POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Numerical Solvers for the Second-Order Wave Equation

Numerical Methods for Partial Differential Equations
MSc in High Performance Computing Engineering
Academic Year: 2025-2026

Roberto Di Lauro, 10869236
Thomas Fabbris, 11174296
Alessandro Ghiotto, 11177200
Cristian Rubbi, 11177799

Abstract: This project presents the development, implementation, and performance analysis of a numerical solver for the second-order wave equation. The solver is built upon the **deal.II** finite element library and utilizes **Trilinos** wrappers for distributed linear algebra and parallel computing. The computational domain is discretized using simplicial meshes generated internally, supporting various polynomial degrees r for the finite element space.

We investigate two main families of time-integration schemes: the **θ-methods** (e.g., Forward Euler, Crank-Nicolson, and Backward Euler) and the **Newmark-beta** family. A rigorous numerical analysis is performed, focusing on:

1. **Convergence Analysis**: Evaluation of relative L^2 and H^1 errors across multiple spatial and temporal resolutions to verify theoretical accuracy rates.
2. **Dissipation and Dispersion**: Study of energy conservation properties and phase errors introduced by the discrete approximation.
3. **Scalability**: Performance benchmarks conducted on the **Politecnico di Milano HPC cluster**, evaluating parallel speedup and efficiency.

The results provide a comprehensive characterization of the solver's reliability and efficiency, demonstrating its suitability for large-scale dynamic wave propagation simulations.

1. Introduction

The wave equation is a fundamental mathematical model used to describe the propagation of disturbances in various physical media, spanning from acoustics and electromagnetism to seismology and structural dynamics [3]. In its second-order form, this hyperbolic partial differential equation captures the essential physics of wave motion by balancing temporal acceleration with spatial curvature. Due to its wide range of applications, developing accurate and efficient numerical solvers is crucial for predicting complex dynamic behaviors.

The numerical simulation of wave phenomena, however, introduces significant challenges that go beyond simple stability. While the Courant-Friedrichs-Lewy (CFL) condition [2] provides a standard limit for time-stepping in explicit schemes, the long-term accuracy of the solution is often compromised by numerical dispersion and dissipation [4]. Numerical dispersion leads to frequency-dependent velocity errors that distort the wave shape, while numerical dissipation causes an unphysical decay of the wave's amplitude. Mitigating these errors demands a careful balance between spatial resolution and the choice of the time-integration strategy.

Motivated by these challenges, this work conducts a systematic evaluation of established time-marching strategies [5, 6] applied to the semi-discrete wave equation. The primary objective is to move beyond theoretical stability bounds and provide a clear characterization of the practical trade-offs between computational efficiency and physical fidelity in large-scale scenarios. By examining how different algorithmic choices impact energy conservation, phase accuracy, and parallel execution times, this study aims to establish robust guidelines for dynamic wave simulations.

2. Continuous Wave Problem

2.1. Strong Formulation

The propagation of waves in a bounded domain $\Omega \subset \mathbb{R}^2$, as preannounced, is modeled by a second-order hyperbolic Partial Differential Equation [7]. We seek the displacement field $u(\mathbf{x}, t)$ that satisfies the following initial-boundary value problem:

$$\begin{cases} \frac{\partial^2 u}{\partial t^2}(\mathbf{x}, t) - \nabla \cdot (c^2 \nabla u(\mathbf{x}, t)) = f(\mathbf{x}, t) & \text{in } \Omega \times (0, T] \\ u(\mathbf{x}, t) = \phi(\mathbf{x}, t) & \text{on } \partial\Omega \times [0, T] \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}) & \text{in } \Omega \\ \frac{\partial u}{\partial t}(\mathbf{x}, 0) = v(\mathbf{x}) & \text{in } \Omega \end{cases} \tag{1}$$

In this framework, $c(\mathbf{x}, t)$ denotes the wave propagation speed, which is handled in our implementation as a space-time dependent function. The term f represents the external forcing, while u_0 and v define the initial displacement and velocity, respectively. The Dirichlet boundary condition ϕ constrains the solution on the boundary $\partial\Omega$ throughout the simulation interval $[0, T]$.

2.2. Weak Formulation

To apply the Finite Element Method (FEM), we derive the variational form of the strong problem. We begin by defining the appropriate functional spaces for the solution and the test functions. The test space is defined as $V = H_0^1(\Omega) = \{v \in H^1(\Omega) : v|_{\partial\Omega} = 0\}$, which consists of functions with zero trace on the boundary. In the presence of non-homogeneous Dirichlet conditions $\phi \neq 0$, the solution u belongs to the trial space $V\phi = \{v \in H^1(\Omega) : v|_{\partial\Omega} = \phi\}$.

To ensure compatibility with the Lax-Milgram Lemma [6], which requires trial and test functions to belong to the same Hilbert space, we introduce a **Lifting Operator** $\mathcal{R}$. Let $u_\phi = \mathcal{R}g \in V_\phi(\Omega)$ be a

function such that its trace on the boundary $\partial\Omega$ is exactly ϕ. We decompose the global solution as:

$$u(\mathbf{x},t) = u_0(\mathbf{x},t) + u_\phi(\mathbf{x},t) \tag{2}$$

where $u_0 \in V$ is the new unknown function with zero trace on the boundary.
The derivation proceeds by multiplying the strong equation by a test function $v \in V$ and integrating over the domain Ω:

$$\int_\Omega \frac{\partial^2 u}{\partial t^2} v \, d\Omega - \int_\Omega \nabla \cdot (c^2 \nabla u) v \, d\Omega = \int_\Omega f v \, d\Omega \quad \forall v \in V \tag{3}$$

Applying Green's first identity to the second-order spatial term, we obtain:

$$\int_\Omega \frac{\partial^2 u}{\partial t^2} v \, d\Omega + \int_\Omega c^2 \nabla u \cdot \nabla v \, d\Omega - \int_{\partial\Omega} (c^2 \nabla u \cdot \mathbf{n}) v \, d\gamma = \int_\Omega f v \, d\Omega \quad \forall v \in V \tag{4}$$

Since $v = 0$ on $\partial\Omega$, the boundary integral vanishes. Substituting the decomposition $u = u_0 + u_\phi$ and exploiting the linearity of the operators, the semi-discrete weak formulation is expressed in the following way:

$$\begin{cases} \forall t \in (0,T), \text{ find } u_0(t) \in V \text{ such that:} \\ \left(\frac{\partial^2 u_0}{\partial t^2}, v\right)_{L^2(\Omega)} + a(u_0, v) = G(v) \quad \forall v \in V \\ u(x,0) = u_0(x) \\ \frac{\partial u}{\partial t}(x,0) = u_1(x) \end{cases} \tag{5}$$

The bilinear and linear forms utilized in the implementation are defined as:

- **Mass form**: $(w,v)_{L^2(\Omega)} = \int_\Omega wv \, d\Omega$
- **Stiffness form**: Let $a : V \times V \to \mathbb{R}$, with $a(u,v) = \int_\Omega c^2 \nabla u \cdot \nabla v \, d\Omega$
- **RHS**: Let $G : V \to \mathbb{R}$, with $G(v) = \int_\Omega f v \, d\Omega - \left(\frac{\partial^2 u_\phi}{\partial t^2}, v\right)_{L^2(\Omega)} - a(u_\phi, v)$, which includes the entire contribution which doesn't depends on u.

