Report for Numerical Methods for Partial Differential Equation project

Space and time adaptivity

Author: Domenico De Giorgio 10854350, Bianca Maria Marchettini 11098150, Giacomo Merlo 10848938, Aurora Perlini 10788118

Professor: Prof. Alfio Maria Quarteroni, Prof. Michele Bucelli

Academic year: 2025-2026

1. Introduction

In time-dependent PDEs, sharp spatial gradients may appear only in small regions and only during limited time intervals. Using a uniformly fine mesh and a uniformly small time step can therefore be unnecessarily expensive. Adaptive methods address this by increasing resolution only where and when it is needed.

This report presents a `deal.II` implementation [1, 2] of a finite element solver for the two-dimensional heat equation and a systematic comparison between homogeneous and adaptive discretizations. The emphasis is on quantifying how spatial mesh adaptation [5] and time-step adaptation affect both accuracy and runtime, under the same physical setting.

2. Problem statement and goals

We solve the heat equation on a bounded domain $\Omega \subset \mathbb{R}^2$ for $t \in (0, T]$, starting from a zero initial condition and imposing homogeneous Neumann boundary conditions. The problem is driven by a separable forcing term $f(x,t) = g(t)\,h(x)$, where $h(x)$ is a Gaussian centered at x_0 with width σ, and $g(t)$ introduces oscillatory impulses controlled by the parameter N.

The objectives are:

- implement spatial adaptivity via adaptive mesh refinement/coarsening (AMR) and time adaptivity via a variable time step controller;
- compare the four configurations against a numerical reference solution computed with a finer fixed discretisation (global mesh refinement increased by 2 levels and time step reduced by a factor 10);
- assess the accuracy–cost trade-off using the final-time L^2 error and the runtime statistics produced by the code (CPU time, linear solver iterations, time-step statistics, and mesh complexity in terms of active cells and DoFs).

3. Mathematical model

3.1. Domain and boundary conditions

We consider a bounded domain $\Omega \subset \mathbb{R}^2$ and a final time $T > 0$. The unknown $u = u(x,t)$ satisfies

$$\begin{cases} \dfrac{\partial u}{\partial t} - \nabla \cdot (\mu \nabla u) = f & \text{in } \Omega \times (0,T), \\ \mu \nabla u \cdot n = 0 & \text{on } \partial\Omega \times (0,T), \\ u = 0 & \text{in } \Omega \times \{0\}. \end{cases} \tag{1}$$

where n denotes the outward unit normal on $\partial\Omega$.

3.2. Forcing term

The forcing term is separable in time and space:

$$\begin{aligned} f(x,t) &= g(t)\,h(x), \\ g(t) &= \frac{\exp(-a\cos(2N\pi t))}{\exp(a)}, \\ h(x) &= \exp\left(-\frac{\|x-x_0\|^2}{\sigma^2}\right). \end{aligned} \tag{2}$$

In our experiments we fix $a = 5$. The parameters T, $N \in \mathbb{N}^+$, $\sigma > 0$, and $x_0 = (x_{0,x}, x_{0,y}) \in \Omega$ are treated as user-defined inputs.

4. Weak formulation

Let $V := H^1(\Omega)$. Multiplying (1) by a test function $v \in V$ and integrating over Ω, the diffusion term is integrated by parts. The homogeneous Neumann boundary condition makes the boundary integral vanish, and we obtain the variational problem: for $t \in (0,T]$, find $u(t) \in V$ such that

$$(\partial_t u(t), v) + (\nabla u(t), \nabla v) = (f(t), v) \qquad \forall v \in V, \tag{3}$$

with initial condition $u(\cdot,0) = 0$. Here $(\cdot,\cdot)$ denotes the $L^2(\Omega)$ inner product.

5. Numerical discretization

5.1. Spatial discretization (FEM: Q_1)

We discretize in space using continuous Q_1 finite elements on a quadrilateral mesh. Let $V_h \subset H^1(\Omega)$ be the corresponding FE space and let $u_h(t) = \sum_j U_j(t)\,\phi_j \in V_h$ be the discrete solution. The semi-discrete problem can be written in matrix form as

$$M\dot{U}(t) + KU(t) = F(t), \tag{4}$$

where M is the (consistent) mass matrix, K is the Laplace (diffusion) matrix, and $F(t)$ is the load vector obtained from the forcing term.

