Computational Practicum Q2 – Lecture 3: Numerical methods for partial differential equations (Part 2)

Zoë J.G. Gromotka, Artur M. Schweidtmann, Ferdinand Grozema, Tanuj Karia

With support from Giacomo Lastrucci

Computational Practicum

Dept. Chemical Engineering
Delft University of Technology

Wednesday, 26th November 2025

TU Delft

Process Intelligence RESEARCH

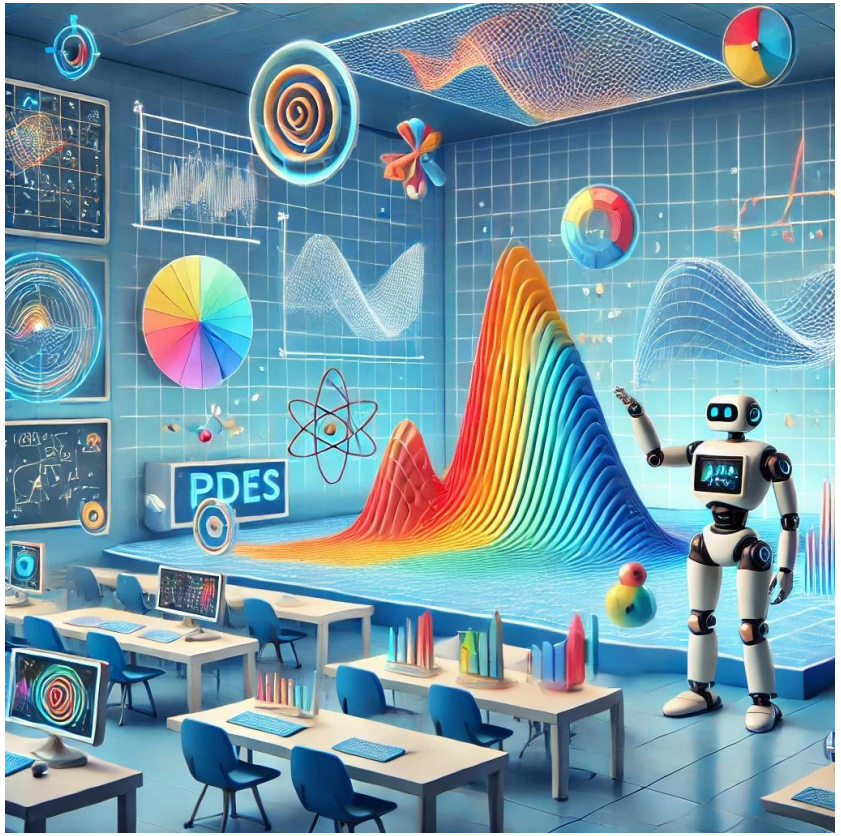
Delft Institute of Applied Mathematics

Reminder: Register to the Q2 Exam!

- Don’t forget to register for the Q2 Exam!
  Date: 13th of January 2026
- You can find instructions on how to register here.
- The registration is already opened!
- Remember to hand-in at least 5/7 assignment in Q2 to participate in the exam!
- If you didn’t pass Q1 Exam, you can still join the Q2 Exam!

Recap last lecture

- Introduction to partial differential equations (PDEs)
- Discretization for multivariate domains
- Parabolic PDEs: 1D Heat Equation
- Semi-discrete method for Parabolic PDEs:
  - Finite difference method
  - Handling boundary conditions w/ and w/o ghost points
  - Forward and backward Euler
  - Stability



Learning objectives

After successfully completing this lecture, you are able to…

- Explain what are elliptic PDEs and their applications in (chemical) engineering
- Implement numerical solution approaches for elliptic PDEs in Python
- Implement efficient sparse matrices in Python using scipy package

Agenda

- Elliptic PDE: Laplace equation (Steady-state heat equation)
  - Boundary conditions at the corners
  - Finite difference method in 2D domain
  - Poisson matrix
- Implementation of sparse matrices using Scipy

Agenda

- Elliptic PDE: Laplace equation (Steady-state heat equation)
  - Boundary conditions at the corners
  - Finite difference method in 2D domain
  - Poisson matrix
- Implementation of sparse matrices using Scipy

Introduction to elliptic PDE

- Elliptic PDEs describe systems that have already reached the steady state and thus are time-independent
  - The two independent variables are 2 spatial coordinates
- An example is the Laplace Equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \qquad 0 \leq x \leq L_x, \qquad 0 \leq y \leq L_y$$

- The Laplace Equation describes the spatial evolution of a quantity *u* in 2D through diffusion:
  - u is the temperature (or concentration, or velocity, etc.)
  - x and y are two spatial coordinates

Boundary conditions for elliptic PDEs

Specifically, for parabolic and elliptic PDEs:

- 2D elliptic PDE: $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$
  - 2x second order spatial derivatives → 4 boundary conditions

Heat transport problem

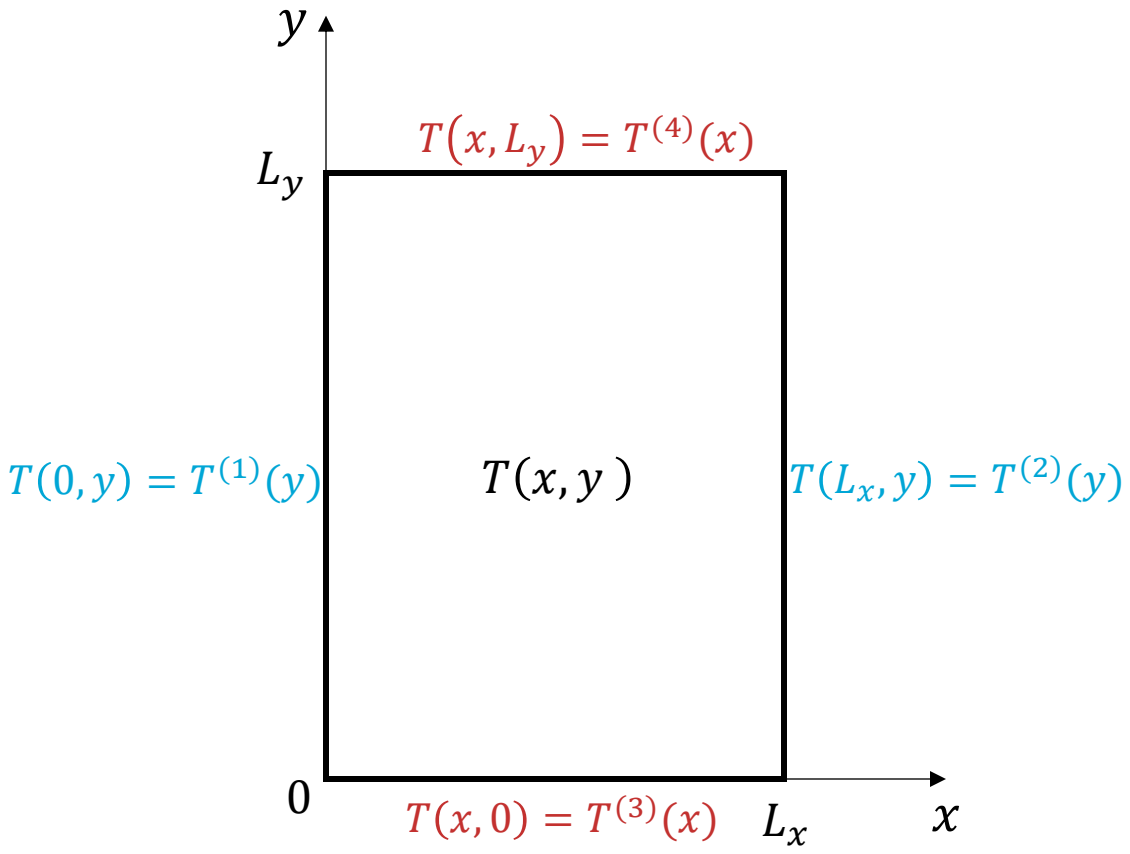
- Microscopic governing equations for transport phenomena are typically modelled by partial differential equations, describing how quantities as concentration, velocity or temperature evolve in space and time.
- We consider the case of heat transport problem as a case study for PDEs.
- More specifically:
  - Unsteady heat transport in 1D (e.g., tube wall)
    → Parabolic PDE (last lecture)
  - Steady state heat transport in 2D (e.g., plate heat exchanger)
    → Elliptic PDE (this lecture)

Case study for elliptic PDEs

- 2D unsteady heat equation:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0, \qquad 0 \le x \le L_x,\ 0 \le y \le L_y$$

- Consider heat transfer on a plate with negligible thickness:
  - The temperature on the edges is fixed and known
  - The system reached the steady state ($\frac{\partial T}{\partial t} = 0$)
  - How does the temperature field $T(x, y)$ is distributed across the 2D spatial domain?

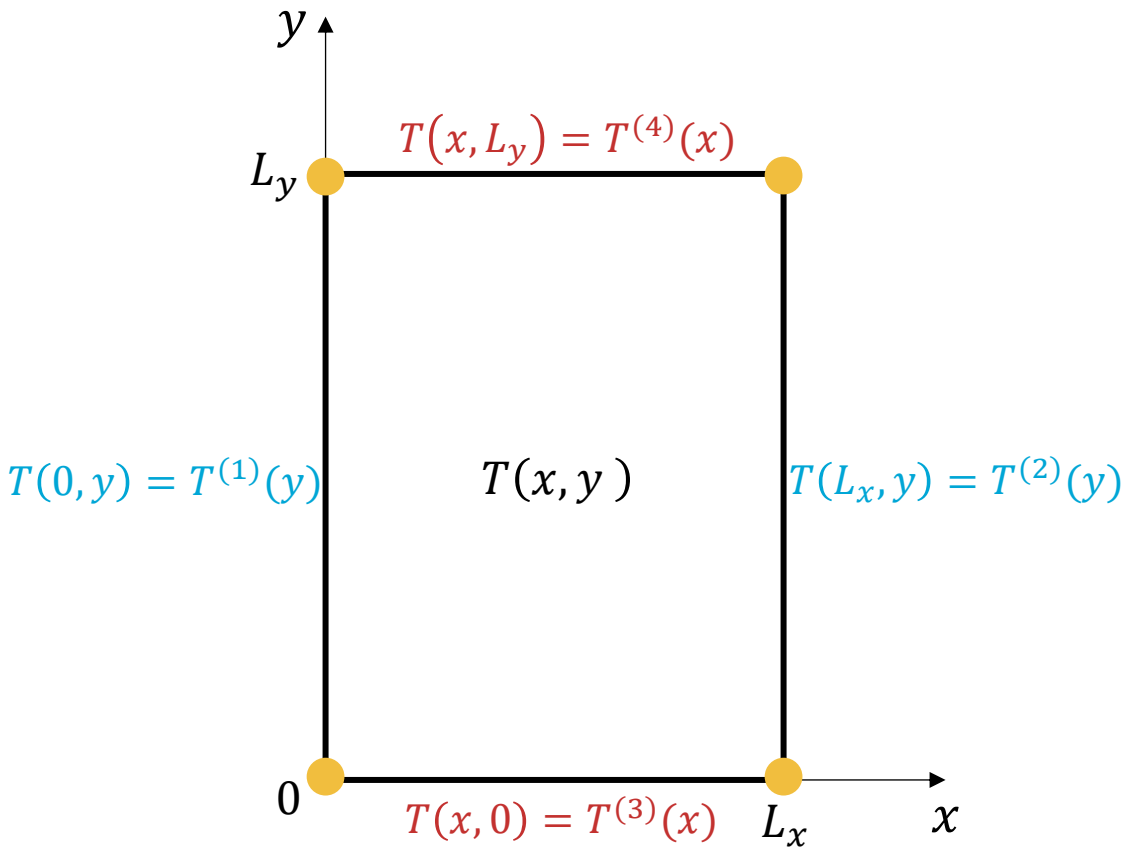


Handling boundary conditions in the corners Continuity

Ideal case:

- Boundary conditions (BCs) are continuous in the corners
- e.g., (bottom right corner)

$$T^{(1)}(y = 0) = T^{(3)}(x = 0)$$

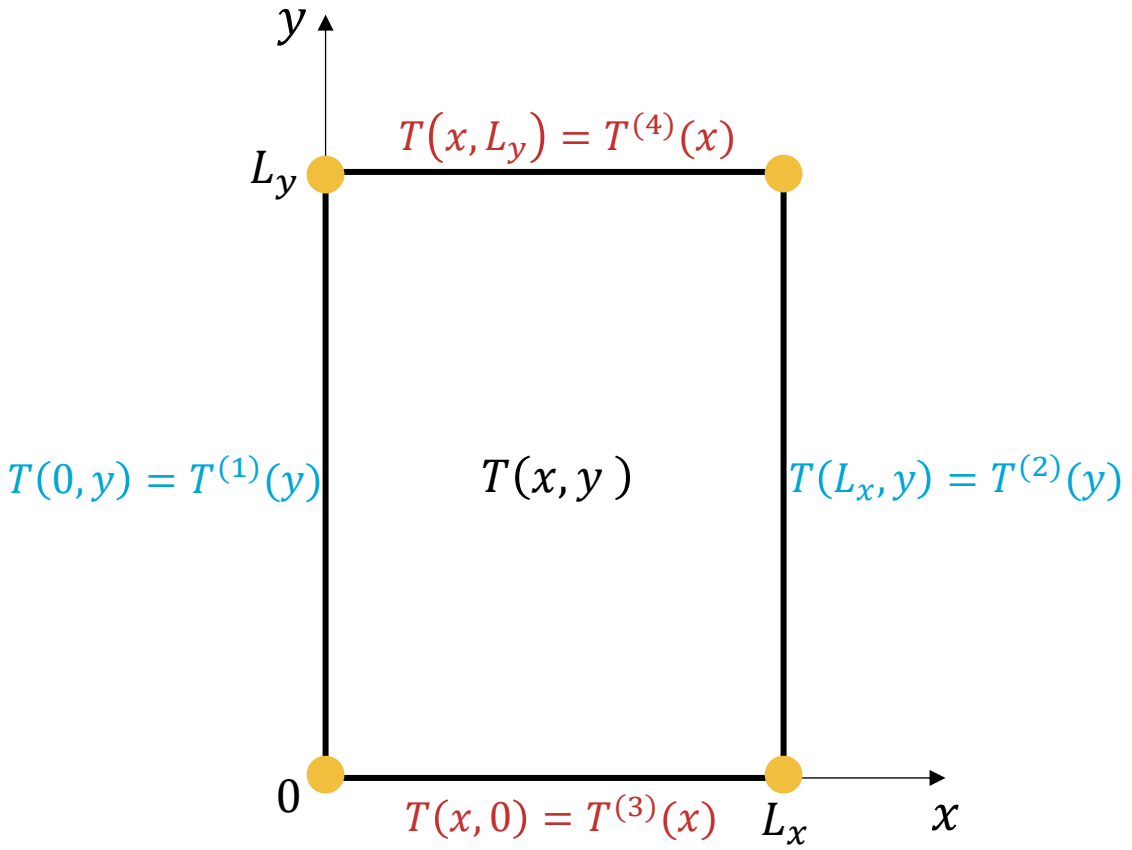


Handling boundary conditions in the corners Discontinuous BCs

For non-continuous BCs, one of the following approaches may be considered:

- **Dominance of one BC**
  - One BC is assigned to an open domain and the other on a closed domain
  - e.g.,
    $x \in (0, L_x) \rightarrow T^{(3)}(x)$ (open domain)
    $y \in [0, L_y] \rightarrow T^{(1)}(y)$ (closed domain)

→ In the corner (0,0), the condition $T^{(1)}(y)$ holds



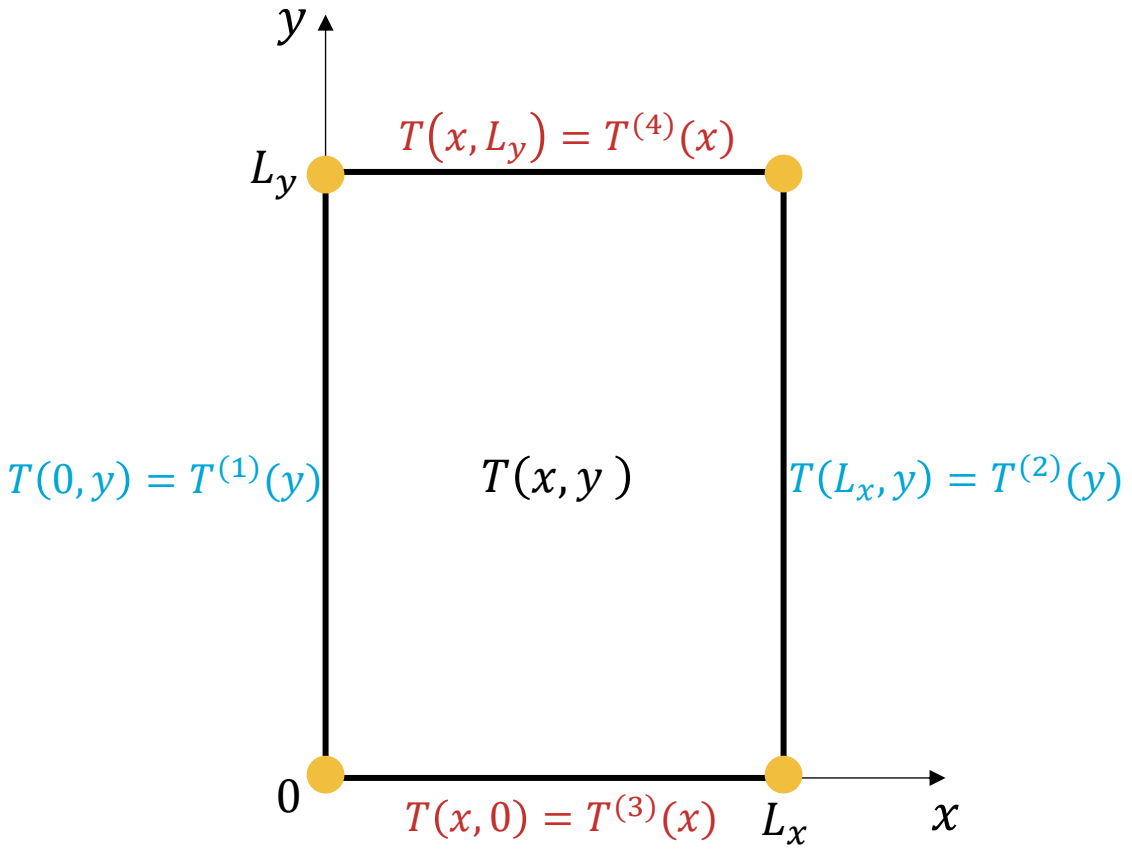
Handling boundary conditions in the corners Discontinuous BCs

Ideal case:

- Boundary conditions (BCs) are continuous in the corners

For non-continuous BCs, one of the following approaches may be considered:

- **Averaging**
  - Corner value is set as the average of the two conflicting conditions
  - Only applicable to Dirichlet BCs
  - e.g., $T(0,0) = \frac{1}{2}\left(T^{(1)}(y=0) + T^{(3)}(x=0)\right)$



Who needs superheroes when we have finite difference methods?

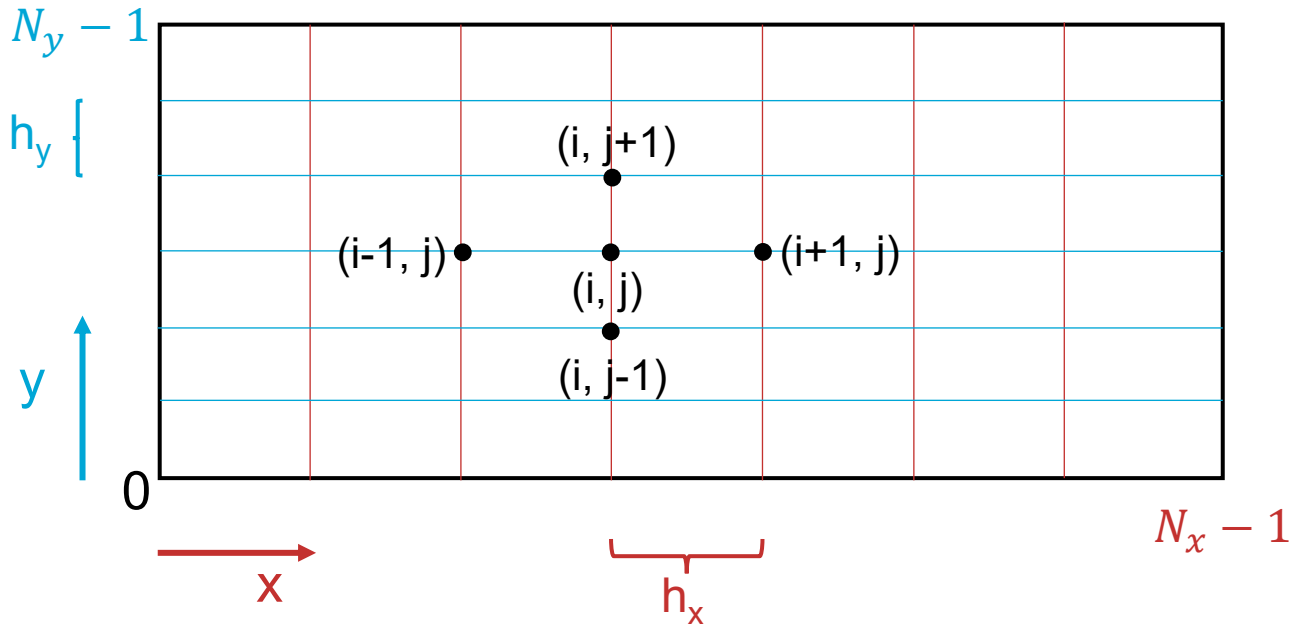
Agenda

- Elliptic PDE: Laplace equation (Steady-state heat equation)
  - Boundary conditions at the corners
  - Finite difference method in 2D domain
  - Poisson matrix
- Implementation of sparse matrices using Scipy