This formulation eliminates the second derivatives in space, making the problem compatible with finite H^1 spaces and allowing numerical discretization. Thanks to the lifting operator, we get homogeneous Dirichlet BC, so the bilinear form $a(\cdot, \cdot)$ remains continuous and coercive on the Hilbert space $H_0^1(\Omega)$, guaranteeing the existence and uniqueness of the solution [3, 6].

3. Spatial Discretization

To discretize the problem in space using the standard Galerkin method, we introduce a finite-dimensional subspace $V_h \subset H_0^1(\Omega)$, where $\dim(V_h) = N_h < +\infty$. In our implementation, this space is constructed using simplicial finite elements of polynomial degree r, specifically through the `FE_SimplexP` class provided by the **deal.II** library [1]. The goal is to find an approximation $u_h(t) \in V_h$ of the exact solution by imposing the weak formulation within the discrete space, leading to the following numerical problem:

$$\begin{cases} \forall t \in (0,T), \text{ find } u_h(t) \in V_h \text{ such that:} \\ \left(\frac{\partial^2 u_h}{\partial t^2}, v_h\right)_{L^2(\Omega)} + a(u_h, v_h) = G(v_h) \quad \forall v_h \in V_h \end{cases} \tag{6}$$

To construct an algebraic representation, we consider a basis of V_h denoted by the shape functions $\{\varphi_j\}_{j=1}^{N_h}$. The discrete solution is expanded as:

$$u_h(\mathbf{x},t) = \sum_{j=1}^{N_h} U_j(t) \varphi_j(\mathbf{x}) \tag{7}$$

where the coefficients $U_j(t)$ represent the degrees of freedom (DoFs) of the system, collected in the vector $\mathbf{U}(t) = [U_1(t), U_2(t), \ldots, U_{N_h}(t)]^T$. Substituting this expansion into the weak formulation and choosing $v_h = \varphi_i$ as test functions, the linearity of the integral yields:

$$\sum_{j=1}^{N_h} \frac{d^2U_j(t)}{dt^2} \int_\Omega \varphi_j \varphi_i \, d\Omega + \sum_{j=1}^{N_h} U_j(t) \int_\Omega c^2 \nabla\varphi_j \cdot \nabla\varphi_i \, d\Omega = \mathbf{G}_i(t) \quad i = 1, ..., N_h \tag{8}$$

This allows us to identify the global matrices assembled by the solver:

- **Mass Matrix M**: $M_{ij} = \int_\Omega \varphi_j \varphi_i \, d\Omega$.
- **Stiffness Matrix A**: $A_{ij} = \int_\Omega c^2 \nabla\varphi_j \cdot \nabla\varphi_i \, d\Omega$.
- **RHS G**$(t) = [G_1(t), G_2(t), \ldots, G_{N_h}(t)]^T$.

The resulting semi-discrete dynamical system is:

$$\mathbf{M} \frac{\partial^2 \mathbf{U}}{\partial t^2}(t) + \mathbf{A}\mathbf{U}(t) = \mathbf{G}(t) \tag{9}$$

This second-order system of Ordinary Differential Equations represents the spatial discretization (semi-discrete) of the wave equation.

4. Time Discretization

The spatial discretization detailed in (9) yields a semi-discrete dynamical system subject to the initial conditions $\mathbf{U}(0) = \mathbf{U}_0$ and $\frac{\partial \mathbf{U}}{\partial t}(0) = \mathbf{V}_0$.
To apply standard time-marching techniques, it is common practice to recast (9) as a system of first-order ordinary differential equations [6]. We introduce the discrete velocity vector $\mathbf{V}(t) = \frac{\partial \mathbf{U}}{\partial t}(t)$, allowing us to rewrite the problem as two coupled equations:

$$\begin{cases} \frac{\partial \mathbf{U}}{\partial t}(t) - \mathbf{V}(t) = \mathbf{0} \\ \mathbf{M}\frac{\partial \mathbf{V}}{\partial t}(t) + \mathbf{A}\mathbf{U}(t) = \mathbf{G}(t) \end{cases} \tag{10}$$

Boundary conditions are naturally inherited by the velocity field; for a prescribed Dirichlet datum $\mathbf{U}(t) = \phi(t)$ on the boundary nodes, the corresponding velocity constraint is strictly enforced as $\mathbf{V}(t) = \frac{\partial \phi}{\partial t}(t)$. This coupled first-order system forms the basis for the θ-method family, while the original second-order form (9) is utilized directly by the Newmark-β integration scheme.

4.1. The θ-Method

To discretize the first-order system (10), we adopt a family of implicit-explicit schemes parameterized by $\theta \in [0, 1]$ [6]. Time derivatives are approximated using finite differences over a time step Δt, and the equation terms are evaluated as a convex combination of levels t^n and t^{n+1}. The discretized system reads:

$$\frac{\mathbf{U}^{n+1} - \mathbf{U}^n}{\Delta t} = \theta \mathbf{V}^{n+1} + (1-\theta)\mathbf{V}^n \tag{11}$$

$$\mathbf{M}\frac{\mathbf{V}^{n+1} - \mathbf{V}^n}{\Delta t} + \mathbf{A}(\theta \mathbf{U}^{n+1} + (1-\theta)\mathbf{U}^n) = \mathbf{G}_\theta^{n+1} \tag{12}$$

where $\mathbf{G}_\theta^{n+1} = \theta \mathbf{G}^{n+1} + (1-\theta)\mathbf{G}^n$.
By isolating $\mathbf{V}^{n+1}$ in (11) and substituting it into (12), we obtain a single linear system for the unknown displacement $\mathbf{U}^{n+1}$:

$$(\mathbf{M} + \theta^2 \Delta t^2 \mathbf{A})\mathbf{U}^{n+1} = \mathbf{M}(\mathbf{U}^n + \Delta t \mathbf{V}^n) - \theta(1-\theta)\Delta t^2 \mathbf{A}\mathbf{U}^n + \theta \Delta t^2 \mathbf{G}_\theta^{n+1} \tag{13}$$

At each time step, we solve (13) for $\mathbf{U}^{n+1}$. Subsequently, the velocity $\mathbf{V}^{n+1}$ is explicitly updated using (11). The parameter θ dictates the properties of the scheme:

- $\theta = 0$: **Forward Euler**. Fully explicit and conditionally stable.
- $\theta = 1/2$: **Crank-Nicolson**. Implicit, second-order accurate in time, and non-dissipative.
- $\theta = 1$: **Backward Euler**. Fully implicit, generally unconditionally stable, but highly dissipative.