5.2. Time discretization (θ-method / Crank–Nicolson)

For a time step $\Delta t = t^{n+1} - t^n$, the θ-method applied to $M\dot{U}(t) + KU(t) = F(t)$ yields

$$\begin{aligned} \big(M + \theta\Delta t\,K\big)U^{n+1} = \big(M - (1-\theta)\Delta t\,K\big)U^n \\ + \Delta t\Big(\theta F^{n+1} + (1-\theta)F^n\Big). \end{aligned} \tag{5}$$

In our implementation we set $\theta = 0.5$ (Crank–Nicolson).

6. Linear system solution

At each time step, the θ-method leads to a symmetric positive definite linear system

$$\big(M + \theta\Delta t\,K\big)U^{n+1} = b^n,$$

which we solve with the Conjugate Gradient (CG) method. To accelerate convergence we use an SSOR preconditioner built from the current system matrix. During the run we record the number of CG iterations per solve and aggregate statistics (min/max/sum) to compare the computational effort across different configurations.

6.1. Stopping criterion and tolerances

The stopping tolerance is chosen relative to the size of the right-hand side in order to keep a consistent accuracy level throughout the simulation. Specifically, we use a residual tolerance

$$\mathrm{tol} = \max\big(10^{-14},\, 10^{-8}\,\|b^n\|_2\big)\,,$$

with a maximum of 1000 CG iterations per solve. This criterion avoids over-solving when the right-hand side is small while still enforcing a strict absolute bound on the residual.

7. Space adaptivity (AMR)

Adaptive Mesh Refinement (AMR) refers to dynamically refining and coarsening the computational mesh during the simulation in order to concentrate degrees of freedom where the solution has the largest spatial variations.
In our solver, AMR is performed at selected times and is driven by a cellwise error indicator computed from the current discrete solution.

7.1. Error indicator (Kelly estimator)

We estimate the spatial error using the Kelly error estimator, which is a residual-based indicator built from jumps of the normal derivative across cell faces. This provides one scalar indicator value per active cell, used to rank regions requiring refinement or coarsening.

7.2. Refinement/coarsening strategy and level bounds

Given the indicator values, we apply a fixed-fraction strategy: at each adaptation event we mark 60% of the cells with the largest indicators for refinement and 40% of the cells with the smallest indicators for coarsening. To keep the mesh complexity under control, we enforce minimum and maximum grid levels by clearing refinement flags above the maximum level and clearing coarsening flags below the minimum level.

7.3. Solution transfer after mesh changes

After executing refinement and coarsening, the discrete solution must be transferred to the new finite element space. We use `SolutionTransfer` to interpolate the pre-adaptation solution onto the adapted mesh, and we re-apply constraints before continuing the time integration. After each adaptation step, the system setup is rebuilt on the new mesh (DoFs, sparsity pattern, and matrices).

7.4. Pre-refinement phase

Before starting the main time loop, we optionally perform a short pre-refinement procedure to construct an initial mesh that already resolves the region influenced by the localized forcing. This phase repeats a probe solve followed by an AMR step for a prescribed number of iterations. After each pre-refinement step, the solution is reset to the initial condition on the refined mesh, and the actual simulation then starts from this adapted initial grid.

8. Time adaptivity

When time adaptivity is enabled, the time step Δt is updated during the simulation to resolve phases with faster temporal dynamics while avoiding unnecessarily small steps when the solution evolves smoothly. We implement two alternative controllers: a step-doubling strategy with accept/reject and a cheaper heuristic controller.

8.1. Step-doubling (accept/reject)

In the step-doubling approach, a candidate step of size Δt is computed in two ways: one full step of size Δt, and two consecutive half steps of size $\Delta t/2$. The difference between the two resulting solutions provides an error estimate. The step is accepted if the estimated error is below a tolerance scaled with the solution norm; otherwise it is rejected and retried with a smaller Δt. After an accepted step, the next time step is updated using a standard controller based on the error ratio.