Recap: Discretize multivariate spatial domains

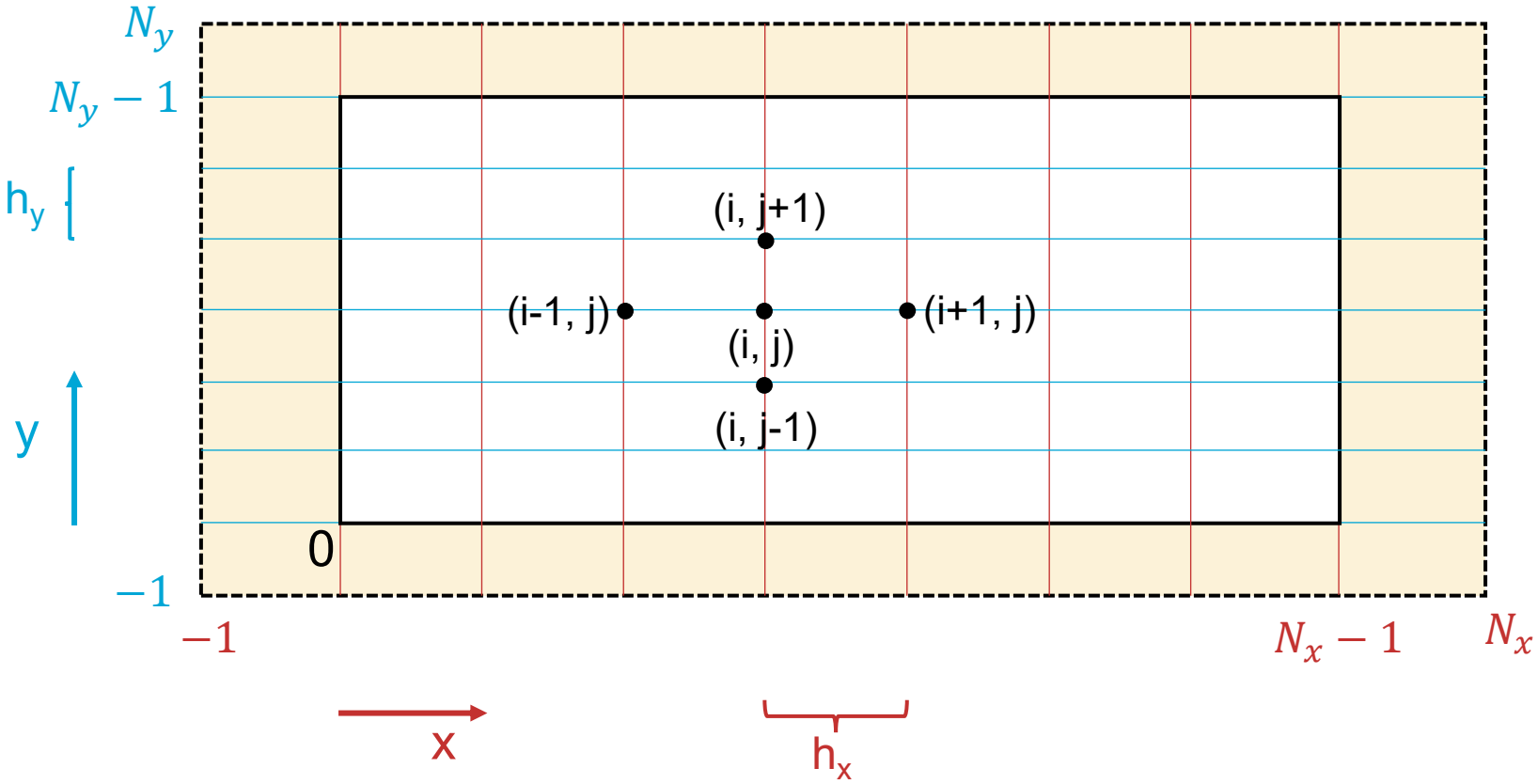
- By definition, PDEs involve multiple independent variables:
  - Space (x, y, z)
  - Time (t)
  - Possibly other variables

- Every variable will have its own (possibly different) discretization scheme
  - In general: $h_x \neq h_y$
  - Often: $h_x = h_y$ (considered in this lecture for notation simplicity)



Recap: Spatial discretization with ghost points

- Artificial (ghost) points can be added as a padding around the domain (for each coordinate: 2 points, 2 intervals)
- Useful to discretize boundary conditions with more accurate schemes

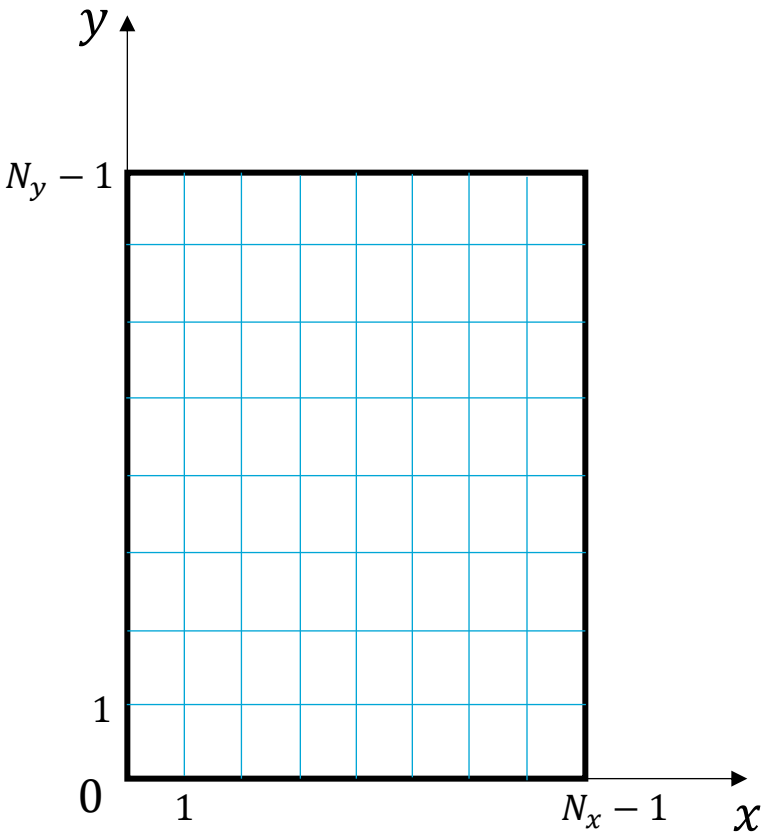


Finite difference method for elliptic PDEs – Discretization

- Finite difference method can be applied to discretize the second order derivatives:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

$$\frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta x^2} + \frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{\Delta y^2} = 0$$



Finite difference method for elliptic PDEs – Grid

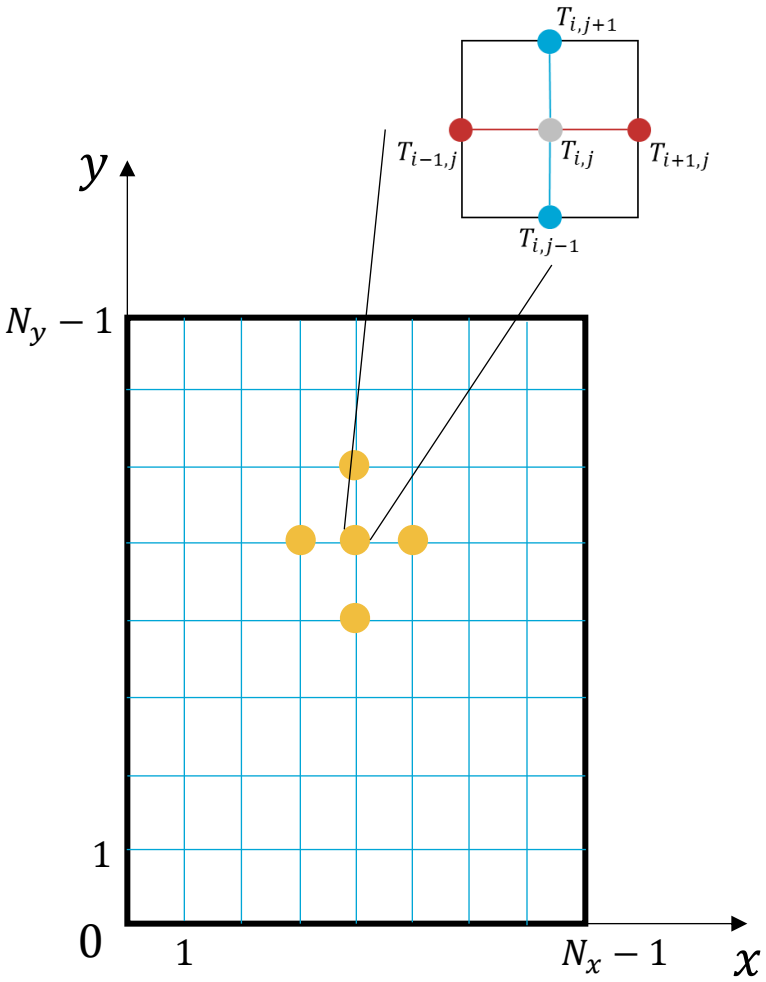
- Finite difference method can be applied to discretize the second order derivatives:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

$$\frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta x^2} + \frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{\Delta y^2} = 0$$

- Considering a homogeneous grid ($\Delta x = \Delta y$):

$$T_{i-1,j} + T_{i+1,j} + T_{i,j-1} + T_{i,j+1} - 4T_{i,j} = 0$$



Finite difference method for elliptic PDEs – Internal points

- Finite difference method can be applied to discretize the second order derivatives:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

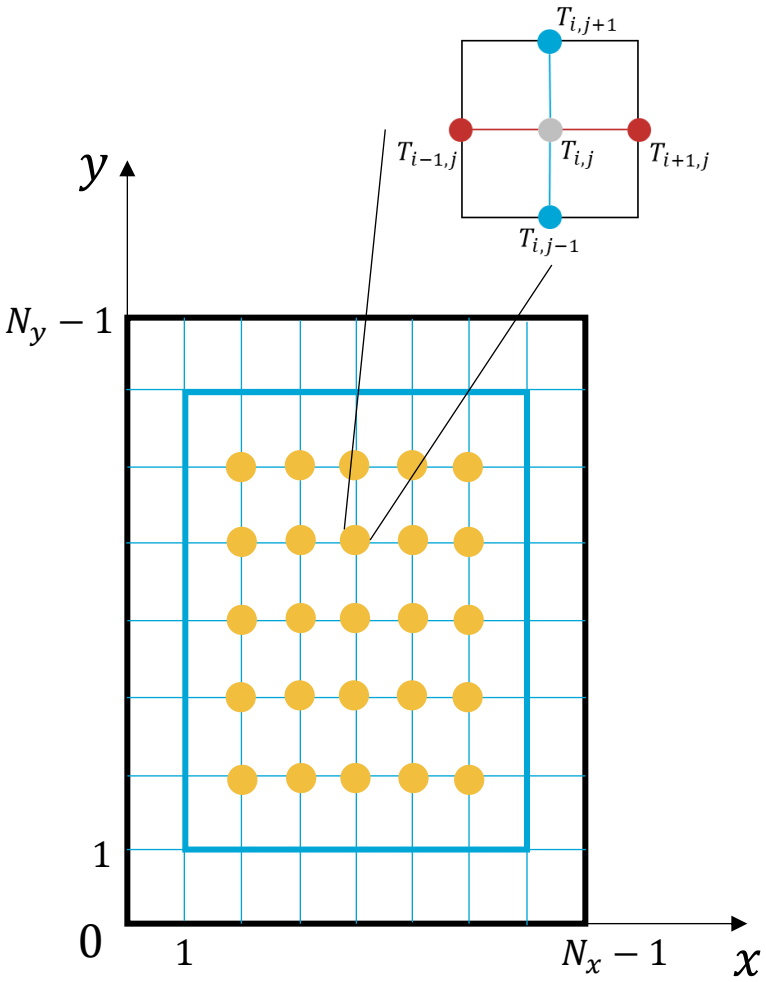
$$\frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{\Delta x^2} + \frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{\Delta y^2} = 0$$

