b_i'}, && d_i' = \frac{d_i'}{b_i'}, \\
&\text{(Back Substitution)} && x_n = d_n', && x_i = d_i' - c_i'\, x_{i+1} \quad \bigl(i = n-1, \dots, 1\bigr),
\end{aligned}
$$

then $\{x_i\}$ is the unique solution of the original **tridiagonal system**.

**That** is the Thomas algorithm in a **step-by-step** derivation, showing how we systematically eliminate subdiagonal terms and then back-substitute.

---

[29]:
```
using LinearAlgebra
using PyPlot
```

[30]:
```
function solve_by_thomas_algorithm(a,b,c,d)
    """
    Solve a tridiagonal system A x = d using the Thomas algorithm.
    a, b, c are the lower, main, and upper diagonals (each 1D arrays).
    d is the right-hand side array.

    Returns the solution array x.
    """
    N = length(b)
    bp = zeros(Float64,N)
    cp = zeros(Float64,N)
    dp = zeros(Float64,N)
    xs = zeros(Float64,N)
    cp[1] = c[1]/b[1]; dp[1] = d[1]/b[1]
    for i = 2:N
        bp[i] = b[i] - a[i]*cp[i-1]
        dp[i] = d[i] - a[i]*dp[i-1]
        cp[i] = c[i]/bp[i]
        dp[i] = dp[i]/bp[i]
    end
    xs[end] = dp[end]
    for i=N-1:-1:1
        xs[i] = dp[i] - cp[i]*xs[i+1]
    end
    return xs
end
```

```
"""
params = α, L, Nx, Nt, Δt, u_boundary, u_ini, r
"""
function solve_heat_equation(params)
    """
    Solve the 1D heat equation using the Crank-Nicolson method with Thomas␣
 ↪algorithm.

    params = (α, L, Nx, Nt, Δt, u_boundary, u_ini, r)
      α           -> Thermal diffusivity
      L           -> Length of the domain
      Nx          -> Number of spatial grid points
      Nt          -> Number of time steps
      Δt          -> Time step size
      u_boundary  -> Tuple (u_left, u_right) for Dirichlet BCs
      u_ini       -> Initial temperature profile (1D numpy array of length Nx)
      r           -> α*Δt / (Δx^2)

    Returns:
      ts        -> 1D array of time values
      solutions -> 2D array of shape (Nx, Nt+1),
                   where solutions[:, k] is the solution at time step k.
    """

    # Retrieve the parameters
    α, L, Nx, Nt, Δt, u_boundary, u_ini, r = params

    # Load intial values
    u = u_ini

    # Define time points for simulation
    ts = range(0.0, length=Nt, step=Δt)

    # Variable for storing the solution (Nt+1 as we also store initial value)
    solutions = zeros(Float64,Nx,Nt+1)

    # Time loop
    for i=1:Nt

        # Store
        solutions[:,i] = u

        # For RHS
        d = zeros(Float64,Nx)
        for i=1:Nx
            d[i] = (1-r)*u[i]
```

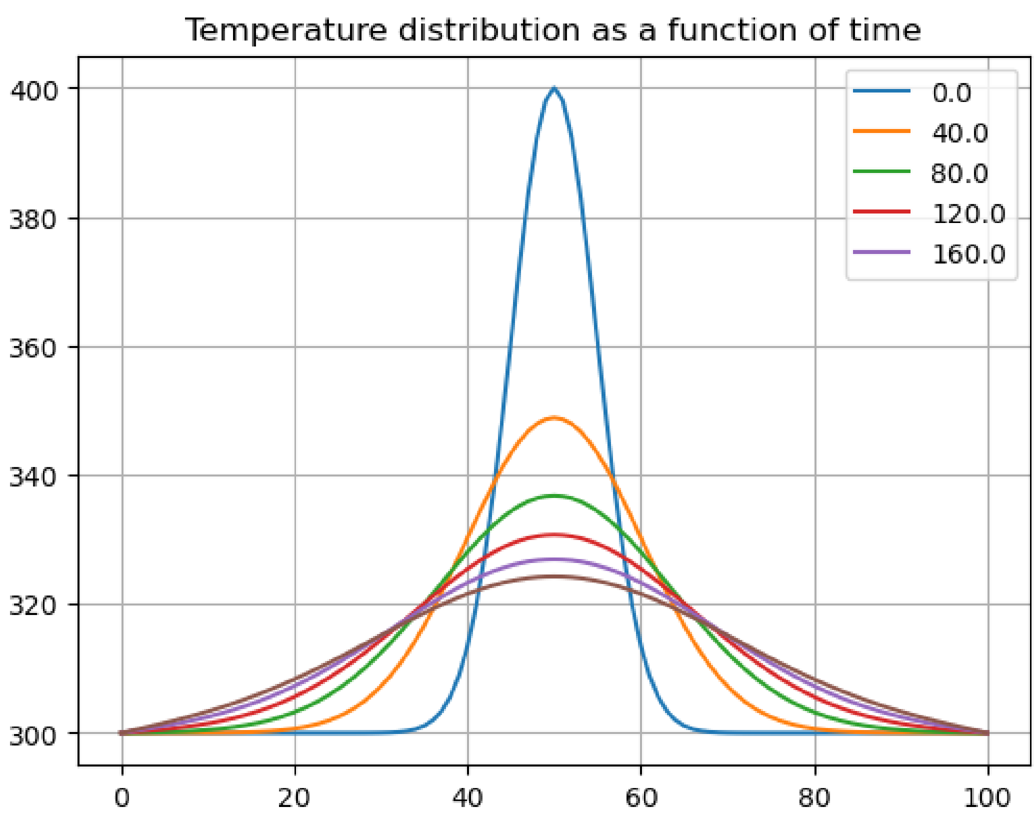
```julia
            if i>1
                d[i] += r/2*u[i-1]
            end
            if i<Nx
                d[i] += r/2*u[i+1]
            end
        end

        # Adjust Boundary conditions
        d[1] += r*u_boundary[1]
        d[end] += r*u_boundary[end]

        # For LHS
        b = (1+r)*ones(Nx)
        a = [0; -r/2*ones(Nx-1)]
        c = [-r/2*ones(Nx-1); 0]

        u = solve_by_thomas_algorithm(a,b,c,d)

        # Fix bounday values
        u[1], u[end] = u_boundary
    end

    # Store the last solution
    solutions[:,Nt+1] = u;

    # Return time points and solutions
    return ts, solutions
end
```

[30]: solve_heat_equation

[33]:
```julia
α = 1.0e-4
L = 1.0
Nx = 101
Nt = 2000
Δt = 0.1
σ = 0.05
u_boundary = (300.0,300.0)

Δx = L/(Nx-1)
r = α*Δt/(Δx*Δx)

xs = range(0.0, L, length=Nx)

# Initial condition: Gaussian bump over 300 K
```

```
u_ini = @. 300 + 100*exp(-(xs - L/2)^2/2/σ^2)

# Pack trial parameters
params = α, L, Nx, 2, Δt, u_boundary, u_ini, r

# Compile
@time ts, solutions = solve_heat_equation(params);

# Pack parameters
params = α, L, Nx, Nt, Δt, u_boundary, u_ini, r

# Final run
@time ts, solutions = solve_heat_equation(params);

# Plot
plot(solutions[:,1:400:end]);legend(ts[1:400:end])
title("Temperature distribution as a function of time")
grid()
```

```
0.000020 seconds (34 allocations: 25.656 KiB)
0.004092 seconds (26.01 k allocations: 23.759 MiB)
```



```
[ ]:
```