4.2. Newmark Time Integration

The Newmark-β method is a widely used time-marching scheme for second-order dynamical systems [4, 5]. It computes the state at $t^{n+1} = t^n + \Delta t$ by introducing two parameters, β and γ, which dictate the stability, accuracy, and numerical dissipation.
The discrete displacement $\mathbf{U}$ and velocity $\mathbf{V}$ are updated using the acceleration $\mathbf{a} \approx \frac{\partial^2 \mathbf{U}}{\partial t^2}$:

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t \mathbf{V}^n + \Delta t^2 \left[\left(\frac{1}{2} - \beta \right) \mathbf{a}^n + \beta \mathbf{a}^{n+1} \right] \tag{14}$$

$$\mathbf{V}^{n+1} = \mathbf{V}^n + \Delta t \left[(1-\gamma)\mathbf{a}^n + \gamma \mathbf{a}^{n+1} \right] \tag{15}$$

To compute the unknown acceleration $\mathbf{a}^{n+1}$, we enforce dynamic equilibrium at t^{n+1}:

$$\mathbf{M}\mathbf{a}^{n+1} + \mathbf{A}\mathbf{U}^{n+1} = \mathbf{G}^{n+1} \tag{16}$$

Substituting (14) into this equilibrium equation yields a linear system for $\mathbf{a}^{n+1}$:

$$(\mathbf{M} + \beta \Delta t^2 \mathbf{A})\mathbf{a}^{n+1} = \mathbf{G}^{n+1} - \mathbf{A}\mathbf{U}^n - \Delta t \mathbf{A}\mathbf{V}^n - \Delta t^2 \left(\frac{1}{2} - \beta \right) \mathbf{A}\mathbf{a}^n \tag{17}$$

At each time step, we solve (17) for $\mathbf{a}^{n+1}$, and then explicitly update $\mathbf{U}^{n+1}$ and $\mathbf{V}^{n+1}$. The method's behavior depends heavily on the chosen parameters:

- $\gamma = 1/2, \beta = 1/4$: **Average Acceleration Method**. Unconditionally stable and non-dissipative.
- $\beta > 1/4$: Introduces numerical dissipation, useful for damping high-frequency modes.
- $\gamma < 1/2$: Explicit scheme (conditionally stable).

5. Implementation Details

This section describes the software architecture and organization of the numerical solver for the wave equation. The implementation is written in C++17 and relies on the **deal.II** finite element library [1] for spatial discretization, combined with the **Trilinos** framework for distributed sparse linear algebra. The code supports both sequential and parallel execution through MPI.

5.1. Overall Architecture

The software follows an object-oriented design built around three high-level components:

- A **wave equation base class** that encapsulates the spatial discretization, parallel mesh management, matrix assembly, and all simulation input/output operations.
- Two **concrete solver classes**, one for each time-integration family, that inherit the common infrastructure and implement the scheme-specific time-stepping logic.
- A **parameter reader** that parses external configuration files, enabling full runtime control over the problem definition and discretization parameters without recompilation.

This separation of concerns allows the spatial discretization and I/O infrastructure to be defined once and shared by all time-integration methods. We have two executables for the time discretization methods, and modifying the problem data requires only a different JSON parameter file, with no changes to the compiled codes.

5.1.1 Wave Equation Base Class

The abstract class `WaveEquationBase` provides the functionalities shared by both solvers. It includes:

- **Mesh generation and distribution.** A simplicial (triangular) mesh is created on a rectangular domain Ω via `subdivided_hyper_rectangle_with_simplices`, partitioned across MPI ranks with `GridTools::partition_triangulation`, and stored in a object of type `parallel::fullydistributed::Triangulation`.
- **Finite element space setup.** The Lagrangian simplex element `FE_SimplexP` of polynomial degree r is instantiated together with a matching `QGaussSimplex` quadrature rule of order $r+1$.
- **Degree-of-freedom management.** Degrees of freedom are distributed on the partitioned mesh via the `DoFHandler`, and the associated distributed sparsity pattern is built for the Trilinos sparse matrices.
- **Global matrices.** The mass matrix $\mathbf{M}$ and the stiffness matrix $\mathbf{A}$ (weighted by the wave speed c^2) are assembled once at startup, since both are time-independent. Assembly is performed in parallel, each MPI process computes local element contributions and inserts them into `TrilinosWrappers::SparseMatrix` objects, which handle the distributed storage and communication.
- **Diagnostics and logging.** At user-specified intervals the base class computes the discrete energy $E^n = \frac{1}{2}(\mathbf{V}^T\mathbf{M}\,\mathbf{V} + \mathbf{U}^T\mathbf{A}\,\mathbf{U})$, evaluates L^2 and H^1 errors against an exact solution (when available), records a point probe at the domain centre, and logs CG iteration counts. All diagnostic data are written to CSV files for post-processing.
- **Solution output.** Displacement and velocity fields are exported in VTU/PVTU format at configurable intervals, enabling visualization in ParaView.
- **Divergence detection.** The L^2 norms of the solution vectors are monitored at each time step; if they exceed a threshold the simulation is terminated early with a diagnostic message.

The class exposes a single pure-virtual method `run()` that derived classes must override to implement the time-stepping loop.

5.1.2 Theta-Method Solver

The class `WaveTheta` inherits from `WaveEquationBase` and implements the θ-method described in Section 4.1. At each time step it solves *two* SPD systems sequentially, one for the displacement $\mathbf{U}^{n+1}$

and one for the velocity $\mathbf{V}^{n+1}$, enforcing Dirichlet conditions $\phi(t^{n+1})$ and $\partial_t\phi(t^{n+1})$ respectively. Both solves use Conjugate Gradient with an AMG preconditioner.

5.1.3 Newmark Solver

The class `WaveNewmark` inherits from `WaveEquationBase` and implements the Newmark-β family described in Section 4.2. It introduces the discrete acceleration $\mathbf{a}^n$ as an additional unknown, a consistent $\mathbf{a}^0$ is computed at startup by solving $\mathbf{M}\,\mathbf{a}^0 = \mathbf{G}(0) - \mathbf{A}\,\mathbf{U}^0$. At each subsequent step only *one* SPD system is solved for $\mathbf{a}^{n+1}$, after which $\mathbf{U}^{n+1}$ and $\mathbf{V}^{n+1}$ are updated algebraically. The same CG + AMG solver strategy is used.

5.1.4 Parameter Reader

The class `ParameterReader` wraps deal.II's `ParameterHandler` and manages the parsing of external JSON configuration files. A parameter file specifies:

- **Domain and discretization**: geometry bounds (e.g. $[0,1]^2$), number of mesh elements per direction, and polynomial degree r.
- **Time integration**: final time T, time-step size Δt, and method-specific parameters (θ for the theta-method; γ and β for Newmark).
- **Problem data**: symbolic mathematical expressions for $c(\mathbf{x})$, $f(\mathbf{x},t)$, $u_0(\mathbf{x})$, $v_0(\mathbf{x})$, $\phi(\mathbf{x},t)$, $\partial_t\phi(\mathbf{x},t)$, and optionally the exact solution.
- **Output control**: flags for saving VTU snapshots, enabling CSV logging, and specifying the logging and printing frequency.

The function expressions are evaluated at runtime by `deal.II`'s `FunctionParser`, which supports standard mathematical operators and constants (e.g. `pi`). This design enables the definition of arbitrary test cases without recompiling the solver.

5.2. Execution Flow

Each simulation is launched via a dedicated executable (`main-theta` or `main-newmark`) with the path to a JSON parameter file. After MPI initialization, the `ParameterReader` parses the file and populates all `FunctionParser` objects. A solver instance is then constructed and its `run()` method is invoked, which creates the distributed mesh, sets up the FE space, assembles the global matrices, and enters the time-stepping loop from $t = 0$ to $t = T$. At configurable intervals, energy, errors, and solution snapshots are logged to CSV and VTU files. Upon completion a summary row is appended to a convergence CSV for cross-run comparison.

5.3. Parametric Studies and HPC Execution

The numerical experiments presented in this report require hundreds of solver runs with varying discretization parameters and schemes. To automate this process, three Python driver scripts repeatedly invoke the C++ executables with different parameter combinations and collect the resulting CSV data:

- **Convergence sweep** (`convergence_sweep.py`): iterates over combinations of scheme, mesh size N_{el}, polynomial degree r, and time step Δt. For conditionally stable (explicit) methods, runs that violate the CFL condition are automatically filtered out.
- **Dissipation and dispersion sweep** (`dissipation_dispersion_sweep.py`): fixes the spatial mesh and sweeps over Δt for each scheme, logging per-step energy, errors, and a point-probe time series to characterize numerical dissipation and phase shift.
- **Scalability sweep** (`scalability_sweep.py`): fixes the discretization and measures the wall-clock time for an increasing number of MPI processes, producing the data for strong-scaling analysis.