- Considering a homogeneous grid ($\Delta x = \Delta y$):

$$T_{i-1,j} + T_{i+1,j} + T_{i,j-1} + T_{i,j+1} - 4T_{i,j} = 0$$

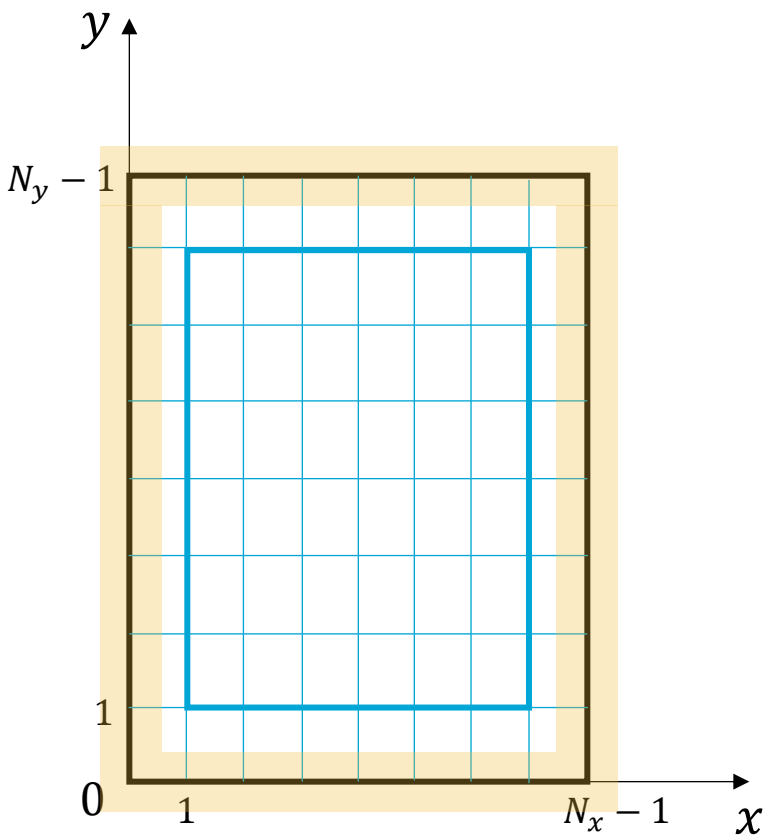
- This equation is valid for every internal point:

$$i = 2, \dots, N_x - 3; \quad j = 2, \dots, N_y - 3$$



Finite difference method for elliptic PDEs

- The boundary conditions are handled separately:
  - We develop *specialized* equations on the boundaries so that the system dimensionality will be $(N_x - 2) \times (N_y - 2)$



Dirichlet boundary conditions – Left boundary

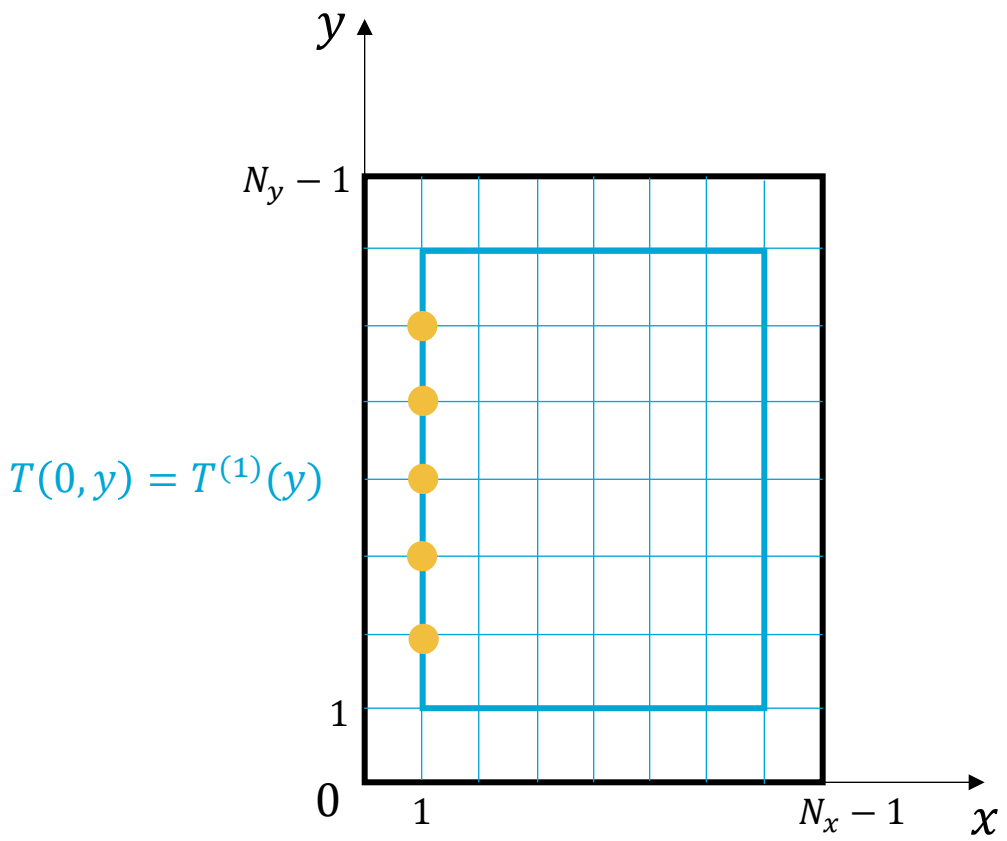
Considering the case study discussed earlier…

- **Dirichlet BC1:** $T(0, y) = T^{(1)}(y)$, where $T^{(1)}(y)$ is a function $f: \mathbb{R} \to \mathbb{R}$

We write specialized equations for the nodes $i = 1$, $j = 2, \ldots, N_y - 3$: (note, this are $N_y - 4$ equations)

$$T_{0,j} + T_{2,j} + T_{1,j-1} + T_{1,j+1} - 4T_{1,j} = 0$$

→ Eliminate boundary nodes



Dirichlet boundary conditions – Left boundary

Considering the case study discussed earlier…

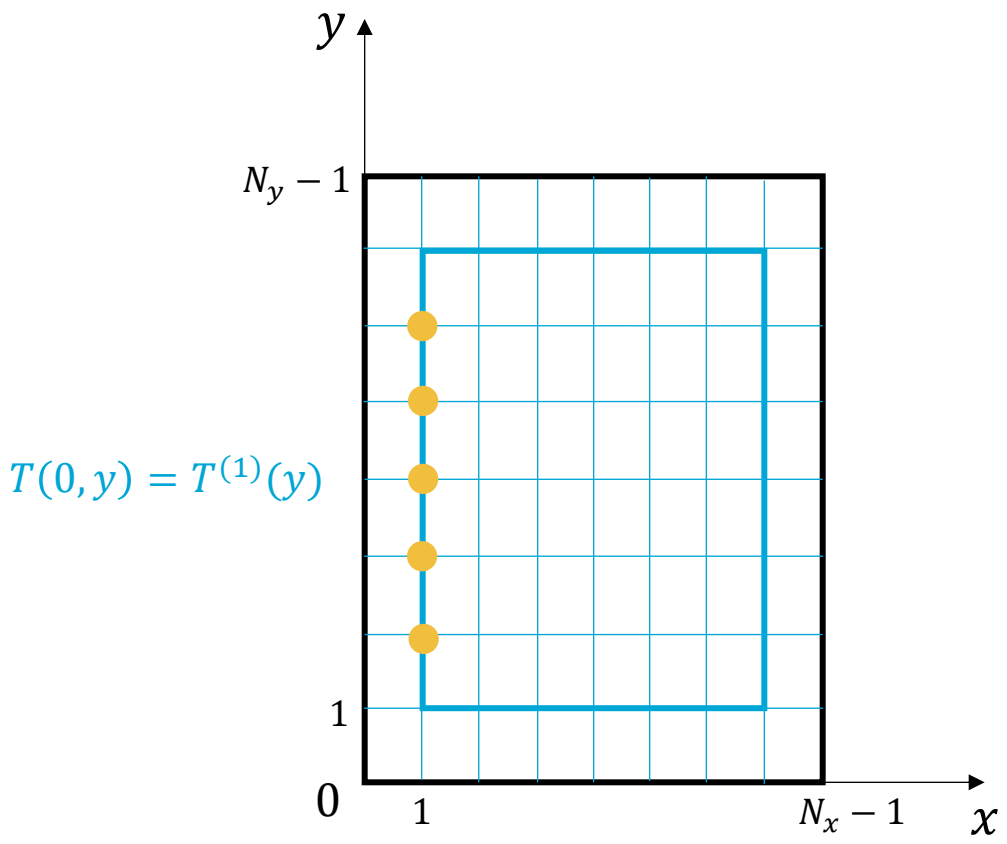
- **Dirichlet BC1:** $T(0, y) = T^{(1)}(y)$, where $T^{(1)}(y)$ is a function $f: \mathbb{R} \to \mathbb{R}$

We write specialized equations for the nodes $i = 1$, $j = 2, \ldots, N_y - 3$: (note, this are $N_y - 4$ equations)

$$T_{0,j} + T_{2,j} + T_{1,j-1} + T_{1,j+1} - 4T_{1,j} = 0$$

But $T_{0,j} = T_j^{(1)}$, which is given by Dirichlet BC1, then:

$$T_{2,j} + T_{1,j-1} + T_{1,j+1} - 4T_{1,j} = -T_j^{(1)}$$



Dirichlet boundary conditions – Right boundary

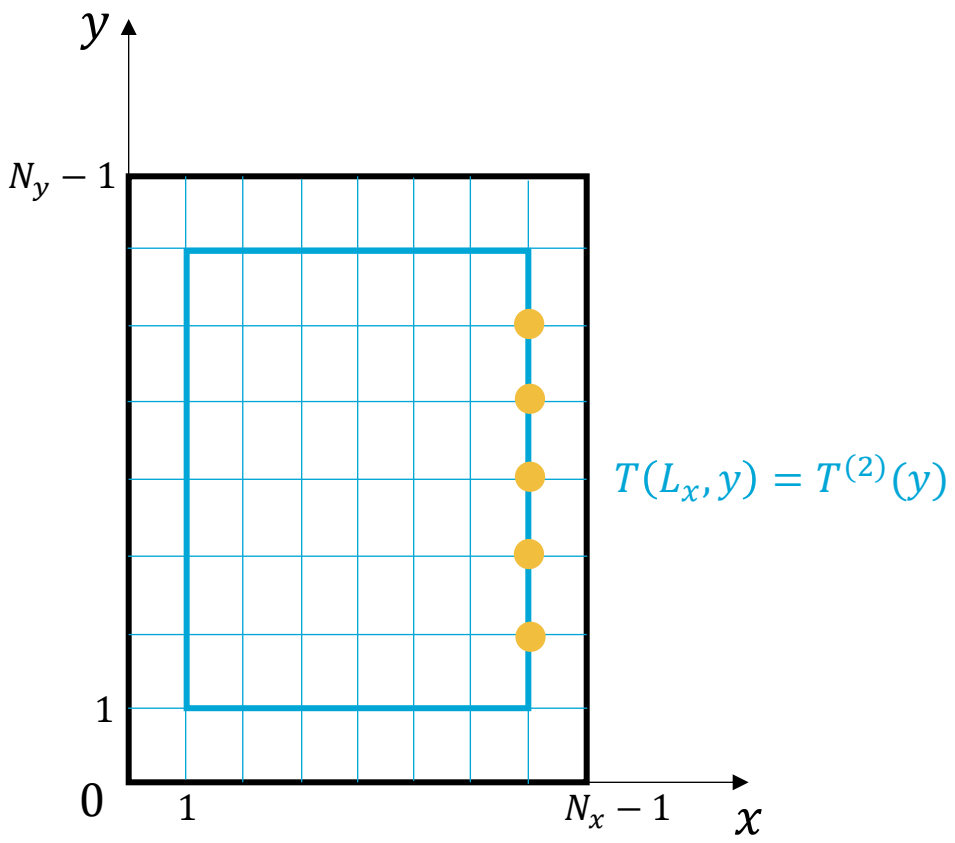
Considering the case study discussed earlier…

