Computational Physics

One-Dimensional Stationary Heat Equation

- Boundary value problem
- Heat Equation
- Finite Differences
- Inhomogeneous case
- Gaussian Elimination (LU decomposition)

Boundary Value Problems

A linear boundary value problem (of order n) can be defined as

$$\begin{cases} L[y] = f(x), & x \in [a,b] \ , \\ U_\nu[y] = \lambda_\nu, & \nu = 1, \ldots, n \ . \end{cases}$$

with the linear operator L[y]

$$L[y] = \sum_{k=0}^{n} a_k(x) y^{(k)}(x)$$

f(x) and $a_k(x)$ are continuous, given functions; $y^{(k)}(x)$ are the kth spatial derivates
Boundary conditions (BC) are therefore defined as (constants $\alpha_{\nu k}$, $\beta_{\nu k}$, λ_ν are given)

$$U_\nu[y] = \sum_{k=0}^{n-1} \left[\alpha_{\nu k} y^{(k)}(a) + \beta_{\nu k} y^{(k)}(b) \right] = \lambda_\nu$$

Example for n=2:

$$y(a) = \alpha \ , \qquad y(b) = \beta \qquad \text{Dirichlet BC}$$

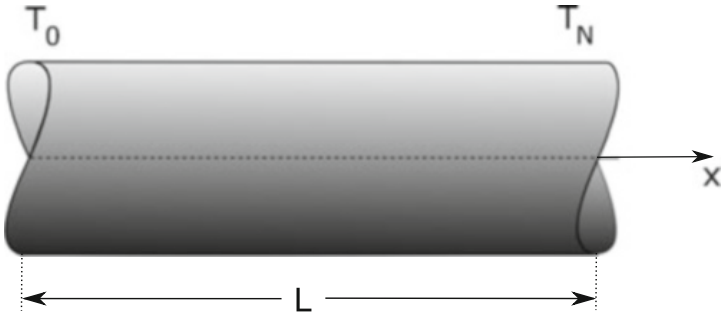
$$y'(a) = \alpha \ , \qquad y'(b) = \beta \qquad \text{von-Neumann BC}$$

Discretization (using central differences) results then in a linear equation system for the y_k, which one needs to solve, e.g. Gaussian elimination,

Read chapter 8 & Appendix C

The homogeneous heat equation

Here we consider a rod of length L, which is kept at temperatures T_0 and T_N at its ends



The temperature profile in this rod as a function of time is T(x,t). The heat equation is then defined as

$$\frac{\partial}{\partial t}T(x,t) = \kappa \Delta T(x,t)$$

which is a linear *partial* differential equation. The operator Δ is the Laplace operator, given by $\Delta=\nabla^2=\partial_x^2+\partial_y^2+\partial_z^2$ in 3D. κ is the thermal diffusivity.

...

The heat equation is typically solved together with initial and boundary value conditions. It is equivalent to solving the diffusion equation:

$$\frac{\partial}{\partial t}\rho(x,t) = D\Delta\rho(x,t)$$

where ρ(x,t) is the (particle) density and *D* the diffusion constant.
Here we concentrate on the 1D case and assume that a *steady-state* has been reached when $\partial_t T(x,t)=0$. Therefore, we get the boundary value problem:

$$\begin{cases} \dfrac{\mathrm{d}^2}{\mathrm{d}x^2}T(x) = 0, & x \in [0, L] \\ T(0) = T_0\ , \\ T(L) = T_N\ . \end{cases}$$

...