All production runs were executed on the Politecnico di Milano HPC "Students Cluster" through PBS job scripts that copy the project to the node *scratch_local* storage for fast I/O, bind MPI ranks to physical cores, and copy the resulting CSV files back upon completion.

6. Convergence Analysis

Under the standard *a priori* error estimate for finite-element semi-discretizations of the wave equation, the total error at time $t_n = n\Delta t$ is bounded by

$$\|u(\cdot, t_n) - u_h^n\| \;\le\; C(h^s + \Delta t^q), \tag{18}$$

where s denotes the spatial convergence order ($r{+}1$ in the L^2 norm, r in the H^1 norm for polynomial degree r) and q is the temporal convergence order of the time-stepping scheme. The purpose of this section is to verify (18) numerically by performing a systematic convergence study on a test problem with a known analytical solution. The Python script `convergence_sweep.py` automates a grid search over all combinations of scheme, mesh size, polynomial degree, and time step, collecting the final relative errors produced by each run.

6.1. Test Problem

We consider a standing-mode wave on the unit square $\Omega = [0,1]^2$ with homogeneous Dirichlet boundary conditions:

$$\begin{cases} \partial_{tt} u - \Delta u = 0 & \text{in } \Omega \times (0,T], \\ u = 0 & \text{on } \partial\Omega \times [0,T], \\ u(\mathbf{x}, 0) = \sin(\pi x)\sin(\pi y), \\ \partial_t u(\mathbf{x}, 0) = 0, \end{cases} \tag{19}$$

whose exact solution is $u_{\text{exact}}(\mathbf{x}, t) = \cos\!\left(\sqrt{2}\,\pi\, t\right)\, \sin(\pi x)\sin(\pi y)$. The final time is $T = 1$.

6.2. Sweep Parameters

The convergence sweep explores the following parameter grid:

Parameter	**Values**	
Mesh elements per side N_{el}	10, 20, 40, 80, 160, 320	
FE polynomial degree r	1, 2	
Time step Δt	10^{-1}, $5{\times}10^{-2}$, …, 10^{-4}	(10 values)

Five schemes are tested:

Scheme	**Parameters**	**Temporal order**	**Stability**
θ-method (FE)	$\theta = 0$	1	Conditional
θ-method (CN)	$\theta = 0.5$	2	Unconditional
θ-method (BE)	$\theta = 1$	1	Unconditional
Newmark (explicit)	$\beta = 0,\ \gamma = \frac{1}{2}$	2	Conditional
Newmark (implicit)	$\beta = \frac{1}{4},\ \gamma = \frac{1}{2}$	2	Unconditional

For the two conditionally stable schemes (Forward Euler and explicit Newmark), the sweep script automatically filters out $(N_{\text{el}}, \Delta t)$ combinations that violate the CFL condition, using a safety factor of 0.9. Runs that produce relative errors above 10^5 are flagged as diverged and excluded from the analysis. The total sweep comprises 490 solver runs.

6.3. Temporal Convergence

To isolate the temporal error, we plot the relative L^2 and H^1 errors at $t = T$ as a function of Δt for fixed mesh sizes h. As the mesh is refined, the spatial error decreases and the temporal convergence rate becomes visible.

The results from Figures 1 to 5 confirm the expected asymptotic behavior:

- **Forward Euler** ($\theta = 0$) and **Backward Euler** ($\theta = 1$) exhibit first-order temporal convergence, $\mathcal{O}(\Delta t)$, in both the L^2 and H^1 norms.
- **Crank–Nicolson** ($\theta = 0.5$) exhibits second-order temporal convergence, $\mathcal{O}(\Delta t^2)$.
- Both **Newmark** variants ($\beta = 0$ and $\beta = 0.25$, with $\gamma = 0.5$) exhibit second-order temporal convergence, $\mathcal{O}(\Delta t^2)$.

For coarse meshes the error is dominated by the spatial component, and the curves plateau regardless of how small Δt is taken. Only when h is sufficiently refined does the temporal slope emerge clearly.

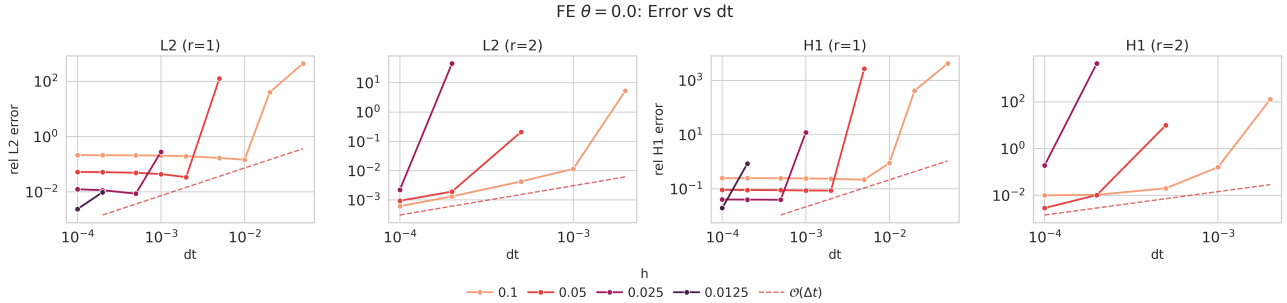

Figure 1: Temporal convergence for the Forward Euler scheme ($\theta = 0$). Each sub-panel shows L^2 or H^1 relative error vs. Δt for $r = 1$ (top) and $r = 2$ (bottom); each curve corresponds to a fixed mesh size h.

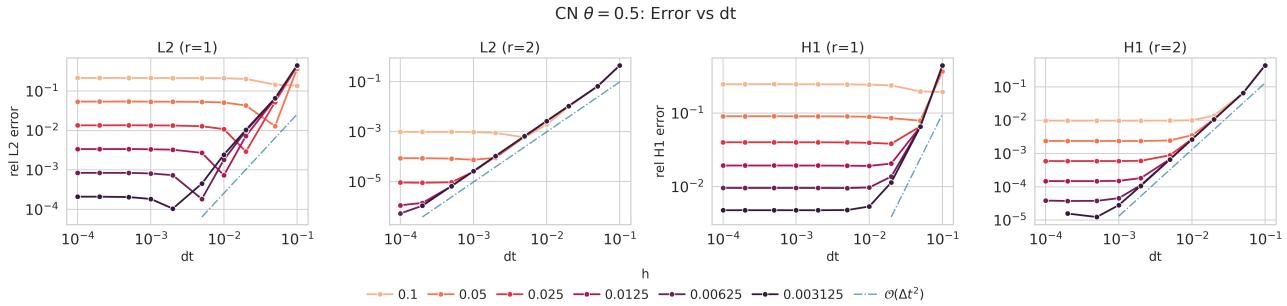

Figure 2: Temporal convergence for the Crank–Nicolson scheme ($\theta = 0.5$). Layout as in Figure 1.