- **Dirichlet BC2:** $T(L_x, y) = T^{(2)}(y)$, where $T^{(2)}(y)$ is a function $f: \mathbb{R} \to \mathbb{R}$

We write a specialized equation for the nodes $i = N_x - 2$, $j = 2, \ldots, N_y - 3$: (note, this is $N_y - 4$ equations)

$$T_{N_x-3,j} + T_{N_x-1,j} + T_{N_x-2,j-1} + T_{N_x-2,j+1} - 4T_{N_x-2,j} = 0$$

→ Eliminate boundary nodes



Dirichlet boundary conditions – Right boundary

Considering the case study discussed earlier…

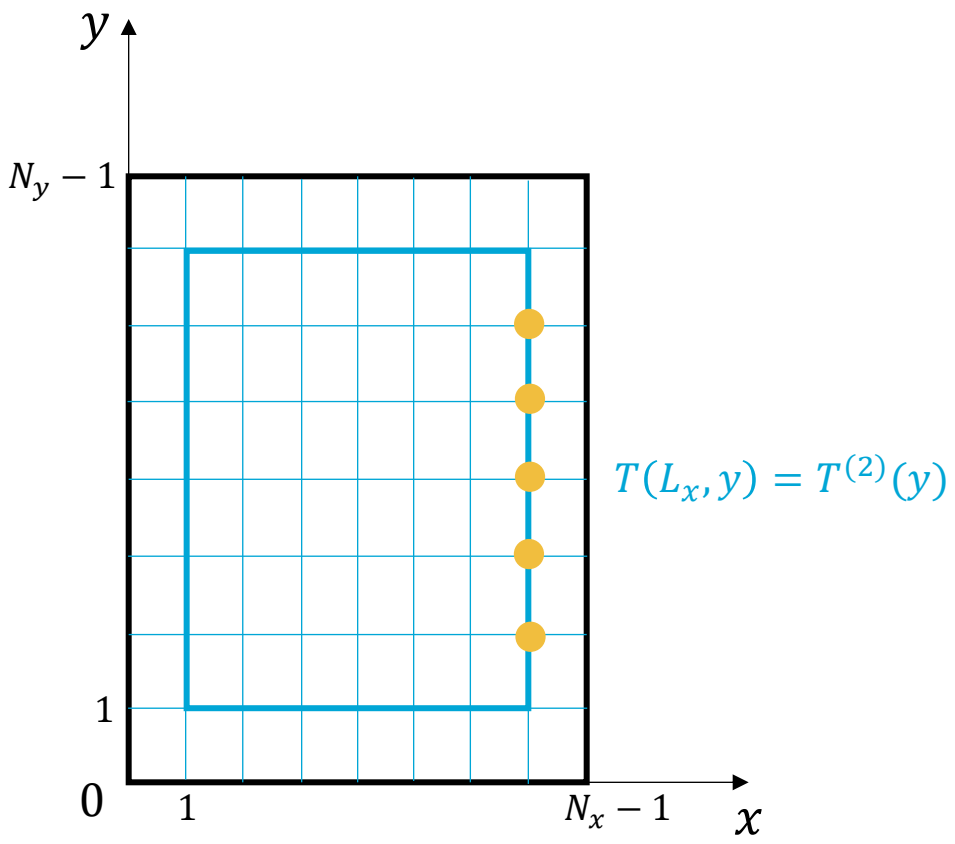
- **Dirichlet BC2:** $T(L_x, y) = T^{(2)}(y)$, where $T^{(2)}(y)$ is a function $f: \mathbb{R} \to \mathbb{R}$

We write a specialized equation for the nodes $i = N_x - 2$, $j = 2, \ldots, N_y - 3$: (note, this is $N_y - 4$ equations)

$$T_{N_x-3,j} + T_{N_x-1,j} + T_{N_x-2,j-1} + T_{N_x-2,j+1} - 4T_{N_x-2,j} = 0$$

But $T_{N_x-1,j} = T_j^{(2)}$, which is given by Dirichlet BC2, then:

$$T_{N_x-3,j} + T_{N_x-2,j-1} + T_{N_x-2,j+1} - 4T_{N_x-2,j} = -T_j^{(2)}$$



Dirichlet boundary conditions – Bottom boundary

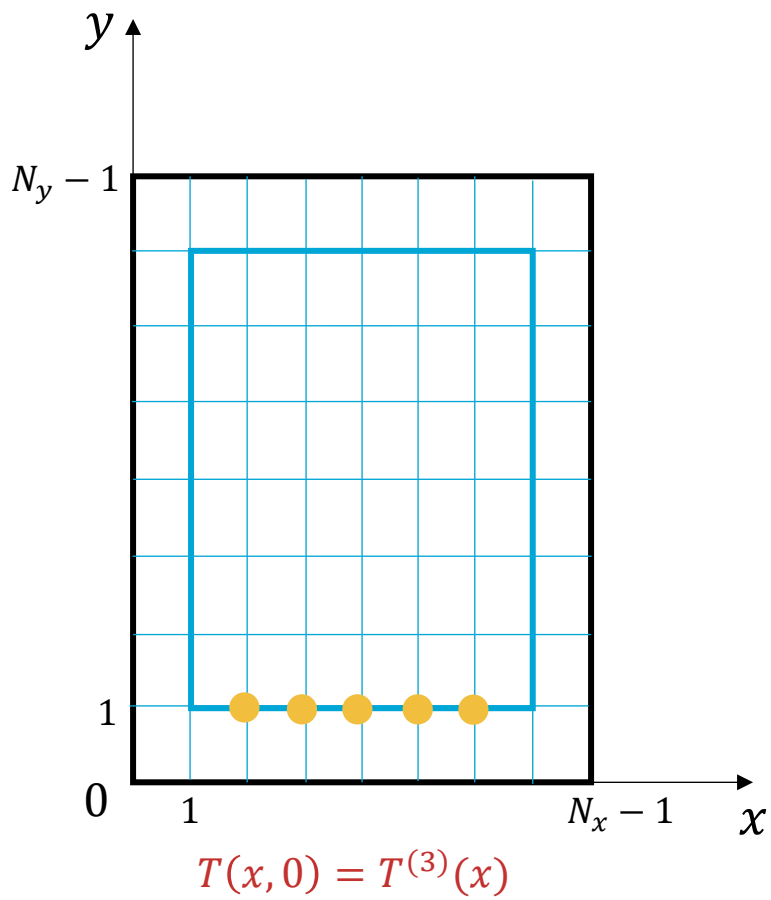
Considering the case study discussed earlier…

- **Dirichlet BC3:** $T(x,0) = T^{(3)}(x)$, where $T^{(3)}(x)$ is a function $f: \mathbb{R} \to \mathbb{R}$

We write a specialized equation for the nodes $i = 2, \ldots, N_x - 3$, $j = 1$: (note, this is $N_x - 4$ equations)

$$T_{i-1,1} + T_{i+1,1} + T_{i,0} + T_{i,2} - 4T_{i,1} = 0$$

→ Eliminate boundary nodes



Dirichlet boundary conditions – Bottom boundary

Considering the case study discussed earlier…

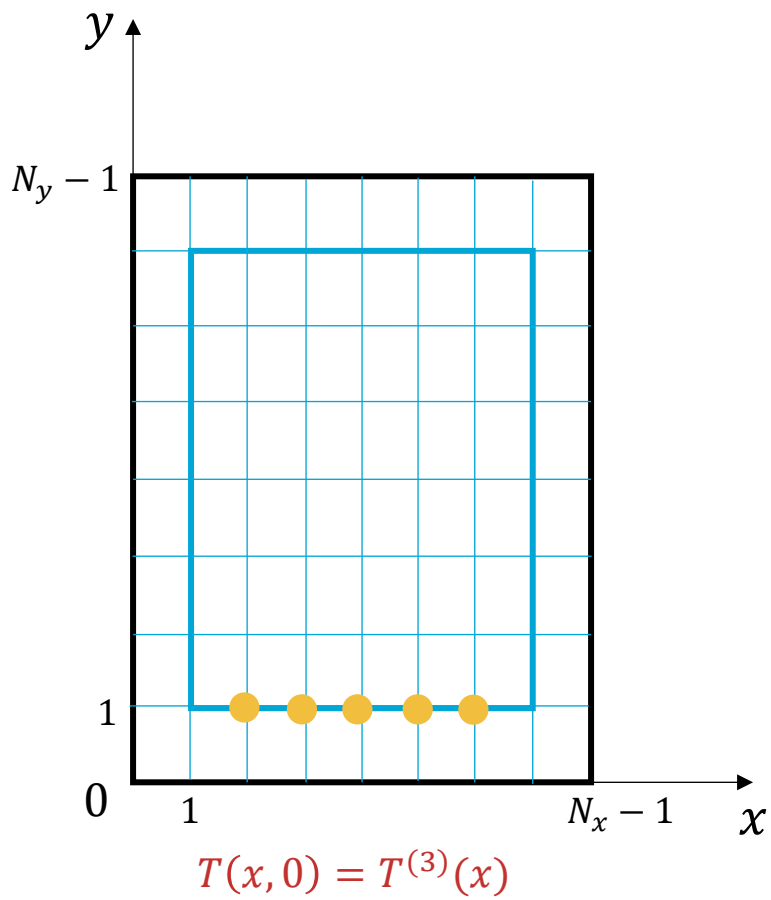
- **Dirichlet BC3:** $T(x,0) = T^{(3)}(x)$, where $T^{(3)}(x)$ is a function $f: \mathbb{R} \to \mathbb{R}$

We write a specialized equation for the nodes $i = 2, \ldots, N_x - 3$, $j = 1$: (note, this is $N_x - 4$ equations)

$$T_{i-1,1} + T_{i+1,1} + T_{i,0} + T_{i,2} - 4T_{i,1} = 0$$

But $T_{i,0} = T_i^{(3)}$, which is given by Dirichlet BC1, then:

$$T_{i-1,1} + T_{i+1,1} + T_{i,2} - 4T_{i,1} = -T_i^{(3)}$$



Dirichlet boundary conditions – Upper boundary

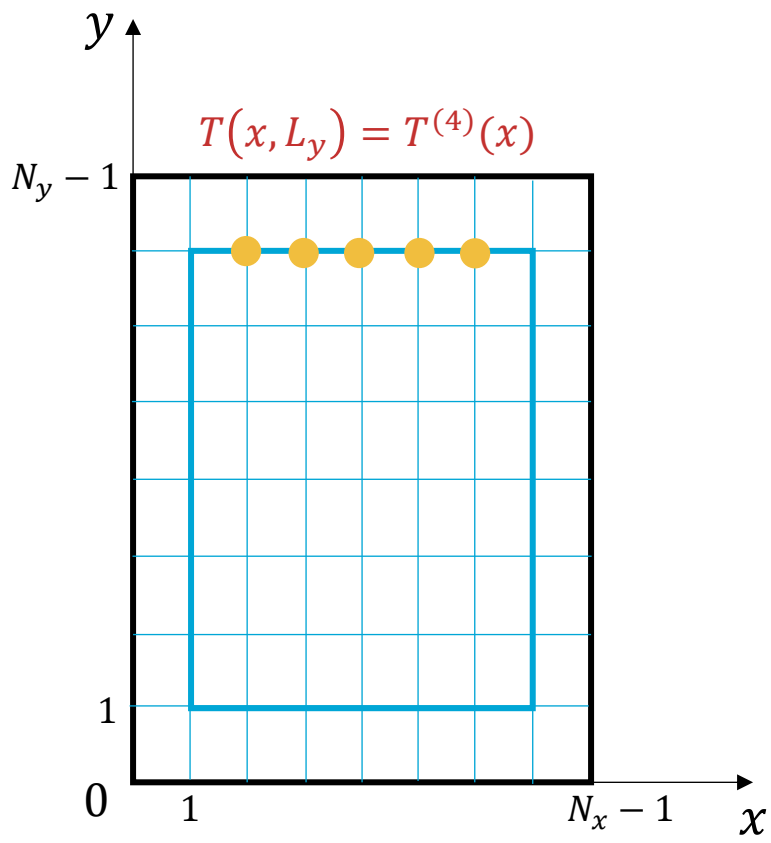
Considering the case study discussed earlier…