8.2. Heuristic time-step controller

As a lower-cost alternative, we estimate temporal activity by comparing the forcing term at consecutive times. The controller computes a proxy indicator from the relative change of the discrete right-hand side and then increases or decreases Δt according to simple rules: shrinking the step when the indicator is large and enlarging it when the indicator is small. This mode never rejects a step: it always advances time and performs exactly one linear solve per step. This option avoids step rejection and requires only one linear solve per step, at the price of a less direct control of the solution error.

8.3. Time-step bounds and safety factors

In both approaches, the time step is constrained within predefined bounds $\Delta t_{\min} \leq \Delta t \leq \Delta t_{\max}$ to prevent instability or excessive cost. A safety factor is applied when updating Δt to avoid aggressive changes between consecutive steps. In our implementation we use `time_step_tolerance` $= 10^{-8}$, $\Delta t_{\min} = 10^{-6}$, $\Delta t_{\max} = 10^{-1}$, and `safety` $= 0.9$. For step-doubling, we also record the number of accepted and rejected attempts together with Δt statistics to quantify the controller behavior.

9. Implementation details (deal.II workflow)

This section summarizes the main workflow of the `deal.II` implementation, from mesh construction to linear algebra setup and data export. The solver is organized around a class that manages the triangulation, DoF handler, matrices/vectors, and the time loop.

9.1. Mesh generation/import

The code supports two alternatives. In the generated-mesh mode, the domain is built as a subdivided unit square (via `GridGenerator::subdivided_hyper_cube`) and then globally refined. In the imported mode, the mesh is read from a fixed `.msh` path via `GridIn` and then globally refined in the same way as before. Both options produce a `Triangulation` that can later be modified by AMR when space adaptivity is enabled.

9.2. Assembly and constraints

For a given mesh, degrees of freedom are distributed using `DoFHandler` with Q_1 finite elements. Hanging-node constraints are collected in an `AffineConstraints` object and used to build a consistent sparsity pattern. The consistent mass matrix and the Laplace matrix are assembled with `MatrixCreator` and Gaussian quadrature. At each time step, the system matrix is formed as a linear combination of these matrices, and the right-hand side is assembled from the previous solution and the forcing term. Constraints are applied by condensation before the linear solve and by distributing the solution afterwards.

9.3. Output (VTK) and logs (CSV)

For visualization, the discrete solution is exported in VTK format using `DataOut`, with time and step index stored in the output metadata. In addition, the code writes CSV logs to enable post-processing: a mesh log records active cells, DoFs, and refinement/coarsening activity at each AMR event, while a time log records the time-step evolution (including error estimates and the chosen Δt update). In step-doubling mode the time log can also include accepted/rejected attempts; in the heuristic controller the step is never rejected. A final summary CSV aggregates accuracy and performance metrics for each run configuration.

10. Experimental setup

All experiments solve the same heat equation with the forcing parameters (T, N, σ, x_0) provided at runtime. The comparison is performed automatically by the `main()` routine, which first computes a numerical reference solution on a finer discretization and then runs four configurations obtained by switching space and/or time adaptivity on and off.

10.1. Configurations compared (fixed/adaptive space/time)

The following runs are executed:

- `fixed_space_fixed_time`
- `adaptive_space_fixed_time`
- `fixed_space_adaptive_time`
- `adaptive_space_adaptive_time`

All four runs use the same baseline settings for the initial time step and mesh parameters (initial global refinement, pre-refinement iterations, and refinement frequency). They differ only in whether AMR and/or time adaptivity are enabled; when time adaptivity is enabled, the controller type (step-doubling or heuristic) is selected once at runtime and applied to the time-adaptive runs.

10.2. Reference solution construction

A numerical reference is computed before the four runs. In the code, the reference uses a finer discretization obtained by reducing the initial time step (one order of magnitude smaller than the baseline) and increasing the global refinement level. When a generated mesh is used, the reference run also doubles the number of initial cells per direction. The reference solution at $t = T$ is stored and exposed as a finite element field function to enable consistent error evaluation.

10.3. Error metric (L^2 at final time)

For each configuration we measure the final-time error as

$$E(T) = \|u_h(\cdot, T) - u_{\text{ref}}(\cdot, T)\|_{L^2(\Omega)}.$$

In the implementation this quantity is computed by numerical quadrature through `VectorTools::integrate_difference` with the `L2_norm` option.