which has the (obvious) analytical solution:

$$T(x) = T_0 + (T_N - T_0)\,\frac{x}{L}$$

Finite Differences

We discretize the interval [0; L] in N equal length, h, subintervals using N+1 grid points: $x_n=nh$, $h=L/N$, $x_0=0$, $x_N=L$, $T_n=T(x_n)$

$$\frac{T_{n+1}-2T_n+T_{n-1}}{h^2}=0$$

$$\Leftrightarrow \qquad T_{n+1}-2T_n+T_{n-1}=0$$

The boundary conditions are defined by T_0 and T_N

This linear equation system for the $T_1,\ldots, T_{N-1}$ are then written in the form

$$\underline{\underline{A}}\cdot\underline{T}=\underline{F}$$

with $\underline{T}=(T_1,\ldots, T_{N-1})^T$

with and

$$\underline{\underline{A}}=\begin{pmatrix} -2 & 1 & 0 & & \ldots & 0 \\ 1 & -2 & 1 & 0 & \ldots & 0 \\ 0 & 1 & -2 & 1 & & \\ \vdots & & \ddots & \ddots & \ddots & \\ 0 & \ldots & & & 1 & -2 \end{pmatrix} \qquad \underline{F}=\begin{pmatrix} -T_0 \\ 0 \\ \vdots \\ 0 \\ -T_N \end{pmatrix}$$

...

This is easily solved: $T_{n+1} = 2T_n - T_{n-1}, \qquad n = 1, \ldots, N-1$

i.e.

$$\begin{aligned} T_2 &= 2T_1 - T_0 \ , \\ T_3 &= 2T_2 - T_1 \ , \\ &= 3T_1 - 2T_0 \\ T_4 &= 2T_3 - T_2 \ , \\ &= 4T_1 - 3T_0 \end{aligned}$$

in general: $T_n = nT_1 - (n-1)T_0$

which is quickly proven by induction:

$$\begin{aligned} T_{n+1} &= 2T_n - T_{n-1} \\ &= 2[nT_1 - (n-1)T_0] - [(n-1)T_1 - (n-2)T_0] \\ &= (n+1)T_1 - nT_0 \ . \end{aligned}$$

...

Since T_N is kept constant, we can infer T_1: $T_N = NT_1 - NT_0 + T_0$

$$T_1 = \frac{T_N - T_0}{N} + T_0$$

and finally:

$$T_n = T_0 + (T_N - T_0)\frac{n}{N}$$

$$= T_0 + (T_N - T_0)\frac{nh}{L}$$

which is just the discretized version of the analytic solution. I.e. the solution is exact, which is not surprising as finite derivatives are exact for linear functions.

Inhomogeneous heat equation

We consider now:

$$\frac{\partial}{\partial t}T(x,t) = \kappa \Delta T(x,t) - \Gamma(x,t)$$

Here $\Gamma(x,t)=\Gamma(x)$

in particular the 1D stationary case:

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}T(x) = \frac{1}{\kappa}\Gamma(x)$$

Next, we consider the special case of

$$\Gamma(x) = \frac{\Theta}{\ell}\exp\left[-\frac{\left(x-\frac{L}{2}\right)^2}{\ell^2}\right]$$

i.e. a Gaussian peak in the middle of the system of width ℓ and height Θ
Physically this means the rod is locally heated or cooled in the center.
for $\ell \to 0$ this function becomes:

$$\lim_{\ell\to 0}\Gamma(x) \propto \Theta\delta\left(x-\frac{L}{2}\right)$$

...

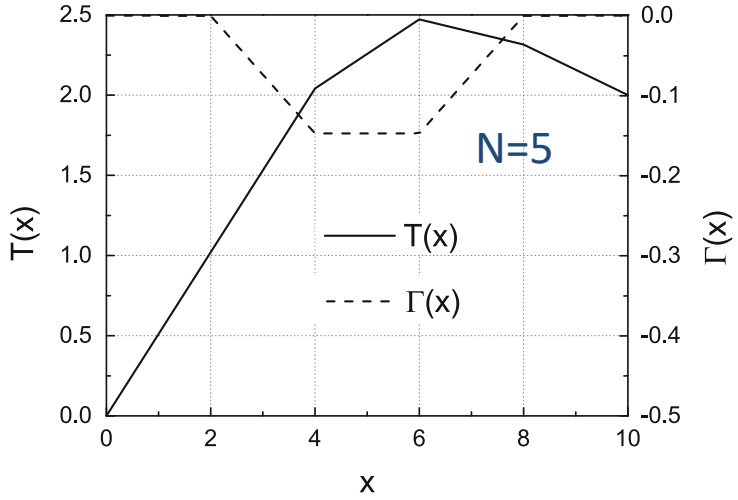
The rhs of the linear equation system is then changed to:

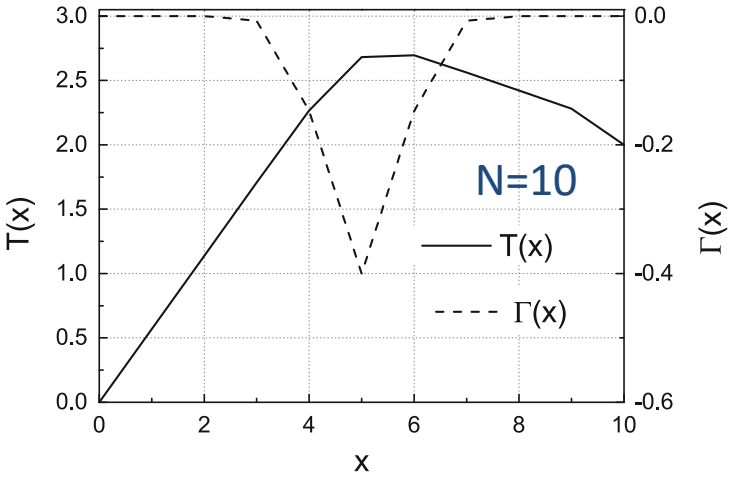
$$\underline{F} = \frac{h^2}{\kappa} \begin{pmatrix} \Gamma_1 - \frac{\kappa}{h^2} T_0 \\ \Gamma_2 \\ \vdots \\ \Gamma_{N-2} \\ \Gamma_{N-1} - \frac{\kappa}{h^2} T_N \end{pmatrix} \qquad \Gamma_n \equiv \Gamma(x_n)$$

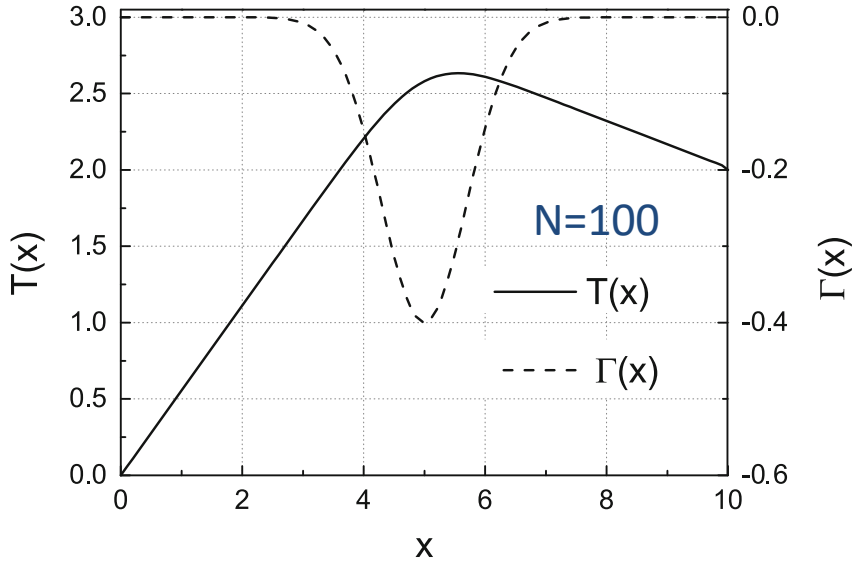
everything else is unchanged.

numerical solutions

For L=10, κ=1, Θ=-0.4, ℓ=1, T_0=0, T_N=2.0







Gaussian elimination (GE)

Gaussian Elimination:

- Reduce A**x**=**b** to an equivalent system (that is, having the same solution) of form U**x**=$\mathcal{b}$
  U: upper triangular matrix, $\mathcal{b}$: updated right side vector.
- The latter system can then be solved by backward substitution
- Let us denote the original system by $A^{(1)}\mathbf{x} = \mathbf{b}^{(1)}$

1. Introduce the multipliers:

$$m_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, \quad i = 2, 3, \ldots, n$$

2. Eliminate the unknown x_1 in the following rows i below row 1:

$$a_{ij}^{(2)} = a_{ij}^{(1)} - m_{i1} a_{1j}^{(1)}, \quad i, j = 2, \ldots, n,$$

$$b_i^{(2)} = b_i^{(1)} - m_{i1} b_1^{(1)}, \quad i = 2, \ldots, n,$$

➔

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \ldots & a_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(2)} & \ldots & a_{nn}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{bmatrix}$$

$\Leftrightarrow A^{(2)}\mathbf{x} = \mathbf{b}^{(2)}$

...

Then eliminate x_2 from rows 3,…,n, etc.
In general after k-1 elimination steps, we have a system: $\mathrm{A}^{(k)}\mathbf{x} = \mathbf{b}^{(k)}, \quad 1 \le k \le n,$

$$\mathrm{A}^{(k)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & \dots & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & & & & a_{2n}^{(2)} \\ \vdots & & \ddots & & & \vdots \\ 0 & \dots & 0 & a_{kk}^{(k)} & \dots & a_{kn}^{(k)} \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & a_{nk}^{(k)} & \dots & a_{nn}^{(k)} \end{bmatrix}$$

And finally, we get:

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & & & a_{2n}^{(2)} \\ \vdots & & \ddots & & \vdots \\ 0 & & & \ddots & \vdots \\ 0 & & & & a_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ \vdots \\ b_n^{(n)} \end{bmatrix} \Leftrightarrow \mathrm{U}\mathbf{x} = \mathcal{b}$$

We assumed $a^{(i)}_{ii} \neq 0$ (i=1,…,n-1). These elements are called **pivots**.

$O(n^3)$ algorithm

Triangular matrices: forward & backward substitution

Consider the non-singular, lower triangular 3x3 matrix:

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

→
$$\begin{aligned} x_1 &= b_1/l_{11}, \\ x_2 &= (b_2 - l_{21}x_1)/l_{22}, \\ x_3 &= (b_3 - l_{31}x_1 - l_{32}x_2)/l_{33} \end{aligned}$$

Can be extended to systems n × n: *forward substitution algorithm*

$$\begin{aligned} x_1 &= \frac{b_1}{l_{11}}, \\ x_i &= \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} x_j \right), \quad i = 2, \ldots, n \end{aligned}$$

O(n^2) algorithm

...

Equivalent for upper triangular matrix [U**x**=**b**]: *backward substitution*

$$x_n = \frac{b_n}{u_{nn}},$$

$$x_i = \frac{1}{u_{ii}} \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right), \quad i = n-1, \ldots, 1$$

Algorithms (MatLab code)

```
function [b]=forward_col(L,b)
[n]=mat_square(L);
for j=1:n-1,
    b(j)= b(j)/L(j,j); b(j+1:n)=b(j+1:n)-b(j)*L(j+1:n,j);
end; b(n) = b(n)/L(n,n);
```

```
function [b]=backward_col(U,b)
[n]=mat_square(U);
for j = n:-1:2,
    b(j)=b(j)/U(j,j); b(1:j-1)=b(1:j-1)-b(j)*U(1:j-1,j);
end; b(1) = b(1)/U(1,1);
```

GE Example

3x3 Hilbert matrix:

$$(\mathrm{A}^{(1)}\mathbf{x} = \mathbf{b}^{(1)}) \quad \begin{cases} x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3 = \frac{11}{6} \\ \frac{1}{2}x_1 + \frac{1}{3}x_2 + \frac{1}{4}x_3 = \frac{13}{12} \\ \frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 = \frac{47}{60} \end{cases}$$

