Numerical Methods for Partial Differential Equations
A.Y. 2024/2025

Laboratory 06
Finite Element method for the heat equation

Exercise 1.

Let $\Omega = (0,1)^3$ be the unit cube and $T > 0$. Let us consider the following time-dependent problem:

$$\begin{cases} \dfrac{\partial u}{\partial t} - \boldsymbol{\nabla} \cdot (\mu \boldsymbol{\nabla} u) = f & \text{in } \Omega \times (0,T), & \text{(1a)} \\ \mu \boldsymbol{\nabla} u \cdot \mathbf{n} = 0 & \text{on } \partial\Omega \times (0,T), & \text{(1b)} \\ u = u_0 & \text{in } \Omega \times \{0\}, & \text{(1c)} \end{cases}$$

where $\mathbf{x} = (x,y,z)^T$, $\mu = 0.1$, $f(\mathbf{x},t) = 0$ and

$$u_0(\mathbf{x}) = x(x-1)\,y(y-1)\,z(z-1)\ .$$

1.1. Write the weak formulation to problem (1), and derive the semi-discrete formulation with the finite element method.

Solution. Let $V = H^1(\Omega)$. The weak formulation reads:

$$\begin{aligned} &\text{for all } t \in (0,T)\ ,\ \text{find } u(t) \in V \text{ such that } u(0) = u_0 \text{ and} \\ &\int_\Omega \frac{\partial u}{\partial t} v\,d\mathbf{x} + \underbrace{\int_\Omega \mu\, \boldsymbol{\nabla} u \cdot \boldsymbol{\nabla} v\,d\mathbf{x}}_{a(u,v)} = \underbrace{\int_\Omega f\,v\,d\mathbf{x}}_{F(v)} \quad \text{for all } v \in V\ . \end{aligned}$$

Let us now introduce a mesh $\mathcal{T}_h$ over the domain Ω, and let $V_h = V \cap X_h^r(\Omega)$ be the associated finite element space of degree r. We obtain the semi-discrete formulation by looking for $u(t) \in V_h$ and testing against functions $v \in V_h$:

$$\begin{aligned} &\text{for all } t \in (0,T)\ ,\ \text{find } u_h(t) \in V_h \text{ such that } u_h(0) = u_{0,h} \text{ and} \\ &\int_\Omega \frac{\partial u_h}{\partial t} v\,d\mathbf{x} + a(u_h, v_h) = F(v_h) \quad \text{for all } v_h \in V_h\ . \end{aligned} \tag{2}$$

In the above, $u_{0,h} \in V_h$ is an approximation of u_0.

Introducing the basis functions $\{\varphi_i\}_{i=1}^{N_h}$ of the space V_h, the semi-discrete formulation (2) can be rewritten as the following system of ordinary differential equations (ODEs):

$$\begin{cases} M\dfrac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} + A\mathbf{u}(t) = \mathbf{f}(t) \quad \text{for } t \in (0,T] \,, \\ \mathbf{u}(0) = \mathbf{u}_0 \,, \end{cases}$$

where

- $\mathbf{u}(t) = (U_1(t), U_2(t), \ldots, U_{N_h}(t))^T$ is the vector of the time-dependent control variables;
- M is the *mass matrix*, whose entries are $M_{ij} = \displaystyle\int_\Omega \varphi_i\,\varphi_j\,d\mathbf{x}$;
- A is the matrix associated to the bilinear form $a(u,v)$ (also known as *stiffness matrix*), whose entries are $A_{ij} = \displaystyle\int_\Omega \mu\,\boldsymbol{\nabla}\varphi_i \cdot \boldsymbol{\nabla}\varphi_j\,d\mathbf{x}$;
- $\mathbf{f}(t)$ is the vector associated to the forcing term f, whose entries are $\mathbf{f}_i(t) = \displaystyle\int_\Omega f\,\varphi_i\,d\mathbf{x}$;
- $\mathbf{u}_0$ is the vector of the control variables for $u_{0,h}$.

1.2. Write the fully discrete formulation of problem (1) using the θ-method to discretize the time derivative.

Solution. Let us introduce a partition of the time interval into N_T sub-intervals $(t^n, t^{n+1}]$ of width Δt, with $n = 0, 1, 2, \ldots, N_T - 1$ and $t^0 = 0$, $t^{N_T} = T$. We denote with a superscript n the approximation of the solution at the discrete time t^n, i.e. $\mathbf{u}^n \approx \mathbf{u}(t^n)$. Let $\theta \in [0,1]$ be the parameter of the theta method. Then, the fully discrete formulation is the following:

$$\begin{cases} M\dfrac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \theta A\mathbf{u}^{n+1} + (1-\theta)A\mathbf{u}^n = \theta\mathbf{f}^{n+1} + (1-\theta)\mathbf{f}^n \quad \text{for } n = 0,1,2,\ldots,N_T-1 \,, \\ \mathbf{u}^0 = \mathbf{u}_0 \,, \end{cases}$$

which can be rewritten as

$$\begin{cases} \left(\dfrac{M}{\Delta t} + \theta A\right)\mathbf{u}^{n+1} = \left(\dfrac{M}{\Delta t} - (1-\theta)A\right)\mathbf{u}^n + \theta\mathbf{f}^{n+1} + (1-\theta)\mathbf{f}^n \quad \text{for } n = 0,1,2,\ldots,N_T-1 \,, \\ \mathbf{u}^0 = \mathbf{u}_0 \,. \end{cases}$$

For $\theta = 0$, the method falls back to the explicit (or forward) Euler method. For $\theta = 1$, the method is the implicit (or backward) Euler method. For $\theta = \dfrac{1}{2}$, the method is known as Crank-Nicolson method.

1.3. Implement in `deal.II` a finite element solver for problem (1) using the theta method to approximate the time derivative.

Solution. See `src/lab-06.cpp` for the implementation. Below we provide an overview of the code. More details can be found in comments within the source files.

The `Heat` class, that manages the solution of problem (1), has a `setup` method that takes care of initialization, whose contents are very similar to previous exercises.

The solution of the problem is carried out by the `Heat::solve` method. Within its body, we first apply the initial conditions, by interpolating the u_0 function onto the mesh (i.e. the function is evaluated on all mesh nodes, and the resulting values are the control variables of $u_{0,h}$). Then, we enter into a `while` loop that advances in time. Within the loop, at each iteration, we assemble and solve the problem for one time step.

The assembly is actually split in two: the assembly of relevant matrices (M, A, $\dfrac{M}{\Delta t}+\theta A$ and $\dfrac{M}{\Delta t} - (1-\theta)A$) and the assembly of the right-hand side of the algebraic system. Indeed, the latter changes over time (since it depends on $\mathbf{u}^n$ and on the time-dependent forcing term), and so it needs to be reassembled at every temporal iteration. Conversely, the matrices are always the same: therefore, it is computationally more efficient to assemble them only once outside of the temporal loop. The assembly is carried out by the methods `assemble_matrices` and `assemble_rhs`.

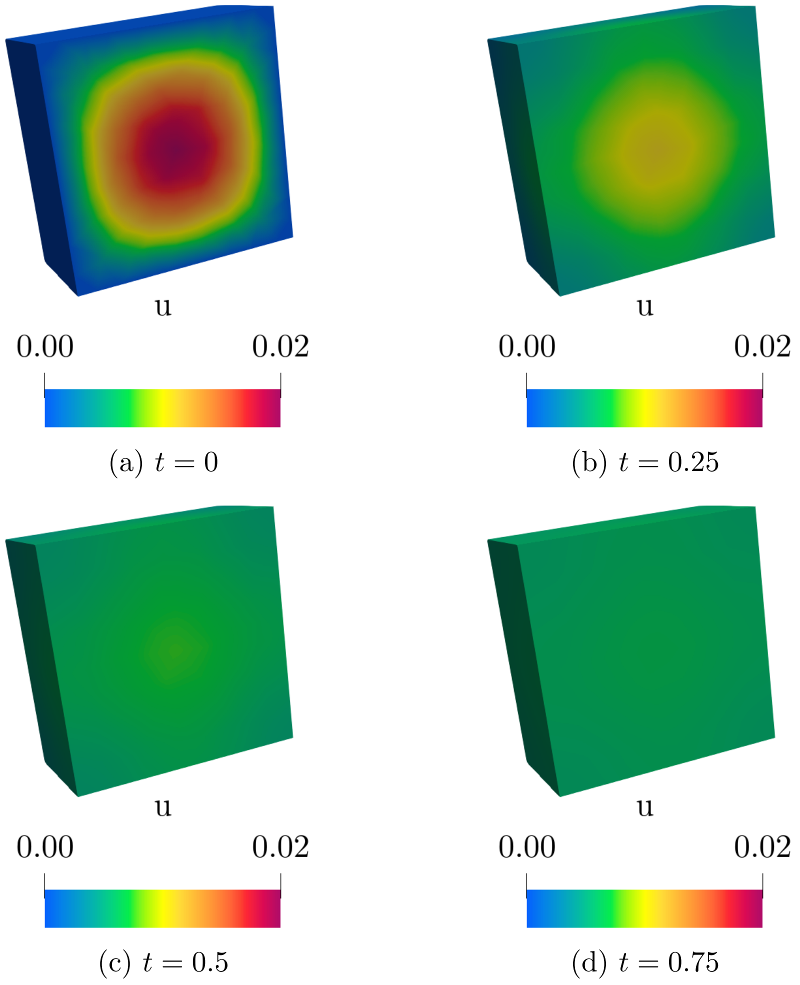
Finally, the `output` method has been slightly modified so that the output file names are numbered according to the current timestep, so that in the end we obtain a sequence of files, numbered according to the time step. Paraview can recognize and open these sequence of files, allowing to visualize a time-dependent solution.