10.4. Cost metrics collected

Computational cost is evaluated using the run statistics collected by the solver: total CPU time; number of linear solves; CG iteration counts (min/max/sum); time-stepping statistics (total attempts, accepted/rejected counts, and Δt min/max/mean); and mesh complexity indicators (DoFs min/max/mean and active cells min/max during the run). These quantities are written to `summary_comparison.csv` for the final comparison.

11. Results

This chapter reports the quantitative results of this project by comparing four solver configurations that differ only in the use of spatial adaptivity (AMR) and/or time-step adaptivity. The goal is to assess the accuracy–cost trade-off under the same PDE setting and forcing parameters.

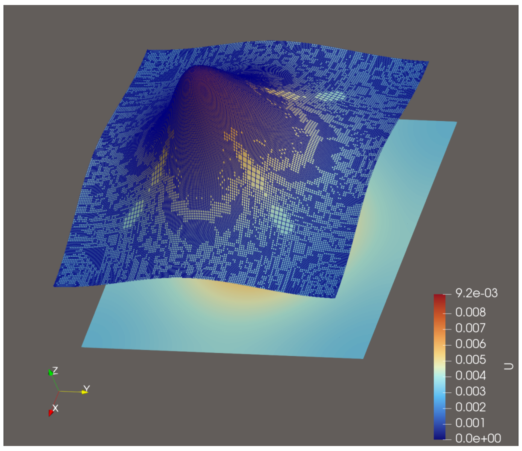

Figure 1: Paraview render of the solution with WarpByScalar filter with sigma=0.1, n=2 e a=2

11.1. Test 1

11.1.1 Run setup and user-selected inputs

To ensure reproducibility, the code is driven by a small set of interactive inputs that define the mesh source, the forcing term, the final time, and (when time adaptivity is enabled) the time-step control strategy.
In the runs reported here the following inputs were used:

- Mesh: generated internally (`s`), with `cells_per_direction` = 3 (uniform base grid).
- Time adaptivity strategy: heuristic controller (`no` to step-doubling).
- Final time: $T \in \{0.5,\ 0.1\}$ (two separate experiments with identical remaining parameters).
- Forcing parameters: $\sigma = 0.50$, $x_0 = (0.50, 0.50)$, $N = 5$.

All other algorithmic parameters are kept fixed across configurations.

11.1.2 Compared configurations and metrics

The comparison includes the following configurations:

- `fixed_space_fixed_time`: uniform mesh, fixed Δt (baseline).
- `adaptive_space_fixed_time`: spatial AMR, fixed Δt.
- `fixed_space_adaptive_time`: uniform mesh, adaptive Δt (heuristic controller).
- `adaptive_space_adaptive_time`: spatial AMR and adaptive Δt (heuristic controller).

For each run we report: number of accepted time steps, mean step size Δt_{mean} (and, when adaptive, the observed range), mesh growth through DoFs, CPU time, and final-time error $\|e(T)\|_{L^2}$ against a finer reference solution.

11.1.3 Summary tables

Tables 1 and 2 summarise the key quantities extracted from `summary_comparison.csv` for the two final times considered. In particular, they report:

- **Time-integration effort:** number of time steps and Δt_{mean};
- **Spatial resolution:** maximum number of degrees of freedom (DoFs$_{\text{max}}$) reached during the run;
- **Cost–accuracy trade-off:** total CPU time and final-time error $\|e(T)\|_{L^2}$.

11.1.4 Accuracy–cost trade-off

The tables highlight a clear separation between what drives accuracy and what drives cost.

Spatial adaptivity is the dominant accuracy lever. In both experiments, enabling spatial AMR reduces the final-time error by roughly one order of magnitude: at $T = 0.5$, $\|e(T)\|_{L^2}$ decreases from 3.96×10^{-5} to 2.42×10^{-6} (about 16× smaller); at $T = 0.1$, it decreases from $\approx 3.9 \times 10^{-5}$ to 2.0×10^{-6} (about 20× smaller). This improvement is obtained by a substantial mesh growth, with DoFs increasing from 169 (fixed mesh) to $\mathcal{O}(3 \times 10^4)$ in the adaptive runs.