$m_{21}=1/2$, $m_{31}=1/3$:

$$(\mathrm{A}^{(2)}\mathbf{x} = \mathbf{b}^{(2)}) \quad \begin{cases} x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3 = \frac{11}{6} \\ 0 + \frac{1}{12}x_2 + \frac{1}{12}x_3 = \frac{1}{6} \\ 0 + \frac{1}{12}x_2 + \frac{4}{45}x_3 = \frac{31}{180} \end{cases}$$

$m_{32}=1$:

$$(\mathrm{A}^{(3)}\mathbf{x} = \mathbf{b}^{(3)}) \quad \begin{cases} x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3 = \frac{11}{6} \\ 0 + \frac{1}{12}x_2 + \frac{1}{12}x_3 = \frac{1}{6} \\ 0 + 0 + \frac{1}{180}x_3 = \frac{1}{180} \end{cases}$$

→ $x_3=1$, $x_2=1$, $x_1=1$

General Hilbert matrix: $h_{ij}=1/(i+j-1)$; $i,j=1,\ldots,n$

pivots

GE only works if the pivots are finite.

There are classes of matrices, when GE is “safe”

- A is diagonally dominant by rows
- A is diagonally dominant by column
- A is symmetric and positive definite

If zero (or small) pivots are encountered, one can reorder the remaining rows of $A^{(k)}$ [$\mathbf{b}^{(k)}$ elements accordingly] in order to move the largest (absolute value) element to the pivot position and continue.

Pseudocode for GE with pivoting

```
for k = 1 ... m:
  //Find pivot for column k:
  i_max  := argmax (i = k ... m, abs(A[i, k]))
  if A[i_max, k] = 0
    error "Matrix is singular!"
  swap rows(k, i_max)
  //Do for all rows below pivot:
   for i = k + 1 ... m:
     //Do for all remaining elements in current row:
     for j = k ... n:
       A[i, j]  := A[i, j] - A[k, j] * (A[i, k] / A[k, k])
    //Fill lower triangular matrix with zeros:
    A[i, k]  := 0
```

LU decomposition

GE is equivalent to performing a factorization of the matrix A into the product of two matrices, A=LU, with $U=A^{(n)}$.

- L and U do not depend on **b** and can therefore be used to solve the linear system for different **b.**
  This means a reduction of computation time to $O(n^2)$
- Let us go back to the Hilbert matrix example to see how the matrix L is constructed:

define:

$$\mathrm{M}_1 = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{3} & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -m_{21} & 1 & 0 \\ -m_{31} & 0 & 1 \end{bmatrix}$$

indeed:

$$\mathrm{M}_1\mathrm{A} = \mathrm{M}_1\mathrm{A}^{(1)} = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & \frac{1}{12} & \frac{1}{12} \\ 0 & \frac{1}{12} & \frac{4}{45} \end{bmatrix} = \mathrm{A}^{(2)}$$

...

and

$$\mathrm{M}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -m_{32} & 1 \end{bmatrix}$$

therefore

$$\mathrm{M}_2\mathrm{M}_1\mathrm{A} = \mathrm{A}^{(3)} = \mathrm{U}$$

$$\mathrm{A} = (\mathrm{M}_2\mathrm{M}_1)^{-1}\mathrm{U} = \mathrm{LU}$$

In general

$$\mathbf{m}_k = (0, \ldots, 0, m_{k+1,k}, \ldots, m_{n,k})^T \in \mathbb{R}^n$$

$$\mathrm{M}_k = \begin{bmatrix} 1 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & & 1 & 0 & & 0 \\ 0 & & -m_{k+1,k} & 1 & & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & -m_{n,k} & 0 & \ldots & 1 \end{bmatrix} = \mathrm{I}_n - \mathbf{m}_k \mathbf{e}_k^T$$

$$\mathrm{A}^{(k+1)} = \mathrm{M}_k \mathrm{A}^{(k)}$$

...

