Numerical Methods in ESP

Numerical Methods II

Graziano Giuliani

International Centre for Theoretical Physics

Second Semester 2023-24

/afs/ictp.it/public/g/ggiulian/WORLD/num2_lesson6.pdf

Heat transfer equation

- Consider the linear diffusion equation:

$$\frac{\partial \phi}{\partial t} = K \frac{\partial^2 \phi}{\partial x^2} \tag{1}$$

  where K is a constant.

- Using a second-order accurate centered difference scheme, with $j \in [0, N]$: we have discretized [1] in the last lesson as:

$$\frac{\phi_j^{n+1} - \phi_j^n}{\Delta t} = K \left(\frac{\phi_{j+1}^n - 2\phi_j^n + \phi_{j-1}^n}{\Delta x^2} \right) \tag{2}$$

  with Dirichlet boundary condition:

$$\begin{aligned} x_N &= x_0 + N\Delta x \\ \phi(x_0) &= \phi_0 \qquad \forall t \\ \phi(x_N) &= \phi_N \qquad \forall t \end{aligned} \tag{3}$$

Implicit formulation

We have here evaluated the right hand size at time n. What happens if we evaluate it instead at time $n+1$?

- Implicit formulation:

$$\frac{\phi_j^{n+1} - \phi_j^n}{\Delta t} = K \left(\frac{\phi_{j+1}^{n+1} - 2\phi_j^{n+1} + \phi_{j-1}^{n+1}}{\Delta x^2} \right) \tag{4}$$

- Rearranging [4]:

$$-\alpha\phi_{j+1}^{n+1} + (1+2\alpha)\phi_j^{n+1} - \alpha\phi_{j-1}^{n+1} = \phi_j^n \tag{5}$$

with the Dirichlet boundary conditions in [3] and:

$$\alpha = \frac{K\Delta t}{\Delta x^2} \tag{6}$$

Can we solve this problem?

Matrix Form 1

We can express the problem over our domain in matrix form:

$$
\begin{bmatrix}
-\alpha & 1+2\alpha & -\alpha & 0 & 0 & 0 & \dots \\
0 & -\alpha & 1+2\alpha & -\alpha & 0 & 0 & \dots \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots \\
\dots & \dots & 0 & -\alpha & 1+2\alpha & -\alpha & 0 \\
\dots & \dots & 0 & 0 & -\alpha & 1+2\alpha & -\alpha
\end{bmatrix}
\begin{bmatrix}
\phi_0^{n+1} \\
\phi_1^{n+1} \\
\dots \\
\phi_{N-1}^{n+1} \\
\phi_N^{n+1}
\end{bmatrix}
=
\begin{bmatrix}
\phi_1^{n} \\
\dots \\
\phi_{N-1}^{n}
\end{bmatrix}
\tag{7}
$$

Because the matrix it is not square, we cannot invert it. But we can take advantage of the boundary conditions and rewrite the problem in a different form.

Matrix Form 2

$$
\begin{bmatrix}
1+2\alpha & -\alpha & 0 & \dots \\
-\alpha & 1+2\alpha & -\alpha \dots & \\
\dots & \dots & \dots & \dots \\
\dots & -\alpha & 1+2\alpha & -\alpha \\
\dots & 0 & -\alpha & 1+2\alpha
\end{bmatrix}
\begin{bmatrix}
\phi_1^{n+1} \\ \phi_2^{n+1} \\ \dots \\ \phi_{N-1}^{n+1}
\end{bmatrix}
+
\begin{bmatrix}
-\alpha\phi_0^{n+1} \\ 0 \\ 0 \\ \dots \\ 0 \\ -\alpha\phi_N^{n+1}
\end{bmatrix}
=
\begin{bmatrix}
\phi_1^{n} \\ \phi_2^{n} \\ \dots \\ \phi_{N-1}^{n}
\end{bmatrix}
\tag{8}
$$

In this way we can "recover" the solution:

$$\phi^{n+1} = \mathbf{M}^{-1}\left(\phi^n - \phi_{\mathbf{b}}\right) \tag{9}$$

Let us now look at a "smart way" to solve the problem in [8] called the Thomas Algorithm or Tri-Diagonal Matrix Algorithm, (TDMA).

Tri Diagonal Matrix Form

The generic Tri-diagonal Matrix can be expressed as:

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & 0 & \dots \\ a_2 & b_2 & c_2 & 0 & 0 & \dots \\ 0 & a_3 & b_3 & c_3 & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & a_{N-3} & b_{N-3} & c_{N-3} & 0 \\ \dots & 0 & 0 & a_{N-2} & b_{N-2} & c_{N-2} \\ \dots & 0 & 0 & 0 & a_{N-1} & b_{N-1} \end{bmatrix} \tag{10}$$

Because of all the zeroes, the information present can be stored in just three arrays:

$$\mathbf{A}(2:N-1), \quad \mathbf{B}(1:N-1), \quad \mathbf{C}(1:N-2) \tag{11}$$

containing the diagonal $(\mathbf{B})$, lower diagonal $(\mathbf{A})$ and upper diagonal $(\mathbf{C})$ elements.