Time adaptivity alone increases cost without improving the final error. With fixed space, the adaptive time controller drastically increases the number of time steps while leaving $\|e(T)\|_{L^2}$ essentially unchanged. At $T = 0.5$, the run performs 3666 steps with Δt_{mean} =

Table 1: Summary at $T = 0.5$ (key quantities from `summary_comparison.csv`).

Configuration	steps	Δt_{mean}	DoFs$_{\text{max}}$	CPU [s]	$\|e(T)\|_{L^2}$
fixed_space_fixed_time	250	2.0×10^{-3}	169	2.80	3.96×10^{-5}
adaptive_space_fixed_time	250	2.0×10^{-3}	30237	339	2.42×10^{-6}
fixed_space_adaptive_time	3666	1.36×10^{-4}	169	49.2	3.84×10^{-5}
adaptive_space_adaptive_time	1132	4.42×10^{-4}	31562	2466	2.55×10^{-6}

Table 2: Summary at $T = 0.1$ (key quantities from `summary_comparison.csv`).

Configuration	steps	Δt_{mean}	DoFs$_{\text{max}}$	CPU [s]	$\|e(T)\|_{L^2}$
fixed_space_fixed_time	50	2.0×10^{-3}	169	0.603	3.9×10^{-5}
adaptive_space_fixed_time	50	2.0×10^{-3}	28790	68.46	2.0×10^{-6}
fixed_space_adaptive_time	730	1.37×10^{-4}	169	11.62	3.8×10^{-5}
adaptive_space_adaptive_time	231	4.33×10^{-4}	29479	480.25	3.0×10^{-6}

1.36×10^{-4} (baseline is 2×10^{-3}), reaching an error $3.84 \times 10^{-5} \approx 3.96 \times 10^{-5}$ but costing 49.2 s. At $T = 0.1$, the same pattern appears (730 steps, $\Delta t_{\text{mean}} = 1.37 \times 10^{-4}$, error $\approx 3.8 \times 10^{-5}$, CPU 11.62 s). In this setup, shrinking Δt does not reduce the dominant error contribution, so the extra work is not translated into higher accuracy.

Combined adaptivity yields the worst trade-off. When both AMR and time adaptivity are enabled, the cost increases dramatically without a measurable improvement over spatial AMR alone. At $T = 0.5$, the combined run takes 2466 s (vs 339 s with spatial-only) while achieving 2.55×10^{-6} (comparable to 2.42×10^{-6}). At $T = 0.1$, it takes 480.25 s (vs 68.46 s spatial-only) with 3.0×10^{-6} (same order of magnitude as the spatial-only result). Therefore, under the chosen heuristic time-step controller and tolerances, time adaptivity adds a large overhead on top of AMR without providing additional accuracy.
Table 3 summarizes the performance of the four configurations.

11.2. Test 2

In this second test case, we evaluate the impact of coupling Spatial Adaptivity (AMR) with Temporal Adaptivity based on the *step-doubling* technique. Unlike the heuristic approach used in simple tests, step-doubling offers a more rigorous error control by computing the solution at t_{n+1} twice (one full step vs. two half-steps) and comparing the results.
However, combining AMR with step-doubling introduces a specific numerical challenge: the interpolation error generated when projecting the solution onto a refined mesh can be misinterpreted by the time integrator as a temporal discretization error. This often leads to excessive time-step reduction (rejecting steps repeatedly), causing a dramatic increase in computational cost. To mitigate this issue and ensure feasibility, the time-stepping parameters were adjusted for the fully adaptive runs: the tolerance was relaxed to 10^{-6} and the minimum allowed time-step was increased (new minTimestep $= 10^{-4}$) to avoid stagnation at the solver's lower limit.
Analysis of Results:

- **Accuracy vs. Cost:** The fully adaptive configuration (*Adaptive Space / Adapt. Time*) achieves the best accuracy overall ($7.12 \cdot 10^{-7}$), which is an order of magnitude better than the spatially adaptive fixed-time case and two orders of magnitude better than the fixed-grid cases. However, this precision comes at an extreme computational cost (1597s vs 75s for the fixed-time adaptive case).
- **Impact of Spatial Adaptivity:** The transition from fixed space to adaptive space yields a significant accuracy improvement. The problem is clearly spatially dom-

Table 3: Summary at $T = 0.5$ (results from the fully adaptive test case with step-doubling, Test 2).