- **Dirichlet BC4:** $T(x, L_y) = T^{(4)}(x)$, where $T^{(4)}(x)$ is a function $f: \mathbb{R} \to \mathbb{R}$

We write a specialized equation for the nodes $i = 2, \ldots, N_x - 3$, $j = N_y - 2$: (note, this is $N_x - 4$ equations)

$$T_{i-1,N_y-2} + T_{i+1,N_y-2} + T_{i,N_y-3} + T_{i,N_y-1} - 4T_{i,N_y-2} = 0$$

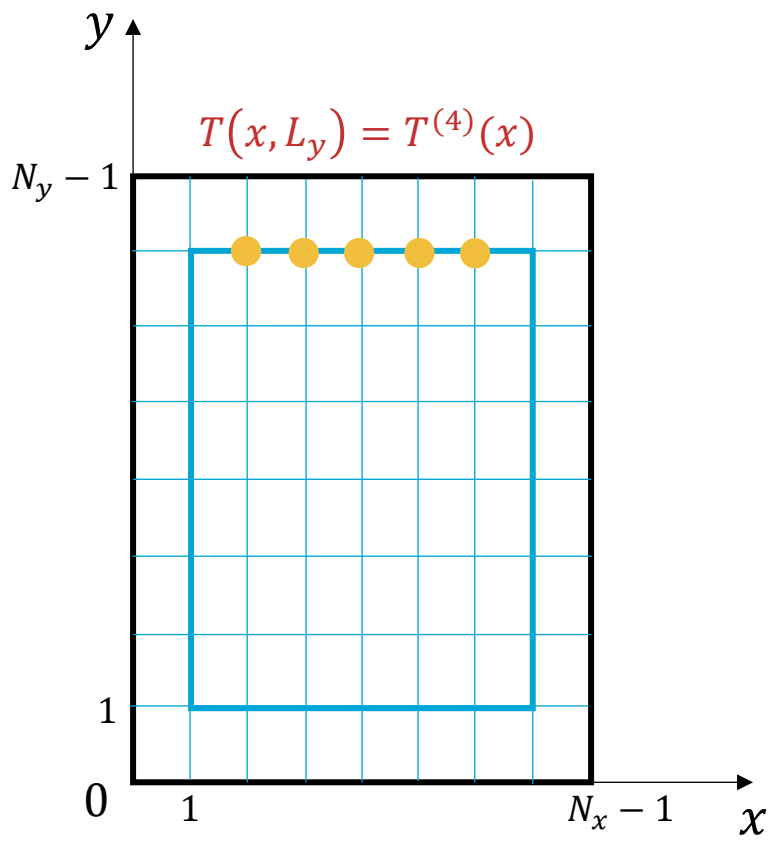
→ Eliminate boundary nodes



Dirichlet boundary conditions – Upper boundary

Considering the case study discussed earlier…

- **Dirichlet BC4:** $T(x, L_y) = T^{(4)}(x)$, where $T^{(4)}(x)$ is a function $f: \mathbb{R} \to \mathbb{R}$



We write a specialized equation for the nodes $i = 2, \ldots, N_x - 3$, $j = N_y - 2$: (note, this is $N_x - 4$ equations)

$$T_{i-1,N_y-2} + T_{i+1,N_y-2} + T_{i,N_y-3} + T_{i,N_y-1} - 4T_{i,N_y-2} = 0$$

But $T_{i,N_y-1} = T_i^{(4)}$, which is given by Dirichlet BC4, then:

$$T_{i-1,N_y-2} + T_{i+1,N_y-2} + T_{i,N_y-3} - 4T_{i,N_y-2} = -T_i^{(4)}$$

Corners – Bottom left

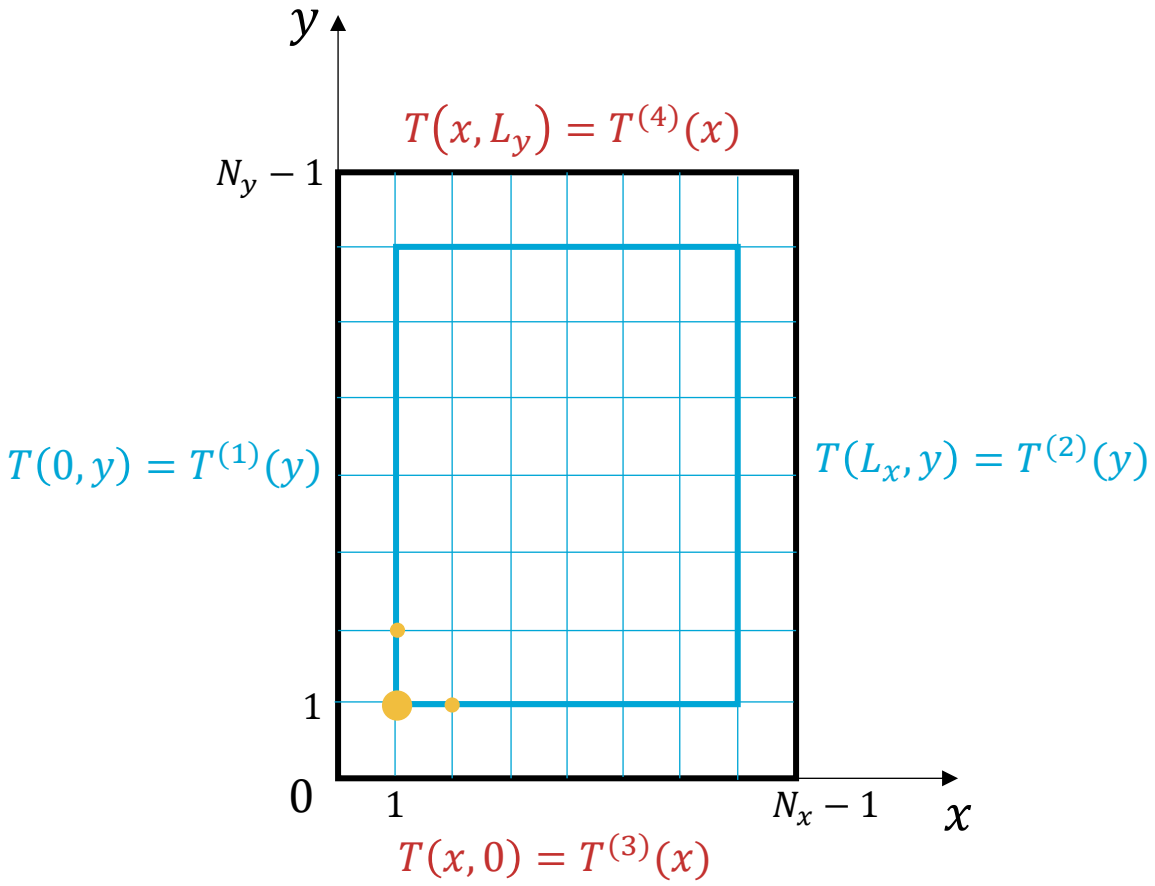
- Node $i = 1, j = 1$:

$$T_{0,1} + T_{2,1} + T_{1,0} + T_{1,2} - 4T_{1,1} = 0$$

  - But: $T_{0,1} = T_{j=1}^{(1)}$ and $T_{1,0} = T_{i=1}^{(3)}$

  - Then:

$$T_{2,1} + T_{1,2} - 4T_{1,1} = -T_{j=1}^{(1)} - T_{i=1}^{(3)}$$



Corners – Top left

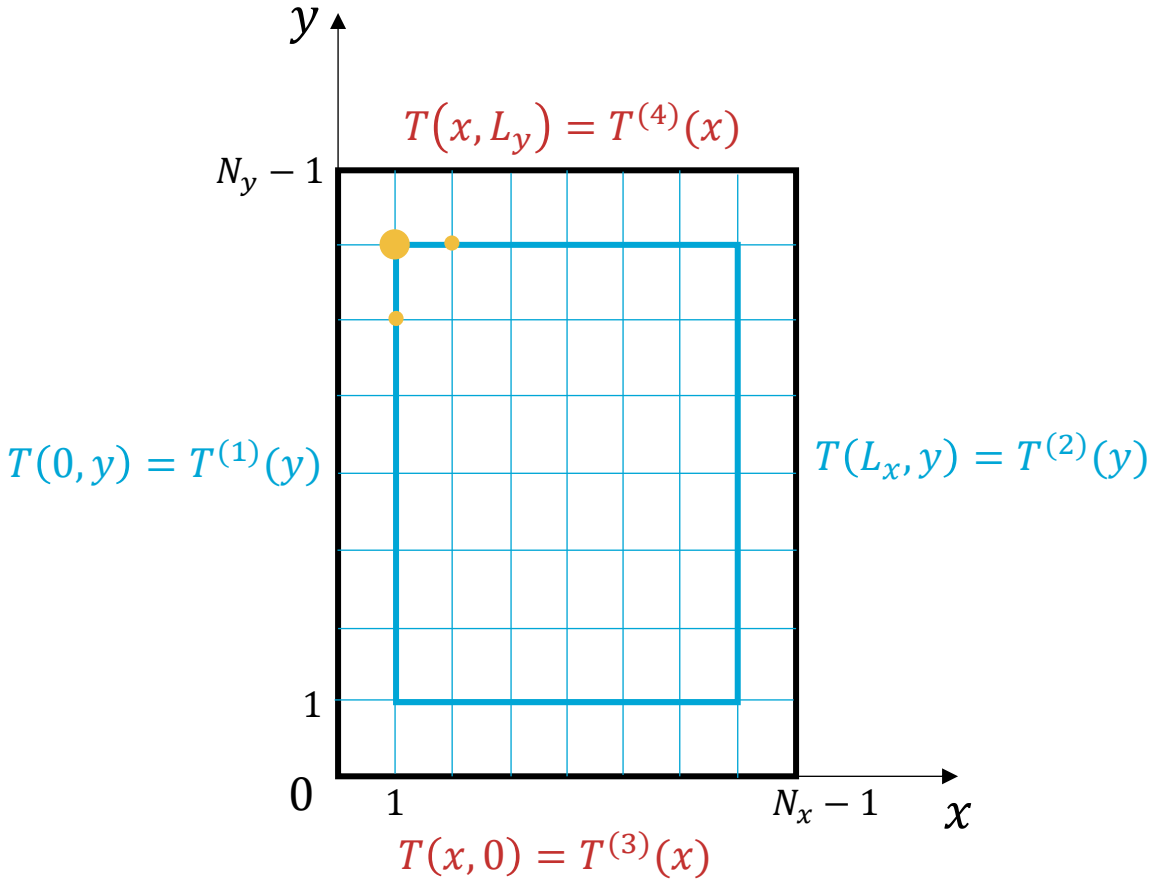
- Node $i = 1, j = N_y - 2$:

$$T_{0,N_y-2} + T_{2,N_y-2} + T_{1,N_y-3} + T_{1,N_y-1} - 4T_{1,N_y-2} = 0$$

  - But: $T_{0,N_y-2} = T^{(1)}_{j=N_y-2}$ and $T_{1,N_y-1} = T^{(4)}_{i=1}$

  - Then:

$$T_{2,N_y-2} + T_{1,N_y-3} - 4T_{1,N_y-2} = -T^{(1)}_{j=N_y-2} - T^{(4)}_{i=1}$$