Thomas algorithm

The Thomas algorithm has two steps:

- Forward elimination
- Back substitution

The first, forward elimination, aims at setting to 0 all elements of the lower diagonal $\mathbf{A}$ and to 1 all elements of the diagonal $\mathbf{B}$ array. The second, back substitution, loops in the reverse order over the elements giving us the solution.

Forward Elimination I

Let us rewrite the problem in [8] in a generic form using the above notation in [10] for the generic matrix $\mathbf{M}$.

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & \dots \\ a_2 & b_2 & c_2 & 0 & \dots \\ 0 & a_3 & b_3 & c_3 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & a_{N-2} & b_{N-2} & c_{N-2} \\ \dots & 0 & 0 & a_{N-1} & b_{N-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} d_1 - a_1 u_0 \\ d_2 \\ d_3 \\ \dots \\ d_{N-2} \\ d_{N-1} - c_N u_N \end{bmatrix} \tag{12}$$

This is the generic form of the problem in [4] for the Thomas algorithm with Dirichlet boundary conditions at u_0 and u_N.

Forward Elimination II

First step is to divide the first line by b_1:

$$\begin{bmatrix} 1 & F_1 & 0 & 0 & \dots \\ a_2 & b_2 & c_2 & 0 & \dots \\ 0 & a_3 & b_3 & c_3 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & a_{N-2} & b_{N-2} & c_{N-2} \\ \dots & 0 & 0 & a_{N-1} & b_{N-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} \delta_1 \\ d_2 \\ d_3 \\ \dots \\ d_{N-2} \\ d_{N-1} - c_N u_N \end{bmatrix} \tag{13}$$

where now:

$$\begin{aligned} F_1 &= \frac{c_1}{b_1} \\ \delta_1 &= \frac{d_1 - a_1 u_0}{b_1} \end{aligned} \tag{14}$$

Forward Elimination III

Multiply the first line by $a2$ and subtract it to the second line:

$$\begin{bmatrix} 1 & F_1 & 0 & 0 & \dots \\ 0 & b_2 - a_2 F_1 & c_2 & 0 & \dots \\ 0 & a_3 & b_3 & c_3 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & a_{N-2} & b_{N-2} & c_{N-2} \\ \dots & 0 & 0 & a_{N-1} & b_{N-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} \delta_1 \\ d_2 - a_2 \delta_1 \\ d_3 \\ \dots \\ d_{N-2} \\ d_{N-1} - c_N u_N \end{bmatrix} \tag{15}$$

Forward Elimination IV

Divide now the second line by $b_2 - a_2 F_1$ to obtain:

$$\begin{bmatrix} 1 & F_1 & 0 & 0 & \dots \\ 0 & 1 & F_2 & 0 & \dots \\ 0 & a_3 & b_3 & c_3 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & a_{N-2} & b_{N-2} & c_{N-2} \\ \dots & 0 & 0 & a_{N-1} & b_{N-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ d_3 \\ \dots \\ d_{N-2} \\ d_{N-1} - c_N u_N \end{bmatrix} \tag{16}$$

where now:

$$\begin{aligned} F_2 &= \frac{c_2}{b_2 - a_2 F_1} \\ \delta_2 &= \frac{d_2 - a_2 \delta_1}{b_2 - a_2 F_1} \end{aligned} \tag{17}$$

Forward Elimination V

Got the gist? Multiply the second line by a_3 and subtract it to the third line:

$$\begin{bmatrix} 1 & F_1 & 0 & 0 & \dots \\ 0 & 1 & F_2 & 0 & \dots \\ 0 & 0 & b_3 - a_3 F_2 & c_3 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & a_{N-2} & b_{N-2} & c_{N-2} \\ \dots & 0 & 0 & a_{N-1} & b_{N-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ d_3 - a_3 \delta_2 \\ \dots \\ d_{N-2} \\ d_{N-1} - c_N u_N \end{bmatrix} \tag{18}$$

What is next step? Divide now the third line by $b_3 - a_3 F_2$. And so on.

Forward Elimination VI

Writing down recursive formulas for the F and δ coefficients as:

$$F_j = \frac{c_j}{b_j - a_j F_{j-1}} \tag{19}$$

$$\delta_j = \frac{d_j - a_j \delta_{j-1}}{b_j - a_j F_{j-1}} \tag{20}$$

we reach the formulation:

$$\begin{bmatrix} 1 & F_1 & 0 & 0 & \dots \\ 0 & 1 & F_2 & 0 & \dots \\ 0 & 0 & 1 & F_3 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & 0 & 1 & F_{N-2} \\ \dots & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \dots \\ \delta_{N-2} \\ \delta_{N-1} - F_{N-1} u_N \end{bmatrix} \tag{21}$$

Last equation is now solvable!