Configuration	Steps	Δt_{mean}	DoFs_{max}	CPU [s]	$\|e(T)\|_{L^2}$
fixed_space_fixed_time	250	2.0×10^{-3}	441	0.57	1.5×10^{-5}
adaptive_space_fixed_time	250	2.0×10^{-3}	80422	75.66	2.3×10^{-6}
fixed_space_adaptive_time	358	1.4×10^{-3}	441	1.16	1.4×10^{-5}
adaptive_space_adaptive_time	3055	1.6×10^{-4}	86608	1597.10	7.1×10^{-7}

inated, as seen by the huge jump in DoFs (from 441 to over 80k) required to capture the solution features correctly.

- **Overhead of Step Doubling:** The step-doubling method proves to be particularly expensive when coupled with AMR. The solver performed over 3000 time steps (compared to 250 in the fixed case) to satisfy the error tolerances, often battling against the interpolation noise introduced by mesh refinement. This confirms that while step-doubling is rigorous, a heuristic time-stepping strategy might be more efficient for problems with frequent mesh changes.

In conclusion, `adaptive_space_adaptive_time` provides the highest fidelity solution but is computationally demanding. For practical applications where a slightly larger error is acceptable, `adaptive_space_fixed_time` offers a much better trade-off, providing high spatial accuracy at a fraction of the time (75s vs 1600s).

11.3. Test 3

In the previous tests we focused on a single choice of forcing parameters (σ=0.1, N = 5.0, a = 5.0) and compared fixed vs. adaptive strategies in space and time. Here we take a more qualitative point of view and investigate how the convenience of space and time adaptivity depends on the structure of the forcing term, in particular on:

- the spatial width σ of the Gaussian source;
- the temporal activity encoded by the pair (a, N) in $g(t)$.

Rather than reporting another full set of numerical tables, we summarize the behaviour observed in the runs and expected from the theory in terms of four representative regimes.

11.3.1 Qualitative regimes

The following scenarios are considered:

- **Large** σ: the source is spatially smooth and spread over a relatively wide region;
- **Small** σ: the source is sharply localized in space;
- **Large** (a, N): the forcing is stiff in time, with fast oscillations or sharp temporal transitions;
- **Small** (a, N): the forcing varies slowly in time and the dynamics are mildly unsteady.

Table 4 summarises, in a compact way, when spatial AMR and time adaptivity are expected to be most effective.

Table 4: Qualitative effect of space/time adaptivity in different forcing regimes.

Scenario	Space adaptivity	Time adaptivity
σ large	useful	neutral
σ small	difficult	neutral
a, N large	neutral	useful
a, N small	neutral	mildly useful

11.3.2 Discussion

Effect of σ (spatial localization). For **large** σ, the forcing is smooth and well resolved by a moderate number of mesh levels. In this regime the solution error is dominated by spatial resolution, and adaptive mesh refinement is clearly beneficial: it concentrates DoFs around the region influenced by the source, improving the final-time L^2 error at a reasonable cost. Time adaptivity, on the other hand, tends to be neutral: the temporal dynamics are not stiff, so a fixed time step chosen as in Test 1 already resolves the evolution satisfactorily.

For **small** σ, the source becomes highly localized. In this configuration spatial adaptivity is *difficult*: the estimator tends to trigger frequent refinement/derefinement cycles around a narrow spike, increasing mesh complexity and interpolation noise without a proportional reduction of the global error. Time adaptivity remains essentially neutral, because the dominant difficulty is spatial rather than temporal.

Effect of (a, N) (temporal stiffness). When (a, N) **are large**, the forcing term exhibits strong temporal variations: rapid oscillations or sharp pulses. In this case the problem becomes stiff in time and adaptive time stepping is genuinely useful: the controller can take small steps only when needed (around peaks of $g(t)$) and larger steps elsewhere, leading to a better cost–accuracy balance than a uniformly small Δt. The spatial discretisation, however, does not change qualitatively with (a, N), so AMR is mostly neutral compared to the σ-driven effects seen above.
When (a, N) **are small**, the forcing is slowly varying in time. Here a fixed time step is already adequate, and time adaptivity provides at most a mild benefit: the controller still reacts to small changes in $g(t)$, but the resulting step-size modulation does not significantly improve the final error and may even add some overhead. Spatial adaptivity remains neutral with respect to (a, N) and is instead governed by the value of σ.