1.4. Using the implicit Euler method (i.e. setting $\theta = 1$), compute the solution to the problem (1). Set $T = 1$, $\Delta t = 0.05$ and using linear polynomials (degree $r = 1$) on the mesh `mesh/mesh-cube-10.msh`.

Opening the solution in Paraview:

1. plot the solution along the line $y = z = \dfrac{1}{2}$;
2. plot the solution over time at one point on the domain;
3. compute the integral $\displaystyle\int_\Omega u\,d\mathbf{x}$ at times $t = 0$ and $t = T$; what can you observe?

Solution. The solution is displayed at four different times in Figure 1.



(a) $t = 0$ (b) $t = 0.25$

(c) $t = 0.5$ (d) $t = 0.75$

Figure 1: Plot of the solution to Exercise 1 at four different times. The domain was clipped along the plane $z = 0.5$ using the Paraview filter “Clip”.

The plot at a point over time is obtained using the Paraview filters “Probe location” (to select the point) and “Plot data over time” (to obtain the plot). Figure 2 shows the result at point $\mathbf{x} = (0.5, 0.5, 0.5)$. We can observe how the solution decays over time as the initial conditions is spread over the domain.

The plot along the line is obtained through the “Plot over line” filter. Figure 3 shows the result at different times. We can appreciate how the initial condition is spreaded over the whole domain. This is consistent with the physical interpretation of problem (1) as the model for the temperature of a solid, where heat diffuses from hotter areas towards cooler ones, so that the former become colder while the latter become hotter until equilibrium is reached.

The integral of the solution can be computed in Paraview using the filter “Integrate

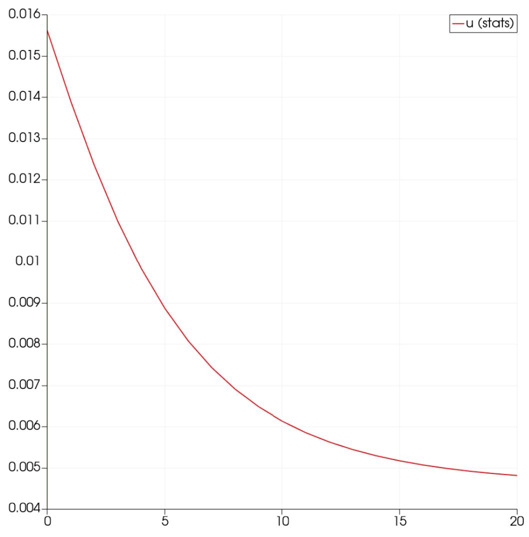


Figure 2: Plot of the solution at point $\mathbf{x} = (0.5, 0.5, 0.5)$ over time. The plot is obtained combining the Paraview filters "Probe location" and "Plot data over time".