Back Substitution

Because we have computed all δ_j and F_j, from the last line we have:

$$u_{N-1} = \delta_{N-1} - F_{N-1} u_N \tag{22}$$

Great! But now that we know u_{N-1}, we can compute u_{N-2} from the line above:

$$u_{N-2} = \delta_{N-2} - F_{N-2} u_{N-1} \tag{23}$$

and so on for every index j:

$$u_j = \delta_j - F_j u_{j+1} \tag{24}$$

Let us now recap the method in a formula we can readily apply.

TDM Algorithm

Given a problem defined by a tri-diagonal matrix as in [10], we can solve it by following this algorithm:

- Set $F_0 = 0, \quad \delta_0 = u_0$.
- Compute the coefficients F_j and δ_j using the formulas in [19] and [20] for $j \in \mathbb{N}[1, N-1]$:

$$\begin{aligned} F_j &= \frac{c_j}{b_j - a_j F_{j-1}} \\ \delta_j &= \frac{d_j - a_j \delta_{j-1}}{b_j - a_j F_{j-1}} \end{aligned}$$

- Apply the backward substitution in [24] to compute the solution taking into account that we have u_N:

$$u_j = \delta_j - F_j u_{j+1} \quad j \in \mathbb{N}[N-1, 1]$$

Diffusion problem with TDMA

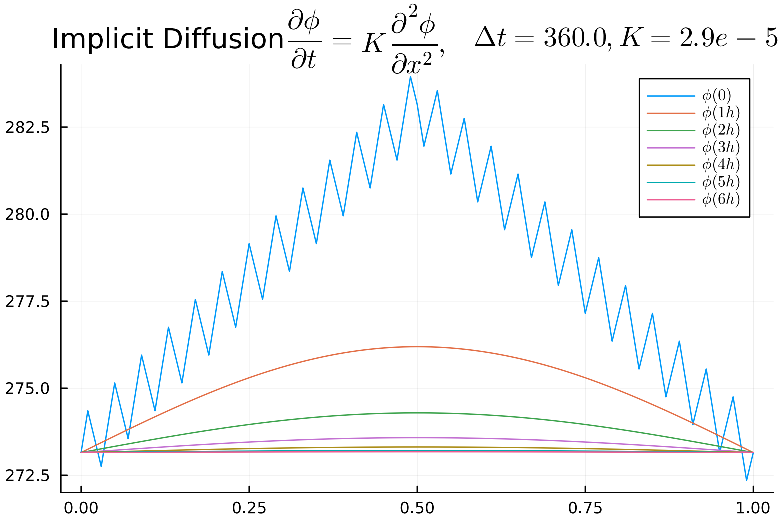
Now that we have a working algorithm, we can apply the method to compute the solution to the fully implicit problem of the discretized numerical solution for the diffusion equation in [4]. In this problem, from the formulation in [8] we can derive the elements of the arrays $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, using the algorithm in slide (15) to compute the solution via the forward elimination coefficients F and δ by applying the back substitution. Note that the implicit diffusion scheme is unconditionally stable. Which timestep would you choose in the following exercise? ☺

Exercise on Diffusion - TDMA

- Use the fully implicit scheme in [4] to solve the diffusion problem in [1] following the TDMA algorithm described in slide (15). Use a spatial resolution of $\Delta x = 0.01m$ with a diffusion coefficient $K = 2.9E^{-5}$. Integrate for at least 6 hours and show the solution every hour. Let the initial condition be the following function, describing the temperature distribution along a $1m$ metal rod heated in the middle point and with extrema kept at a constant temperature of $T_0 = 273.15K$:

$$\begin{aligned}
\phi(x,0) &= \begin{cases} 273.15 + 20x + \sin(50\pi x) & for \quad 0 \leq x \leq 0.5 \\ 273.15 + 20 - 20x + \sin(50\pi x) & for \quad 0.5 < x \leq 1 \end{cases} \\
\phi(0,t) &= 273.15, \quad \forall t \\
\phi(1,t) &= 273.15, \quad \forall t
\end{aligned} \tag{25}$$

Expected result



Julia Code

```
alpha = K*dt/(dx^2)
A = fill(-alpha,nx-2)
C = fill(-alpha,nx-2)
B = fill(1+2.0*alpha,nx-1)
F = similar(x,nx-1)
delta = similar(x,nx-1)
function diffusion(phi_now,phi_new)
  F[1] = 0
  delta[1] = temp0
  for j in eachindex(C)
    F[j+1] = C[j] / (B[j+1] - A[j] * F[j])
    delta[j+1] = (phi_now[j+1] - A[j] * delta[j]) /
                 (B[j+1] - A[j] * F[j])
  end;
  phi_new[nx] = temp0
  for j in reverse(eachindex(F))
    phi_new[j] = delta[j] - F[j] * phi_new[j+1]
  end;
end;
```