Corners – Bottom right

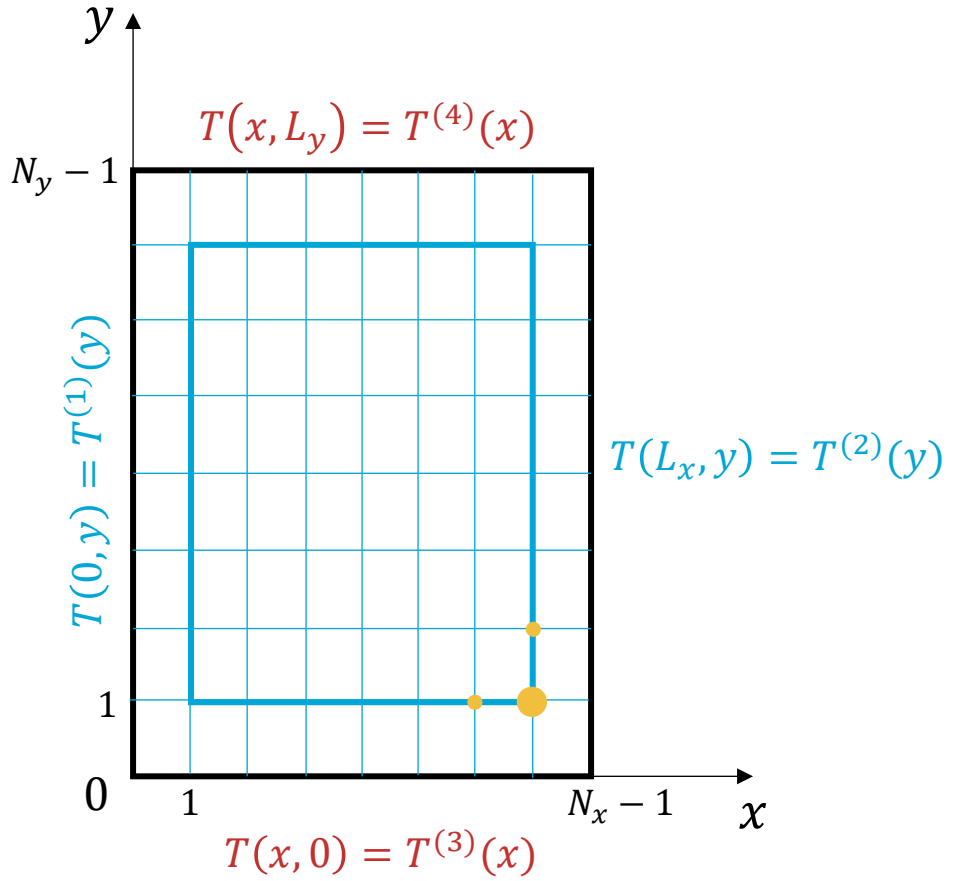
- Node $i = N_x - 2, j = 1$:

$$T_{N_x-3,1} + T_{N_x-1,1} + T_{N_x-2,0} + T_{N_x-2,2} - 4T_{N_x-2,1} = 0$$

  - But: $T_{N_x-1,1} = T^{(2)}_{j=1}$ and $T_{N_x-2,0} = T^{(3)}_{i=N_x-2}$

  - Then:

$$T_{N_x-3,1} + T_{N_x-2,2} - 4T_{N_x-2,1} = -T^{(2)}_{j=1} - T^{(3)}_{i=N_x-2}$$



Corners – Top right

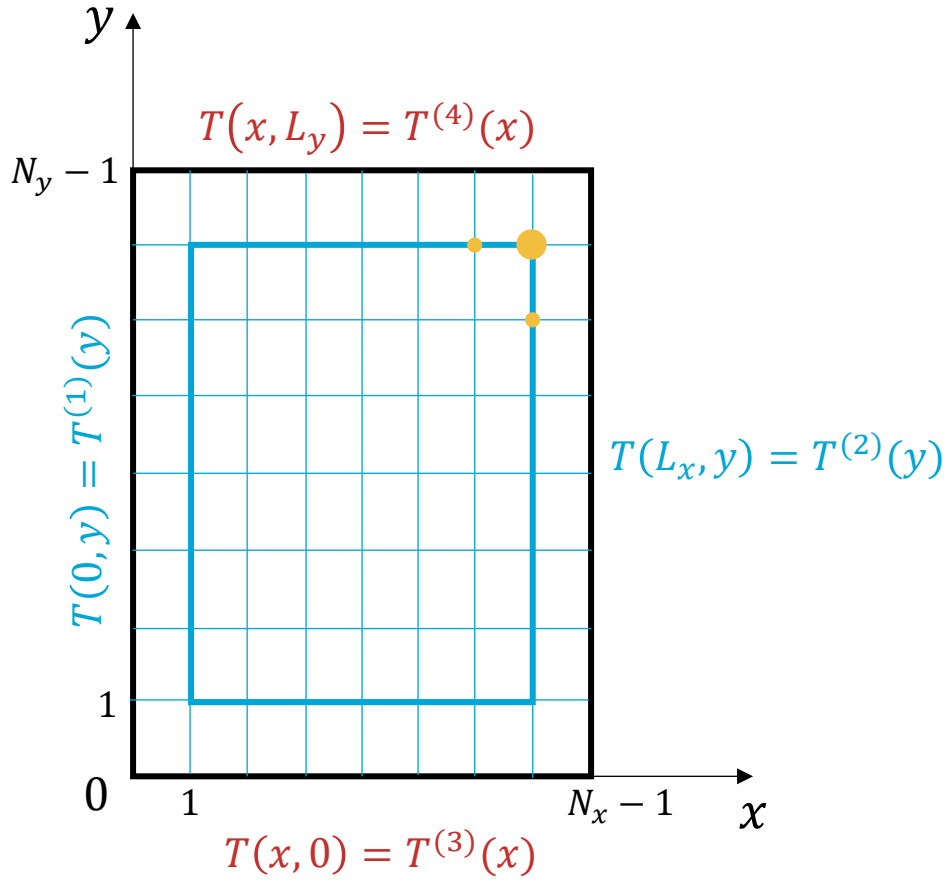
- Node $i = N_x - 2, j = N_y - 2$:

$$T_{N_x-3,N_y-2} + T_{N_x-1,N_y-2} + T_{N_x-2,N_y-3} + T_{N_x-2,N_y-1} - 4T_{N_x-2,N_y-2} = 0$$

  - But: $T_{N_x-1,N_y-2} = T^{(2)}_{j=N_y-2}$ and $T_{N_x-2,N_y-1} = T^{(4)}_{i=N_x-2}$

  - Then:

$$T_{N_x-3,N_y-2} + T_{N_x-2,N_y-3} - 4T_{N_x-2,N_y-2} = -T^{(1)}_{j=N_y-2} - T^{(4)}_{i=N_x-2}$$



Arranging the system of equations

- The equations defined in the nodes can in principle be arranged arbitrarily to compose a linear system of $(N_x - 2) \times (N_y - 2)$ equations.
- However, meaningful arrangement can improve computation performance by exploiting sparsity patterns.
- We consider the following arrangement of the temperature vector:

$$T = \Big[\underbrace{T_{1,1}, T_{2,1}, \dots, T_{N_x-2,1}}_{T_{i,1}}, \underbrace{T_{1,2}, T_{2,2}, \dots, T_{N_x-2,2}}_{T_{i,2}}, \dots\dots, \underbrace{T_{1,N_y-2}, T_{2,N_y-2}, \dots, T_{N_x-2,N_y-2}}_{T_{i,N_y-2}}\Big]^T$$

- $T_{i,1}, T_{i,2}, \dots, T_{i,N_y-2} \in \mathbb{R}^{N_y-2}$
- $T \in \mathbb{R}^{N_P}, \quad \text{with } N_P = (N_x - 2) \cdot (N_y - 2)$

Agenda

- Elliptic PDE: Laplace equation (Steady-state heat equation)
  - Boundary conditions at the corners
  - Finite difference method in 2D domain
  - Poisson matrix
- Implementation of sparse matrices using Scipy

Poisson matrix

- The linear system in matrix form arranged as described can be written as:

$$PT = b$$

$$\begin{bmatrix}
-4 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \ddots & \ddots & \ddots & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & \ddots & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & \ddots & 0 & 0 \\
0 & 0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & \ddots & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 1 & -4
\end{bmatrix}
\begin{bmatrix}
T_{1,1} \\ T_{2,1} \\ \vdots \\ T_{N_x-3,1} \\ T_{N_x-2,1} \\
T_{1,2} \\ T_{2,2} \\ \vdots \\ T_{N_x-3,2} \\ T_{N_x-2,2} \\
T_{1,N_y-2} \\ T_{2,N_y-2} \\ \vdots \\ T_{N_x-3,N_y-2} \\ T_{N_x-2,N_y-2}
\end{bmatrix}
= -
\begin{bmatrix}
T_{j=1}^{(1)} + T_{i=1}^{(3)} \\
T_{i=2}^{(3)} \\ \vdots \\ T_{i=N_x-3}^{(3)} \\
T_{j=1}^{(2)} + T_{i=N_x-2}^{(3)} \\
T_{j=2}^{(1)} \\ 0 \\ \vdots \\ 0 \\ T_{j=2}^{(2)} \\
T_{j=N_y-2}^{(1)} + T_{i=1}^{(4)} \\
T_{i=2}^{(4)} \\ \vdots \\ T_{i=N_y-3}^{(4)} \\
T_{j=N_y-2}^{(1)} + T_{i=N_x-2}^{(4)}
\end{bmatrix}$$

Poisson matrix – Corners

- The linear system in matrix form arranged as described can be written as:

$$PT = b$$

$$\begin{bmatrix}
-4 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \ddots & \ddots & \ddots & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & \ddots & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & \ddots & 0 & 0 \\
0 & 0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & \ddots & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 1 & -4
\end{bmatrix}
\begin{bmatrix}
T_{1,1} \\ T_{2,1} \\ \vdots \\ T_{N_x-3,1} \\ T_{N_x-2,1} \\
T_{1,2} \\ T_{2,2} \\ \vdots \\ T_{N_x-3,2} \\ T_{N_x-2,2} \\
T_{1,N_y-2} \\ T_{2,N_y-2} \\ \vdots \\ T_{N_x-3,N_y-2} \\ T_{N_x-2,N_y-2}
\end{bmatrix}
= -
\begin{bmatrix}
T^{(1)}_{j=1} + T^{(3)}_{i=1} \\
T^{(3)}_{i=2} \\ \vdots \\ T^{(3)}_{i=N_x-3} \\
T^{(2)}_{j=1} + T^{(3)}_{i=N_x-2} \\
T^{(1)}_{j=2} \\ 0 \\ \vdots \\ 0 \\ T^{(2)}_{j=2} \\
T^{(1)}_{j=N_y-2} + T^{(4)}_{i=1} \\
T^{(4)}_{i=2} \\ \vdots \\ T^{(4)}_{i=N_y-3} \\
T^{(1)}_{j=N_y-2} + T^{(4)}_{i=N_x-2}
\end{bmatrix}$$