variables". We obtain

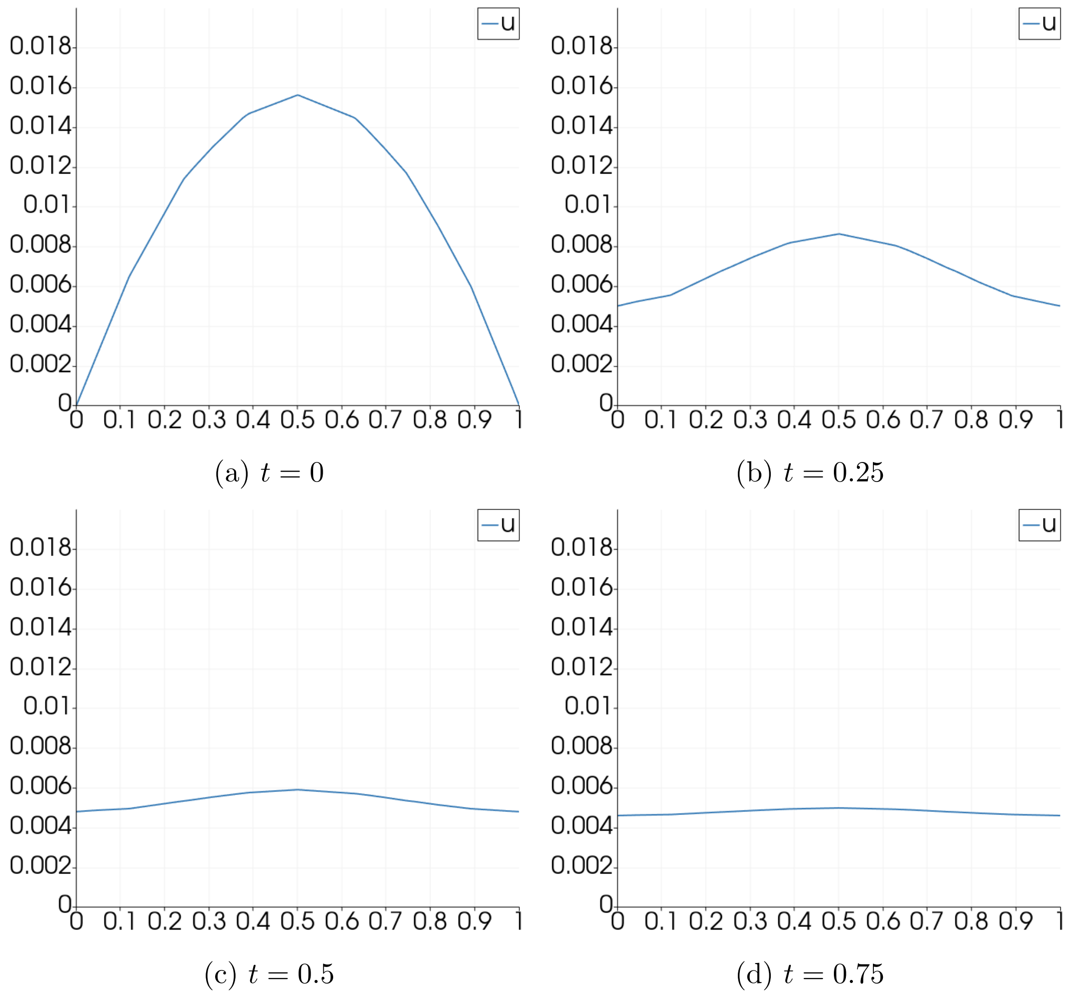
$$\int_\Omega u\, d\mathbf{x} = 0.004\,454\,06 \qquad \text{at } t = 0.0\ ,$$
$$\int_\Omega u\, d\mathbf{x} = 0.004\,454\,06 \qquad \text{at } t = 0.2\ .$$

The two integrals are the same (at least up to 6 significant digits). Indeed, the homogeneous Neumann boundary conditions express the fact that no heat can enter or exit through the boundary. Since the forcing term is zero, and the diffusion term cannot generate or absorb energy, the total energy of the system must be conserved. More formally, we have:

$$\int_\Omega \frac{\partial u}{\partial t}\, d\mathbf{x} - \int_\Omega \boldsymbol{\nabla} \cdot (\mu \boldsymbol{\nabla} u)\, d\mathbf{x} = 0$$
$$\frac{\partial}{\partial t} \int_\Omega u\, d\mathbf{x} = \int_\Omega \boldsymbol{\nabla} \cdot (\mu \boldsymbol{\nabla} u)\, d\mathbf{x} = \int_{\partial\Omega} \mu \boldsymbol{\nabla} u \cdot \mathbf{n} d\sigma = 0\ ,$$

so that the integral of the solution should not change over time. We have empirically verified that our solver satisfies this conservation property.

1.5. Compute the solution to the problem using the explicit Euler method (i.e. setting $\theta = 0$), using the same discretization settings as in previous question.



(a) $t = 0$ (b) $t = 0.25$

(c) $t = 0.5$ (d) $t = 0.75$

Figure 3: Plot of the solution to Exercise 1 along the line $y = z = \dfrac{1}{2}$ at four different times. The plot is obtained through the Paraview filter “Plot over line”.

Solution. Using the same settings as before, but with $\theta = 0$, the numerical solution diverges: the solver computes a numerical solution with very large oscillations and no physical meaning.

Indeed, while the implicit Euler method is unconditionally absolutely stable, the explicit Euler method is only conditionally stable, under a condition of the type $\Delta t < Ch^2$, with C depending on the domain and coefficients of the problem.

This condition can be very stringent. For the particular problem considered here, we can empirically verify that stability is obtained by setting $\Delta t = 0.0025$, while $\Delta t = 0.005$ still yields an unstable solution.

Possibilities for extension

Time adaptivity. We considered a partition of the time domain where all intervals have the same size. However, in practice, it can be useful to choose small time steps during times characterized by fast dynamics (rapid changes), and larger time steps when the dynamics are slow, to save computational time while maintaining accuracy. Even better, the time step can be selected dynamically, depending on the solution itself. This is the idea underlying time-adaptive methods.

Start by modifying the code of Exercise 1 so that it allows a time-varying time step size, but prescribing the evolution of the time step as an input. Then, modify it to automatically choose the time step so that it is smaller when dynamics are fast, and larger when they are slow.