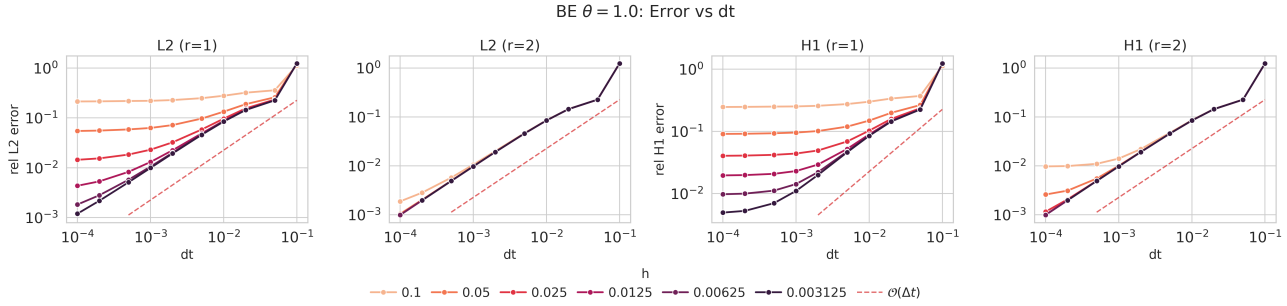

Figure 3: Temporal convergence for the Backward Euler scheme ($\theta = 1$). Layout as in Figure 1.

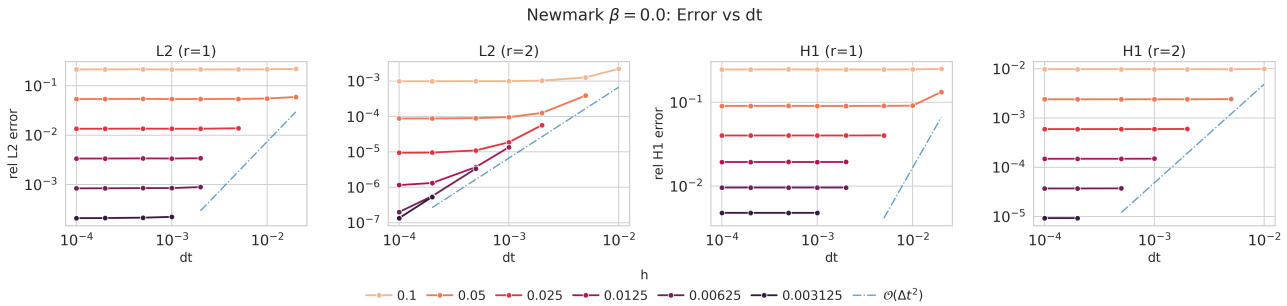

Figure 4: Temporal convergence for explicit Newmark ($\beta = 0$, $\gamma = 0.5$). Layout as in Figure 1.

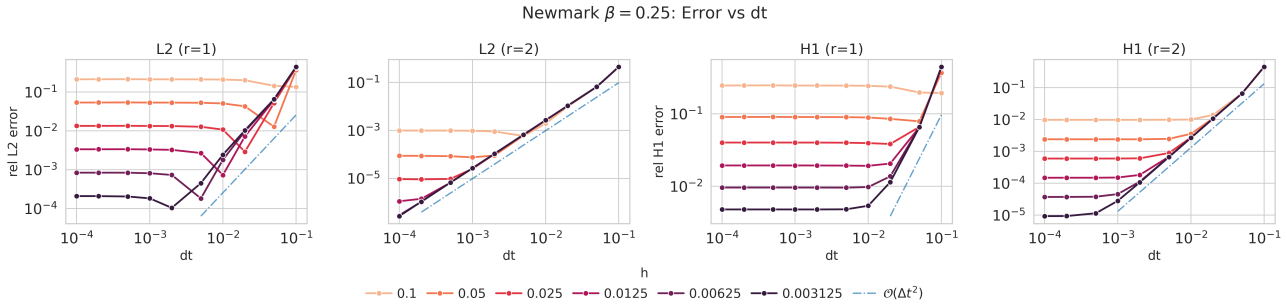

Figure 5: Temporal convergence for implicit Newmark ($\beta = 0.25$, $\gamma = 0.5$). Layout as in Figure 1.

6.4. Spatial Convergence

To isolate the spatial error, we plot the relative errors as a function of the mesh size h for fixed time steps Δt. The expected asymptotic rates for the Galerkin FEM with polynomial degree r are $\mathcal{O}(h^{r+1})$ in the L^2 norm and $\mathcal{O}(h^r)$ in the H^1 norm.
The numerical results obtained in Figures 6 to 10 are in agreement with the theory:

- For $r = 1$: the observed L^2 rate approaches $\mathcal{O}(h^2)$ and the H^1 rate approaches $\mathcal{O}(h)$ as Δt is taken small enough for the temporal error not to dominate.
- For $r = 2$: the observed L^2 rate approaches $\mathcal{O}(h^3)$ and the H^1 rate approaches $\mathcal{O}(h^2)$.

When Δt is too large relative to h, the spatial convergence curves plateau early because the total error is dominated by the temporal component. see Fig. 8 for the L^2 norm with $r = 2$ using BE, where the error is bounded by $C(h^3 + \Delta t)$.
Conversely, for sufficiently small Δt the curves collapse onto one another and the pure spatial rate is visible. See Fig. 7 for the H^1 norm with $r = 1$ using CN, where the error is bounded by $C(h + \Delta t^2)$.

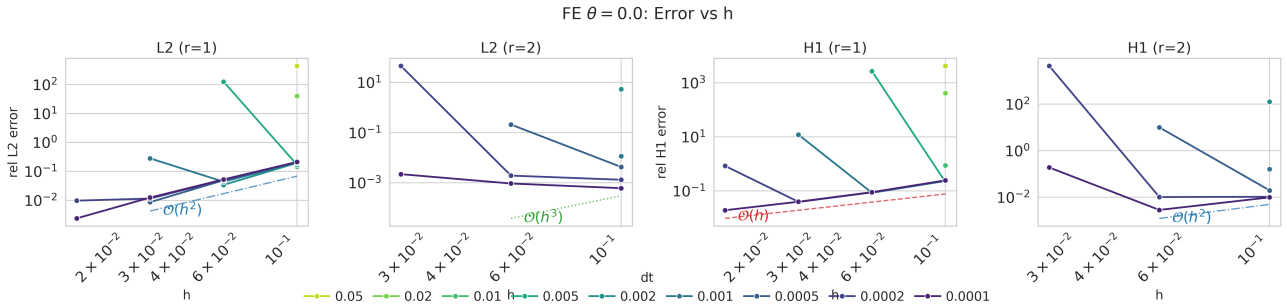

Figure 6: Spatial convergence for the Forward Euler scheme ($\theta = 0$). Each sub-panel shows L^2 or H^1 relative error vs. h for $r = 1$ (top) and $r = 2$ (bottom); each curve corresponds to a fixed time step Δt.

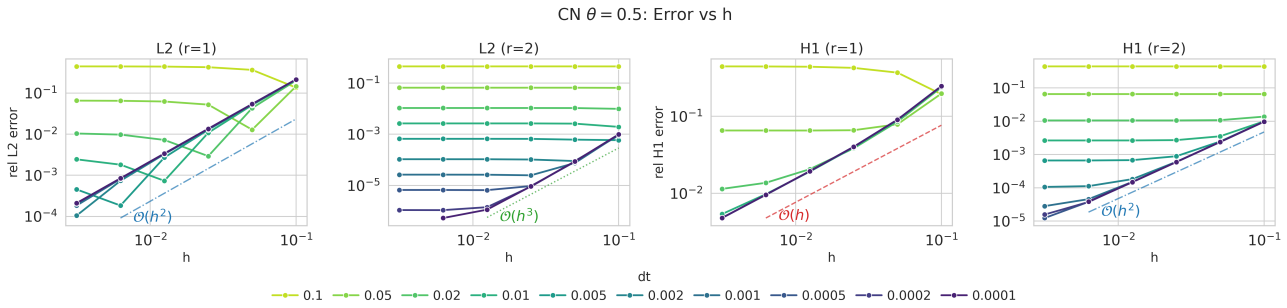

Figure 7: Spatial convergence for the Crank–Nicolson scheme ($\theta = 0.5$). Layout as in Figure 6.