Bottom left

Bottom right

Top left

Top right

Poisson matrix – Edges

- The linear system in matrix form arranged as described can be written as:

$$PT = b$$

$$\begin{bmatrix}
-4 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \ddots & \ddots & \ddots & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & \ddots & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & \ddots & 0 & 0 \\
0 & 0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & \ddots & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 1 & -4
\end{bmatrix}
\begin{bmatrix}
T_{1,1} \\ T_{2,1} \\ \vdots \\ T_{N_x-3,1} \\ T_{N_x-2,1} \\
T_{1,2} \\ T_{2,2} \\ \vdots \\ T_{N_x-3,2} \\ T_{N_x-2,2} \\
T_{1,N_y-2} \\ T_{2,N_y-2} \\ \vdots \\ T_{N_x-3,N_y-2} \\ T_{N_x-2,N_y-2}
\end{bmatrix}
= -
\begin{bmatrix}
T_{j=1}^{(1)} + T_{i=1}^{(3)} \\ T_{i=2}^{(3)} \\ \vdots \\ T_{i=N_x-3}^{(3)} \\ T_{j=1}^{(2)} + T_{i=N_x-2}^{(3)} \\
T_{j=2}^{(1)} \\ 0 \\ \vdots \\ 0 \\ T_{j=2}^{(2)} \\
T_{j=N_y-2}^{(1)} + T_{i=1}^{(4)} \\ T_{i=2}^{(4)} \\ \vdots \\ T_{i=N_y-3}^{(4)} \\ T_{j=N_y-2}^{(1)} + T_{i=N_x-2}^{(4)}
\end{bmatrix}$$

Bottom edge

Top edge

Poisson matrix – Edges

- The linear system in matrix form arranged as described can be written as:

$$PT = b$$

$$\begin{bmatrix}
-4 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \ddots & \ddots & \ddots & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & \ddots & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & \ddots & 0 & 0 \\
0 & 0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & \ddots & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 1 & -4
\end{bmatrix}
\begin{bmatrix}
T_{1,1} \\ T_{2,1} \\ \vdots \\ T_{N_x-3,1} \\ T_{N_x-2,1} \\ T_{1,2} \\ T_{2,2} \\ \vdots \\ T_{N_x-3,2} \\ T_{N_x-2,2} \\ T_{1,N_y-2} \\ T_{2,N_y-2} \\ \vdots \\ T_{N_x-3,N_y-2} \\ T_{N_x-2,N_y-2}
\end{bmatrix}
= -
\begin{bmatrix}
T_{j=1}^{(1)} + T_{i=1}^{(3)} \\ T_{i=2}^{(3)} \\ \vdots \\ T_{i=N_x-3}^{(3)} \\ T_{j=1}^{(2)} + T_{i=N_x-2}^{(3)} \\ T_{j=2}^{(1)} \\ 0 \\ \vdots \\ 0 \\ T_{j=2}^{(2)} \\ T_{j=N_y-2}^{(1)} + T_{i=1}^{(4)} \\ T_{i=2}^{(4)} \\ \vdots \\ T_{i=N_y-3}^{(4)} \\ T_{j=N_y-2}^{(1)} + T_{i=N_x-2}^{(4)}
\end{bmatrix}$$

Left edge
(first element of each block)

Right edge
(last element of each block)

Poisson matrix

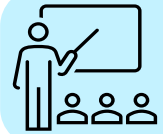
- The Poisson matrix exhibit a defined sparsity pattern:
  - Tridiagonal matrix blocks
  - Diagonal matrix blocks

HINT

Exploit sparsity to solve the linear system efficiently!

$$
\begin{bmatrix}
-4 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \ddots & \ddots & \ddots & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & \ddots & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & \ddots & 0 & 0 \\
0 & 0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 1 & -4 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -4 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 1 & -4
\end{bmatrix}
\begin{bmatrix}
T_{1,1} \\ T_{2,1} \\ \vdots \\ T_{N_x-3,1} \\ T_{N_x-2,1} \\ T_{1,2} \\ T_{2,2} \\ \vdots \\ T_{N_x-3,2} \\ T_{N_x-2,2} \\ T_{1,N_y-2} \\ T_{2,N_y-2} \\ \vdots \\ T_{N_x-3,N_y-2} \\ T_{N_x-2,N_y-2}
\end{bmatrix}
= -
\begin{bmatrix}
T_{j=1}^{(1)} + T_{i=1}^{(3)} \\ T_{i=2}^{(3)} \\ \vdots \\ T_{i=N_x-3}^{(3)} \\ T_{j=1}^{(2)} + T_{i=N_x-2}^{(3)} \\ T_{j=2}^{(1)} \\ 0 \\ \vdots \\ 0 \\ T_{j=2}^{(2)} \\ T_{j=N_y-2}^{(1)} + T_{i=1}^{(4)} \\ T_{i=2}^{(4)} \\ \vdots \\ T_{i=N_y-3}^{(4)} \\ T_{j=N_y-2}^{(1)} + T_{i=N_x-2}^{(4)}
\end{bmatrix}
$$

Other boundary conditions and techniques



See CP Q2
Lecture 2

- Similarly, we can derive a Poisson matrix when the problem is defined with Neumann boundary conditions:
  - The derivation of boundary equations is equivalent to the one for 1D BVPs
  - The 2D spatial domain is handled equivalently to the case of Dirichlet BCs discussed earlier
- For the case of Neumann boundary conditions, one can consider:

1. Using forward and backward difference schemes and derive a system with

$$\mathrm{N_{eq}} = (N_y - 2) \times (N_x - 2)$$

2. Using **ghost points** and central difference scheme and derive a system with

$$\mathrm{N_{eq}} = N_x \times N_y$$

Suggested choice from more accurate discretization

This is equivalent to the case of parabolic PDEs and the same considerations hold.

Elliptic PDE: Live coding

- Open Colab: [Elliptic PDE – 2D steady state heat equation (a.k.a. Laplace equation)]

- Find more in the Github repository of the course: [https://github.com/process-intelligence-research/computational_practicum_lecture_coding/tree/main]

Agenda

- Elliptic PDE: Laplace equation (Steady-state heat equation)
  - Boundary conditions at the corners
  - Finite difference method in 2D domain
  - Poisson matrix
- Implementation of sparse matrices using Scipy

Sparse matrix definition

- Poisson matrix is a large but sparse matrix (most coefficients are zero)
- Scipy package implements efficient handlers and solvers for sparse systems
- If the matrix to be defined is structured with shared coefficient values (for instance, tridiagonal matrix with same values on the diagonals)
  - Use scipy.sparse.spdiags (remember assignment Q2L2)
- If the matrix definition is more **complex**, it might be necessary to define the coefficients with two nested loops, iterating through rows and columns
- For instance, the Poisson matrix does not show same coefficients throughout entire diagonals
  - Use scipy.sparse.lilmatrix (used in assignment Q2L3)

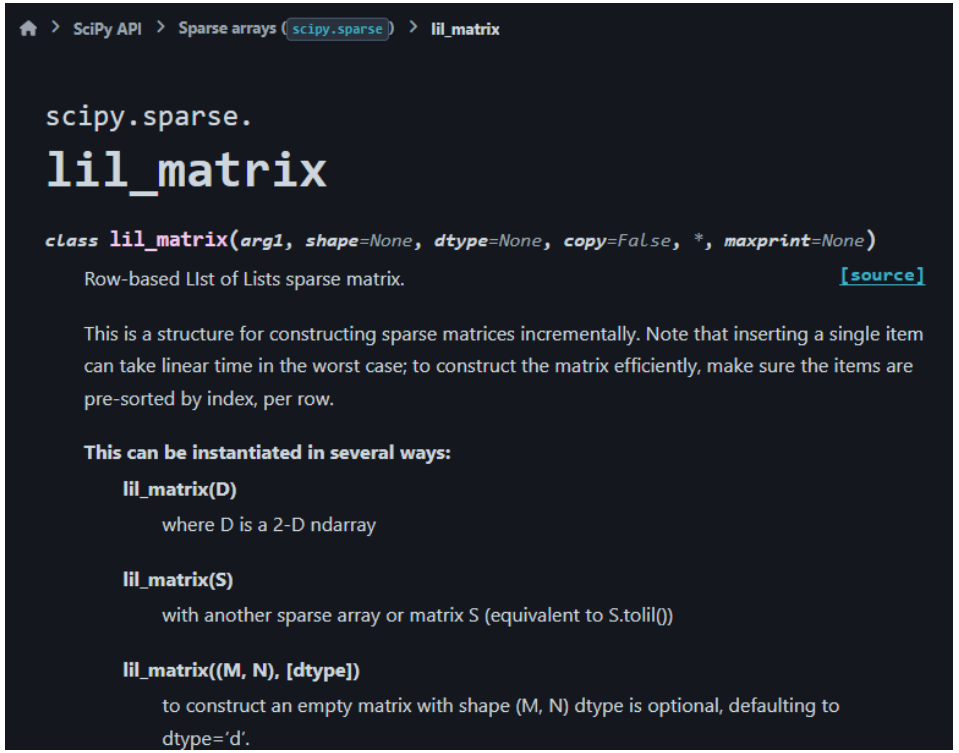
Example pseudo-code

- Pseudo-implementation of the Poisson matrix construction
- Using lil_matrix to assign coefficients efficiently through nested loops
- The coefficients themselves will depend on the specific problem and boundary conditions

```
# PSEUDO-CODE

N = N_z * N_r
A = lil_matrix((N, N))

for j in range(N_r):  # radial index
    for i in range(N_z):  # axial index
        n = i + j * N_z

        A[n, n] = a_P
        A[n, east_index] = a_E
        A[n, west_index] = a_W
        A[n, north_index] = a_N
        A[n, south_index] = a_S
```

lil_matrix

- lil_matrix is a flexible builder before converting into a fast solver format
- Meaning: “List of Lists” sparse matrix
- It is best suited for incremental construction, such as assigning or changing elements inside loops
- Important note: A lil_matrix is not ready for fast solving.
- After construction and before running the solver, a lil_matrix must be converted to efficient format, compressed such as compressed sparse row
  Example:
  A_csr = A.tocsr()
  solution = scipy.sparse.spsolve(A_csr, b)



Solve PDEs with neural networks

- Recently, research have looked into developing artificial neural network able to *learn* the solution of (Partial) Differential equations!
- The original formulation of such method is the popular framework called *Physics-Informed Neural Networks (PINNs)* [1].
- PINNs have be proved to mix the learning capabilities from data and the knowledge of the underlining physics of the problem to solve many class of PDEs.

If you are interested to know more about AI and machine learning, join the course:
AI in (Bio-)Chemical engineering (CH3113) in Q3!

[1] Journal of Computational Physics, M. Raissi et al. (2019)

Learning objectives

After successfully completing this lecture, you are able to…

- Explain what are elliptic PDEs and their applications in (chemical) engineering
- Implement numerical solution approaches for elliptic PDEs in Python
- Implement efficient sparse matrices in Python using scipy package

Thank you very much for your attention!