Numerical Solvers for the Wave Equation

1.0.0

Generated by Doxygen 1.9.8

Chapter 1

Wave Equation — FEM Solver (deal.II)

Parallel finite-element solver for the 2D scalar wave equation

$$ \left\lbrace \begin{array}{lll} \displaystyle \frac{\partial^2 u}{\partial t^2} - c^2 \Delta u = f & \text{in } \Omega \times [0,T] & {} \\[10pt] u(x,t) = g & \text{on } \partial\Omega \times [0,T] & \text{(Dirichlet B.C.)} \\[10pt] u(x,0) = u_0(x) & \text{in } \Omega & \text{(Initial Displacement)} \\[10pt] \displaystyle \frac{\partial u}{\partial t}(x,0) = v_0(x) & \text{in } \Omega & \text{(Initial Velocity)} \end{array} \right. $$

on a rectangular domain Ω with simplicial (triangular) elements, built using `deal.II`, exploiting Trilinos and MPI parallelism.

1.1 Implemented Time-Integration Families

Method	Parameters	Key properties
Theta-method	$\theta \in [0,1]$	$\theta=0$ → Forward Euler (FE) $\theta=\tfrac12$ → Crank–Nicolson (CN) $\theta=1$ → Backward Euler (BE)
Newmark-	γ,β	$\beta=0$ → Central Difference method $\beta=\tfrac14, \gamma=\tfrac12$ → Middle Point rule

1.2 Repository structure

```
nmpde-wave-equation/
 include/              # C++ headers
    WaveEquationBase.hpp   # abstract base (mesh, FE, logging, output)
    WaveNewmark.hpp        # Newmark-beta solver
    WaveTheta.hpp          # theta-method solver
    ParameterReader.hpp    # JSON/PRM parameter parser
 src/                  # C++ sources
    WaveEquationBase.cpp
    WaveNewmark.cpp
    WaveTheta.cpp
    ParameterReader.cpp
    main-newmark.cpp       # executable entry point (Newmark)
    main-theta.cpp         # executable entry point (theta)
 parameters/           # JSON parameter files for different test cases
 scripts/              # Python sweep scripts + PBS job files
 analysis/             # Jupyter notebooks for post-processing
 results/              # simulation output (VTU, CSV) -- git-ignored
 build/                # CMake build directory -- git-ignored
 report/               # LaTeX report
```

1.3 Building

1.3.1 Prerequisites

- C++17 compiler with MPI support
- deal.II 9.3.1 (with Trilinos enabled)

You can get the Apptainer containter used in the project by executing the following commands on a terminal window

```
mkdir -p $HOME/apptainer-tmp/
mkdir -p $HOME/apptainer-cache/
export APPTAINER_TMPDIR=$HOME/apptainer-tmp/
export APPTAINER_CACHEDIR=$HOME/apptainer-cache/
apptainer pull docker://quay.io/pjbaioni/amsc_mk:2025
```

and afterwards load the container and the needed modules

```
apptainer shell /path/to/amsc_mk_2025.sif
source /u/sw/etc/bash.bashrc
module load gcc-glibc dealii
```

1.3.2 Compile

```
mkdir build
cd build
cmake ..
make -j
```

This produces two executables: `main-theta` and `main-newmark`.

1.4 Running a simulation

Both executables take a single command-line argument i.e. the path to a `JSON` parameter file.

```
cd build
mpirun -np 4 ./main-theta   ../parameters/standing-mode-wsol.json
mpirun -np 4 ./main-newmark ../parameters/gaussian-pulse.json
```

If no argument is given, the default `../parameters/sine-membrane.json` is used.

1.4.1 Parameter file format

A `JSON` parameter file with scalar entries and function subsections

```
{
  "Geometry": "[0.0, 1.0] x [0.0, 1.0]",             // domain bounding box
  "Nel": "80",                                       // elements per side (or "80, 60" for rectangular)
  "R": "1",                                          // polynomial degree
  "T": "1.0",                                        // final time
  "Theta": "0.5",                                    // theta parameter (theta-method only)
  "Beta": "0.25",                                    // beta  parameter (Newmark only)
  "Gamma": "0.5",                                    // gamma parameter (Newmark only)
  "Dt": "0.005",                                     // time step
  "Save Solution": true,                             // write VTU output
  "Enable Logging": true,                            // write energy/error CSVs
  "Log Every": 10,                                   // log frequency (0 = off)
  "Print Every": 10,                                 // console output frequency
  "C":  { "Function expression": "1.0", ... },       // wave speed c(x,y,t)
  "F":  { "Function expression": "0.0", ... },       // forcing f(x,y,t)
  "U0": { "Function expression": "sin(pi*x)*sin(pi*y)", ... },  // initial displacement
  "V0": { "Function expression": "0.0", ... },       // initial velocity
  "G":  { "Function expression": "0.0", ... },       // Dirichlet BC for u
  "DGDT": { "Function expression": "0.0", ... },     // time-derivative of g
  "Solution": { "Function expression": "...", ... }  // exact solution (optional)
}
```

Several already available parameter files are provided in `parameters/` (e.g. Gaussian pulse, Ricker wavelet, ...).