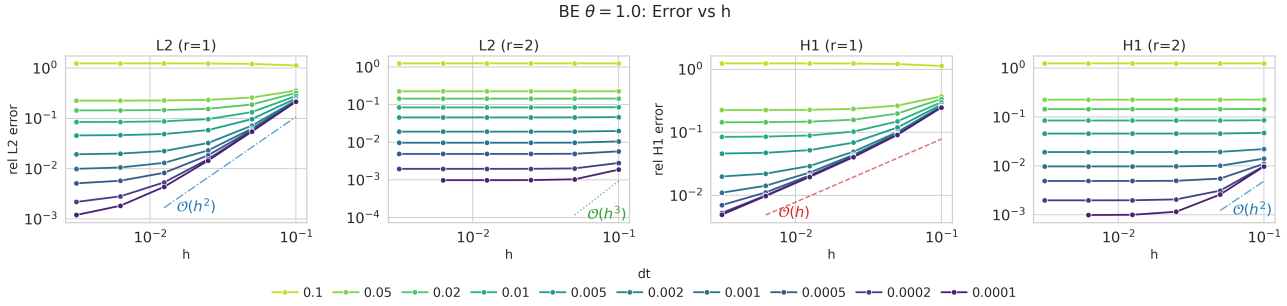

Figure 8: Spatial convergence for the Backward Euler scheme ($\theta = 1$). Layout as in Figure 6.

7. Numerical Dissipation and Dispersion

Even when a time-stepping scheme is stable and convergent, it can still introduce two distinct artefacts that degrade the quality of long-time wave simulations:

- **Numerical dissipation**: unphysical decay of the wave amplitude, manifesting as an energy loss that is absent in the continuous problem.
- **Numerical dispersion**: a frequency-dependent phase-velocity error that causes the numerical solution to drift out of phase with the exact one.

While the convergence study of Section 6 quantifies the global error at a fixed final time, it does not distinguish between these two mechanisms. In this section we design a dedicated experiment to isolate and visualise both effects for all five schemes.

7.1. Test Problem and Sweep Setup

We reuse the standing-mode problem (19) on $\Omega = [0,1]^2$ with the exact solution

$$u_{\text{exact}}(\mathbf{x},t) = \cos\!\big(\sqrt{2}\,\pi\,t\big)\,\sin(\pi x)\,\sin(\pi y), \qquad \omega_{\text{exact}} = \sqrt{2}\,\pi, \tag{20}$$

but now we extend the simulation to $T = 5\,\text{s}$ (roughly 3.5 full oscillation periods) in order to let phase and amplitude errors accumulate.

The spatial mesh is fixed at $N_{\text{el}} = 60$ elements per side with polynomial degree $r = 1$, so that computations needed for each timesteps are not too expensive. All five schemes are run with the same set of eleven time steps:

Parameter	**Values**
Δt	$1.5{\times}10^{-1}$, 10^{-1}, $5{\times}10^{-2}$, ... , 10^{-3}, $5{\times}10^{-4}$, 10^{-4}, $5{\times}10^{-5}$

For the two conditionally stable schemes (Forward Euler and explicit Newmark), the CFL condition is enforced and runs that violate it are automatically. The sweep is driven by the Python script `dissipation_dispersion_sweep.py`, which logs at every time step both the discrete energy and the numerical solution at the domain centre.

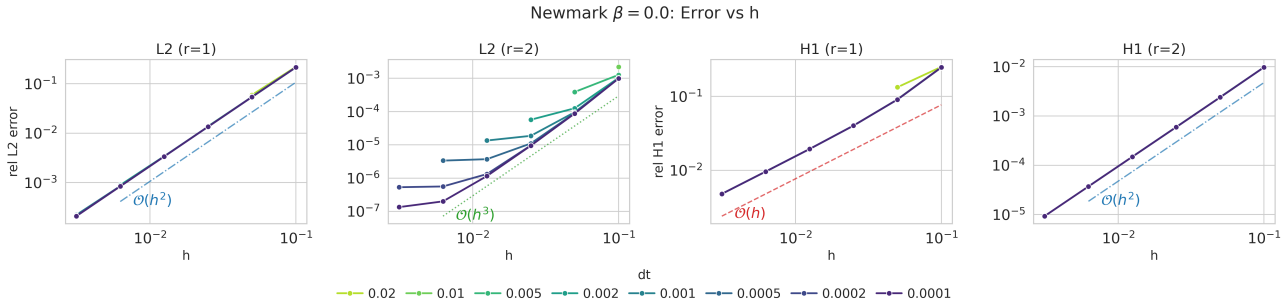

Figure 9: Spatial convergence for explicit Newmark ($\beta = 0,\ \gamma = 0.5$). Layout as in Figure 6.

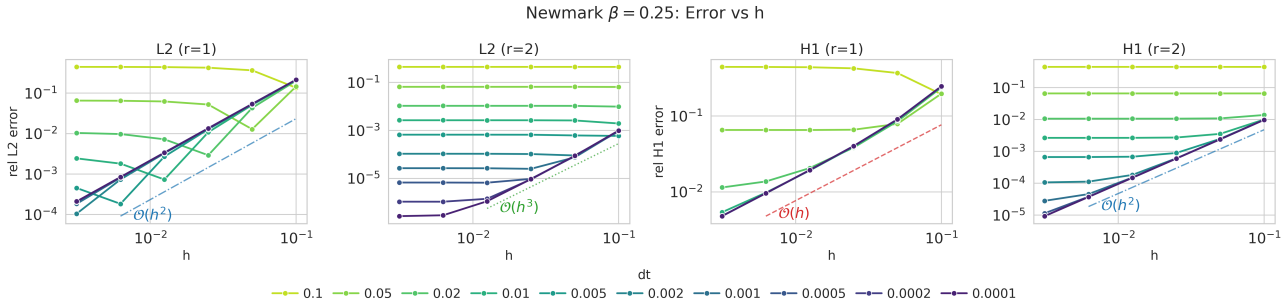

Figure 10: Spatial convergence for implicit Newmark ($\beta = 0.25,\ \gamma = 0.5$). Layout as in Figure 6.

7.2. Dissipation — Energy Evolution

The discrete energy associated with the finite-element semi-discretization is

$$E^n = \frac{1}{2}(\mathbf{V}^{n\,T}\mathbf{M}\,\mathbf{V}^n + \mathbf{U}^{n\,T}\mathbf{A}\,\mathbf{U}^n), \tag{21}$$

where $\mathbf{M}$ and $\mathbf{A}$ are the mass and stiffness matrices, respectively. For the continuous problem with homogeneous Dirichlet conditions and zero forcing, the energy is exactly conserved, so $E(t)/E(0) = 1$ if there is no numerical dissipation $E(t)/E(0) < 1$ if nergy decays.
Figure 11 shows the normalised energy $E(t)/E(0)$ as a function of time for each scheme and several values of Δt. The following behaviour is observed:

- **Crank–Nicolson** ($\theta = 0.5$) and **Newmark Average Acceleration** ($\beta = 0.25,\ \gamma = 0.5$) preserve the discrete energy to machine precision for all tested Δt, confirming their energy-conserving character.
- **Backward Euler** ($\theta = 1$) exhibits strong dissipation: the energy decays monotonically and the decay rate increases with Δt. For $\Delta t = 0.15$ the energy is almost entirely dissipated within the simulation window.
- **Forward Euler** ($\theta = 0$) and **explicit Newmark** ($\beta = 0$) are conditionally stable. For Δt values that satisfy the CFL condition the energy is preserved, while for larger Δt the energy grows without bound and the simulation diverges.