The complete elimination process is therefore: $\mathrm{M}_{n-1}\mathrm{M}_{n-2}\ldots\mathrm{M}_1\mathrm{A}=\mathrm{U}$

with $\mathrm{M}_k^{-1}=2\mathrm{I}_n-\mathrm{M}_k=\mathrm{I}_n+\mathbf{m}_k\mathbf{e}_k^T$

we get L from:

$$\begin{aligned}
\mathrm{A} &= \mathrm{M}_1^{-1}\mathrm{M}_2^{-1}\ldots\mathrm{M}_{n-1}^{-1}\mathrm{U}\\
&= (\mathrm{I}_n+\mathbf{m}_1\mathbf{e}_1^T)(\mathrm{I}_n+\mathbf{m}_2\mathbf{e}_2^T)\ldots(\mathrm{I}_n+\mathbf{m}_{n-1}\mathbf{e}_{n-1}^T)\mathrm{U}\\
&= \left(\mathrm{I}_n+\sum_{i=1}^{n-1}\mathbf{m}_i\mathbf{e}_i^T\right)\mathrm{U}\\
&= \begin{bmatrix} 1 & 0 & \ldots & \ldots & 0\\ m_{21} & 1 & & & \vdots\\ \vdots & m_{32} & \ddots & & \vdots\\ \vdots & \vdots & & \ddots & 0\\ m_{n1} & m_{n2} & \ldots & m_{n,n-1} & 1 \end{bmatrix}\mathrm{U}.
\end{aligned}$$

Once the matrices L and U have been computed, solving the linear system consists only of solving successively the two triangular systems:

$$\mathrm{L}\mathbf{y}=\mathbf{b}$$

$$\mathrm{U}\mathbf{x}=\mathbf{y}$$

LU implementation

Since L is a lower triangular matrix with diagonal entries equal to 1 and U is upper triangular, it is possible (and convenient) to store the LU factorization directly in the same memory area that is occupied by the matrix A. More precisely, U is stored in the upper triangular part of A (including the diagonal), whilst L occupies the lower triangular portion of A (the diagonal entries of L are not stored since they are implicitly assumed to be 1).

MatLab implementation

```
function [A] = lu_kji (A)
[n,n]=size(A);
for k=1:n-1
   A(k+1:n,k)=A(k+1:n,k)/A(k,k);
   for j=k+1:n, for i=k+1:n
                    A(i,j)=A(i,j)-A(i,k)*A(k,j);
   end,         end
end
```

Special cases

In simulations the matrix A is often sparse, i.e., most elements zero.
In particular the have a band structure with finite diagonal elements and a few finite off-diagonals.

Tridiagonal matrices:
(occur e.g. when discretizing gradients and Laplacians)

$$\mathrm{A} = \begin{bmatrix} a_1 & c_1 & & 0 \\ b_2 & a_2 & \ddots & \\ & \ddots & & c_{n-1} \\ 0 & & b_n & a_n \end{bmatrix}$$

Then

$$\mathrm{L} = \begin{bmatrix} 1 & & & 0 \\ \beta_2 & 1 & & \\ & \ddots & \ddots & \\ 0 & & \beta_n & 1 \end{bmatrix} \quad \mathrm{U} = \begin{bmatrix} \alpha_1 & c_1 & & 0 \\ & \alpha_2 & \ddots & \\ & & \ddots & c_{n-1} \\ 0 & & & \alpha_n \end{bmatrix}$$

with

$$\alpha_1 = a_1, \quad \beta_i = \frac{b_i}{\alpha_{i-1}}, \quad \alpha_i = a_i - \beta_i c_{i-1}, \ i = 2, \ldots, n.$$

Thomas algorithm

O(k n) algorithm
(k number of finite off-diagonals)

Lab today & Thursday

Implement a GE solver for the stationary inhomogeneous heat (diffusion) equation

- use Gaussian profile and parameters given above
- use rectangular heat sink in the center with depth $\boldsymbol{\theta}$ and width a, for $T_0<T_N$, $T_0>T_N$, $T_0=T_N$
- N=10, 100, 1000