11.3.3 Summary

Test 3 can be read as a guide for selecting adaptive strategies:

- for *spatially wide* sources (large σ as 0.5), spatial AMR is the main accuracy lever, while time adaptivity is optional;
- for *sharply localized* sources (small σ as 0.1), spatial adaptivity becomes fragile and may be overly expensive, regardless of the time-stepping strategy;
- for *temporally stiff* problems (large (a, N) as (5,5)), time adaptivity is justified and can notably reduce the cost compared to a uniformly small time step;
- for *mildly unsteady* problems (small (a, N) as (2,2)), time adaptivity offers only limited gains.

These regimes complement the quantitative evidence from Tests 1 and 2 and highlight that the effectiveness of adaptive strategies is strongly problem-dependent: space adaptivity is driven by σ, while time adaptivity is driven by (a, N).

11.4. Final discussion

Integrating the results from both testing campaigns one and two, the main conclusions are:

- **Spatial Dominance:** Spatial adaptivity (AMR) provides the most significant accuracy improvement, consistently reducing the L^2 error by an order of magnitude compared to fixed baselines.
- **Pure Time Adaptivity:** On coarse spatial meshes, time adaptivity alone is ineffective. It increases computational cost without reducing the final error, which remains saturated by spatial discretization.
- **The High Cost of Precision:** Unlike simpler heuristics, the rigorous *step-doubling* coupled with AMR achieves the absolute best accuracy ($7.1 \cdot 10^{-7}$). However, this comes at a prohibitive cost ($\approx$ 1600s vs 75s) due to the solver interpreting mesh interpolation noise as temporal error, a known issue in monolithic adaptive solvers [3].

Overall, `adaptive_space_fixed_time` represents the best engineering compromise, leveraging AMR for accuracy while maintaining efficiency. The fully adaptive configuration is recommended only when maximum theoretical precision is required, regardless of CPU time.

12. Conclusions

This study analyzed adaptive strategies for the unsteady heat equation with a moving source. The numerical evidence confirms that the problem is strongly dominated by spatial discretization error, a behavior consistent with theoretical error estimates for parabolic problems [6]. Consequently, **Spatial Adaptivity (AMR)** proved essential, reducing the error by over an order of magnitude compared to fixed grids.
While coupling AMR with rigorous time-step adaptivity based on Step-Doubling [4] yields the highest fidelity solution ($e \approx 10^{-7}$), it introduces a severe computational overhead due to the strict control of interpolation errors. Therefore, we recommend the **spatially adaptive scheme with fixed time steps** as the optimal strategy, offering a 20× speed-up compared to the fully adaptive case with only a minor com-

promise in accuracy.

13. Acknowledgements

References

[1] deal.II tutorial: Step-26. `https://dealii.org/current/doxygen/deal.II/step_26.html`.

[2] Daniel Arndt, Wolfgang Bangerth, Bruno Blais, Marc Fehling, et al. The `deal.II` library, version 9.5. *Journal of Numerical Mathematics*, 31(3):231–246, 2023.

[3] A. Belme, A. Dervieux, and F. Alauzet. Interactions between adaptive time-integrators and adaptive meshing in a monolithic fem solver. *International Journal of Numerical Methods for Heat & Fluid Flow*, 29(7):2297–2324, 2019.

[4] Ernst Hairer, Syvert P Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*, volume 8 of *Springer Series in Computational Mathematics*. Springer, 2nd edition, 1993. See Section II.4 for Richardson Extrapolation and step-size control.

[5] Marco Picasso. Adaptive finite elements for a linear parabolic problem. *Computer Methods in Applied Mechanics and Engineering*, 167(3-4):223–237, 1998.

[6] Vidar Thomée. *Galerkin Finite Element Methods for Parabolic Problems*, volume 25 of *Springer Series in Computational Mathematics*. Springer, 2nd edition, 2006.