7.3. Dispersion — Point-Probe Comparison

To visualise dispersion we compare the numerical solution at the domain centre with the exact solution. Because $\sin(\pi/2) = 1$, the exact trace at $(0.5, 0.5)$ reduces to a pure cosine:

$$u_{\text{exact}}(0.5,\ 0.5,\ t) = \cos\!\left(\sqrt{2}\,\pi\,t\right). \tag{22}$$

A dispersive scheme produces a cosine with a slightly perturbed frequency $\tilde{\omega} \neq \omega_{\text{exact}}$, so the numerical and exact curves gradually drift out of phase. A dissipative scheme additionally shows amplitude decay. Figure 12 overlays the numerical and exact traces for each scheme. The main observations are:

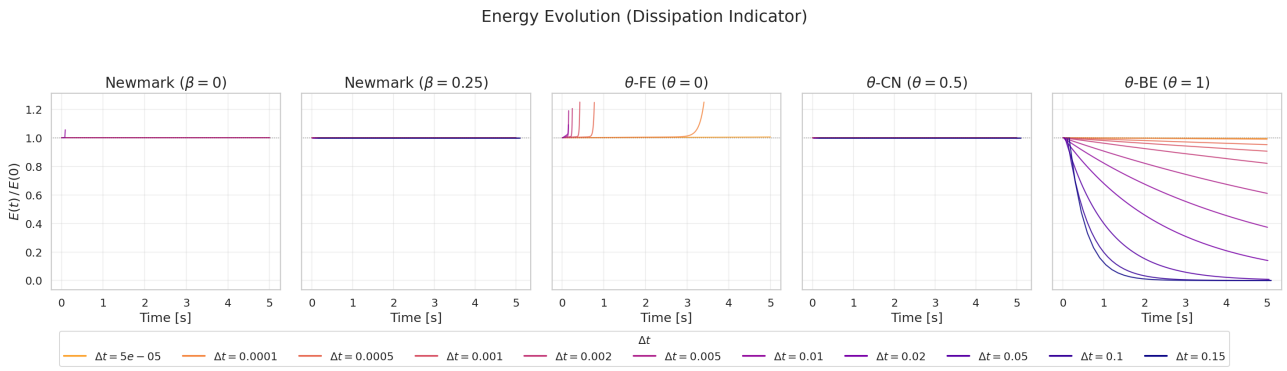

Figure 11: Normalised energy $E(t)/E(0)$ vs. time for each scheme. Each coloured line corresponds to a different Δt. Energy-conserving schemes ($\theta = 0.5$, Newmark $\beta = 0.25$) lie exactly on the dashed reference line $E/E_0 = 1$; Backward Euler shows progressive energy loss; explicit methods are either perfectly conservative (CFL satisfied) or diverge.

- **Forward Euler** and **explicit Newmark**: for Δt values within the CFL, bound the numerical solution overlaps with the exact cosine, consistent with the absence of both dissipation and dispersion. Larger Δt values lead to immediate divergence.
- **Crank–Nicolson** and **Newmark Average Acceleration**: the amplitude is perfectly preserved (no dissipation), but a clear phase shift develops for larger values of Δt. For example, at $\Delta t = 0.1$ a visible lag accumulates by $t \approx 3\,\mathrm{s}$, while for $\Delta t = 0.15$ the numerical wave is significantly out of phase by the end of the simulation. For sufficiently small Δt the dispersion error becomes negligible.
- **Backward Euler**: both amplitude decay and phase drift are evident. The numerical oscillation rapidly damps out for moderate Δt, rendering the solution qualitatively incorrect within a few periods. This confirms that Backward Euler is unsuitable for long-time wave propagation unless extremely small time steps are employed.

8. Scalability Analysis

A good numerical solver must not only be accurate but also able to exploit modern parallel hardware. In this section we assess the **strong scaling** behaviour of the MPI-parallel wave equation solver for all five time-integration schemes.

8.1. Experimental Setup

Test problem. We solve the standing-mode problem (19) on the unit square with $N_{\mathrm{el}} = 640$ elements per side, polynomial degree $r = 1$, time step $\Delta t = 8 \times 10^{-5}$, and final time $T = 0.05$ (625 time steps). The mesh is large enough (over 400 000 degrees of freedom) to let parallelism pay off, and Δt is small enough to keep the explicit schemes within their CFL bound. Solution output and CSV logging are disabled so that only computation time is measured.

Hardware. All tests were executed on the Students Cluster of the Mathematics Department at Politecnico di Milano. Each node is equipped with two **Intel Xeon Gold 6238R** processors (28 physical cores per socket, 38.5 MB L3 cache each) for a total of 56 physical cores per node.

MPI configuration. We use **OpenMPI** with `-bind-to core -map-by socket` binding and set `OMP_NUM_THREADS=1` to prevent hyper-threading. Each PBS job requests 16 CPUs on a *single node*; the process counts $p \in \{1, 2, 4, 8, 16\}$ are swept sequentially within the same job.

Methodology. For each (scheme, p) configuration we perform 3 repeat runs and retain the *minimum* wall time as the representative measurement. The code is copied and executed on `/scratch_local`, which is fast node-local temporary storage. The sweep is automated by the Python script `scalability_sweep.py`.

8.2. Metrics

Given the wall time $T(p)$ for p MPI processes, we define:

- **Speedup**: $S(p) = T(1)/T(p)$,
- **Parallel efficiency**: $E(p) = S(p)/p$.

Ideal (linear) scaling corresponds to $S(p) = p$ and $E(p) = 1$.

8.3. Results

Figure 13 shows wall time, speedup, and parallel efficiency as a function of the number of MPI processes for all five schemes. Table 1 reports the numerical values at $p = 16$.

Scheme	$T(1)$ [s]	$T(16)$ [s]	$S(16)$	$E(16)$
FE ($\theta = 0$)	668.5	58.0	11.52	0.72
CN ($\theta = 0.5$)	624.9	55.0	11.37	0.71
BE ($\theta = 1$)	624.9	54.6	11.44	0.72
Newmark ($\beta = 0$)	330.2	31.3	10.53	0.66
Newmark ($\beta = 0.25$)	296.3	27.6	10.75	0.67

Table 1: Strong scaling summary at $p = 16$ MPI processes. $T(1)$ and $T(16)$ are the best wall times (minimum over 3 repeats).

The main observations are:

- **All schemes scale well** up to 16 cores, with speedups between 10.5× and 11.5× and parallel efficiencies of 66–72%. The departure from ideal scaling is expected at this process count due to MPI communication overhead and the serial fraction of setup/assembly.
- **The three θ-methods are consistently clustered together** in wall time ($\sim$625–669 s serial, $\sim$55–58 s at $p = 16$). This is because each θ-method time step requires solving *two* symmetric positive-definite (SPD) linear systems (one for $\mathbf{U}^{n+1}$ and one for $\mathbf{V}^{n+1}$), and the cost of these solves dominates.
- **The Newmark family is roughly twice as fast** as the θ-methods ($\sim$296–330 s serial), since each Newmark step requires only *one* linear solve (for the acceleration $\mathbf{a}^{n+1}$), followed by explicit algebraic updates for $\mathbf{U}^{n+1}$ and $\mathbf{V}^{n+1}$.
- **Newmark schemes show slightly lower parallel efficiency** ($\sim$ 66–67% vs. $\sim$ 71–72% for the θ-methods). With fewer total floating-point operations per time step, the non-parallelisable overhead (MPI synchronisation, preconditioner setup, vector scatter/gather) is larger, reducing the parallel fraction of the computations.

9. Conclusions

A final section containing the main conclusions of your research/study and possible future developments of your work have to be inserted in the section “Conclusions”.

References

[1] Daniel Arndt, Wolfgang Bangerth, Bruno Blais, et al. The `deal.II` library, version 9.3. *Journal of Numerical Mathematics*, 29(3):171–186, 2021.

[2] Richard Courant, Kurt Friedrichs, and Hans Lewy. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100:32–74, 1928.

[3] Lawrence C Evans. *Partial Differential Equations*. American Mathematical Society, 2010.

[4] Thomas J. R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Courier Corporation, 2012.

[5] Nathan M. Newmark. A method of computation for structural dynamics. *Journal of the Engineering Mechanics Division*, 85(3):67–94, 1959.

[6] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*. Springer, 2007.

[7] Sandro Salsa. *Partial Differential Equations in Action: From Modelling to Theory*. Springer, 2015.

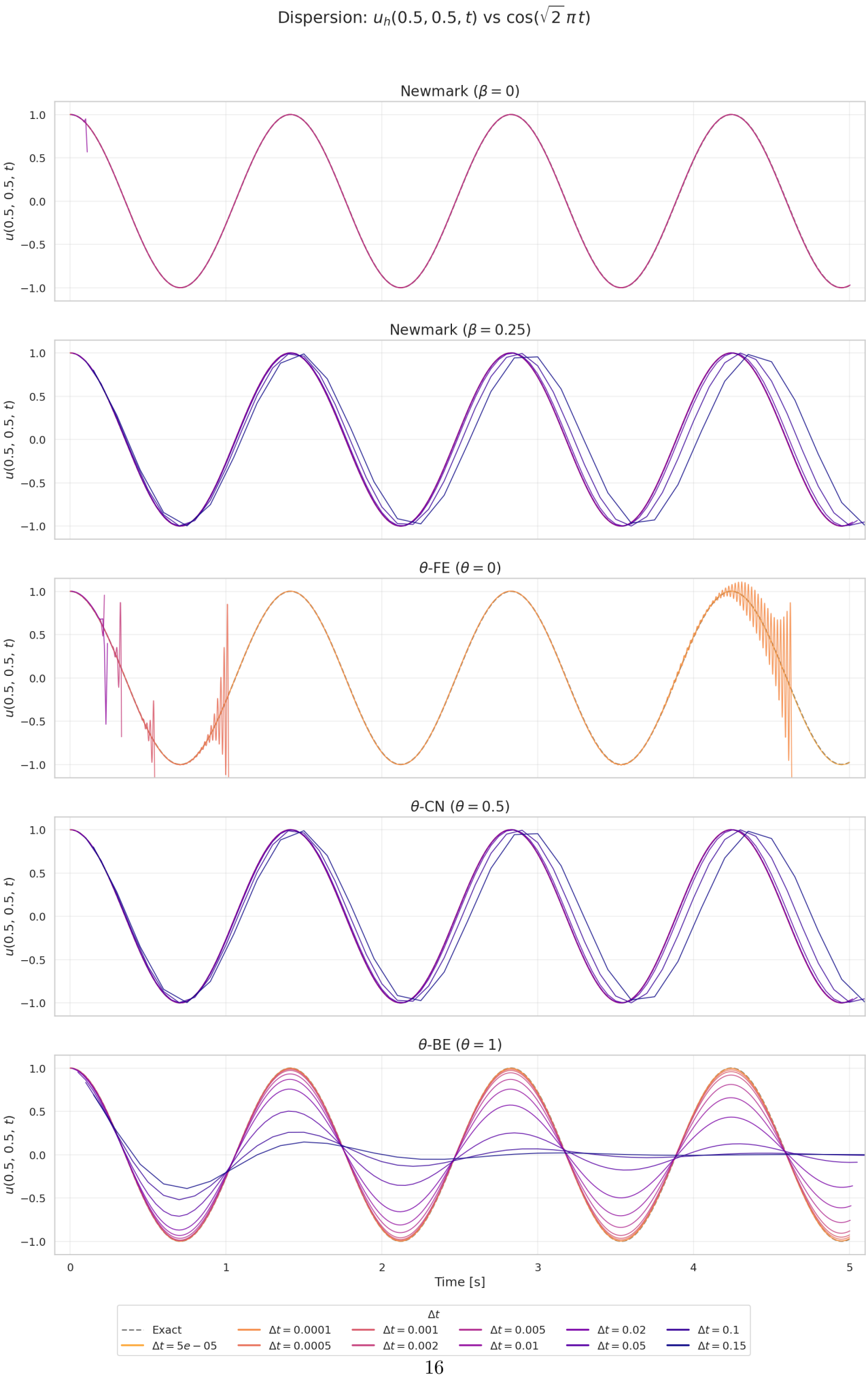

Figure 12: Numerical solution $u_h(0.5, 0.5, t)$ (coloured lines) vs. exact solution $\cos(\sqrt{2}\,\pi\,t)$ (dashed black) at the domain centre. Each sub-panel corresponds to a different scheme; colours encode Δt.

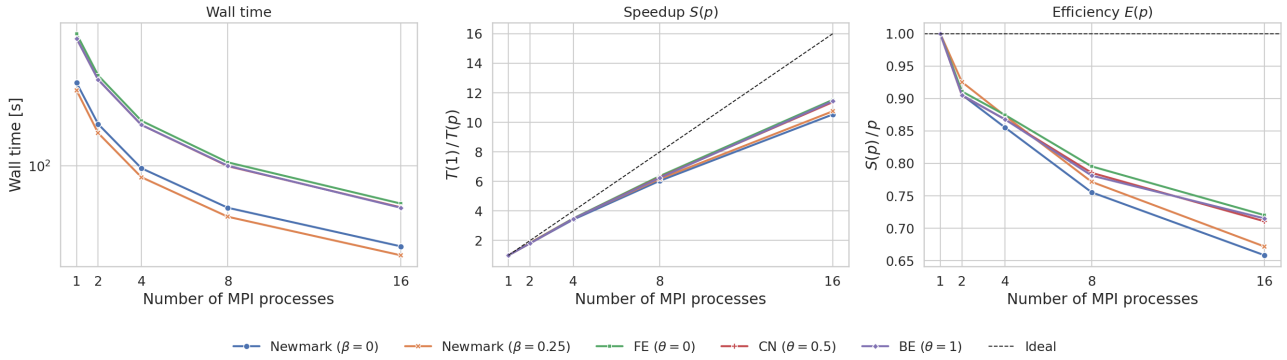

Figure 13: Strong scaling results. *Left:* wall time (log scale) vs. number of MPI processes. *Centre:* speedup $S(p) = T(1)/T(p)$ with the ideal-scaling reference (dashed). *Right:* parallel efficiency $E(p) = S(p)/p$.