1.4.2 Output

The results of the simulation are written to the following folder `/results/<problem_name>/run-R...-N...-dt.../↩` :

- `solution_*.vtu` or `solution_*.pvtu` → displacement, velocity, and eventually the exact solution, ideal for being imported in ParaView
- `energy.csv` → discrete energy time series
- `error.csv` → L2 and H1 error and comparison w.r.t the exact solution, if available

- `probe.csv` → point probe at the domain centre
- `iterations.csv` → Conjugate Gradient (CG) method iteration counts
- `convergence.csv` → execution summary appended from different runs

1.5 C++ code overview

1.5.1 WaveEquationBase

Abstract base class providing:

- Mesh creation (`subdivided_hyper_rectangle_with_simplices`) and parallel partitioning
- FE space setup (`FE_SimplexP`, `QGaussSimplex`)
- DoF distribution
- Energy computation $E^n = \tfrac12 (\mathbf{v}^T M \mathbf{v} + \mathbf{u}^T K \mathbf{u})$
- L2 / H1 error integration against an exact solution
- VTU/PVTU output, CSV logging, divergence detection

1.5.2 WaveTheta

Inherits from `WaveEquationBase`. Rewrites the wave equation as a first-order system and applies the theta-method. Each time step solves **two** SPD systems (one for u^{n+1}, one for v^{n+1}) with CG + AMG.

1.5.3 WaveNewmark

Inherits from `WaveEquationBase`. Uses the Newmark-β family. Each time step solves **one** SPD system for the acceleration a^{n+1}, then updates u and v algebraically. A consistent initial acceleration a^0 is computed at startup by solving $M a^0 = f(0) - K u^0$.

1.5.4 ParameterReader

Thin wrapper around deal.II's `ParameterHandler`. Declares scalar parameters and function subsections, parses JSON/PRM files, and initialises `FunctionParser` objects (supporting symbolic `pi` constants).

1.6 Python scripts

All scripts live in `scripts/` and drive parametric studies by repeatedly invoking the C++ executables with different parameter combinations.

Python script	Purpose
`convergence_sweep.py`	Sweep over `(scheme, Nel, R, dt)` to study spatial and temporal convergence, applying also Courant-Friedrichs-Lewy (CFL) filtering for explicit methods.
`dissipation_dispersion_sweep.py`	Fix the mesh, sweep over `dt` for each scheme, logging energy values, errors and point-probing at each step to analyse numerical dissipation and dispersion.
`scalability_sweep.py`	Fix discretisation, measure the wall-clock time for a given number of MPI processes.

1.6.1 Common CLI flags

```
python3 convergence_sweep.py \
    --nprocs 4 \
```

```
    --nel 10 20 40 80 \
    --r 1 2 \
    --dt 0.01 0.005 0.001 \
    --T 1.0 \
    --schemes theta-0.5 newmark-0.25 \
    --timeout 600 \
    --cfl-safety 0.9
```

Each script produces CSV result files that are later read by the analysis notebooks.

1.7 Running on the cluster with the PBS scheduler

Three PBS job scripts are provided in `scripts/`:

PBS script	What it runs
`convergence_all.pbs`	`convergence_sweep.py` with 16 MPI processes
`dissipation_dispersion_all.pbs`	`dissipation_dispersion_sweep.py` with 16 MPI processes
`scalability_all.pbs`	`scalability_sweep.py` for p = 1, 2, 4, 8, 16 (sequential)

You should be sure to change the directories used in the scripts before submitting the job, and put your desired one.

1.7.1 Submitting a job

```
cd wave-equation/scripts
qsub convergence_all.pbs
```

1.7.2 What the PBS jobs do

1. **Copy the project to `scratch_local`** — the `scripts/`, `parameters/` and `build/` directories are copied to `/scratch_local/nmpde-<name>_${PBS_JOBID}/`. All computation happens on this fast node-local temporary storage and get much better I/O performance for the many small files produced by the sweeps.
2. **Run the Python sweep script** from the scratch directory with `--use-pbs-nodefile` and `--bind-to-core` binding MPI processes to cores, not to threads
3. **Copy results back** to a persistent location under the user's home directory (CSVs, compressed logs, raw convergence files).

Other notes:

- `OMP_NUM_THREADS=1` is set to prevent OpenMP oversubscription.
- For scalability tests, select a free node, so that memory bandwidth is not shared with other jobs.
- Core binding is enabled via `--bind-to core --map-by socket`.

1.8 Analysis notebooks

The `analysis/` folder contains Jupyter notebooks that read the `.csv` files produced by the Python scripts.

Notebook	Content
`convergence-analysis.ipynb`	Convergence plots (relative L2/H1 error vs. h and vs. Δt) for all five schemes. Computes observed convergence rates.
`dissipation-dispersion-analysis.↩ ipynb`	Energy ratio $E(T)/E(0)$ vs. Δt (dissipation) and point-probe time series vs. exact solution (dispersion).
`scalability-analisys.ipynb`	Strong-scaling plots: wall time, speedup and parallel efficiency vs. number of MPI processes. Includes Amdahl's law fit.

All the notebooks expect the `.csv` data to be placed in the folder `analysis/data/`.

1.9 Acknowledgments

We would like to thank Prof. Michele Bucelli for its valuable advices during the execution of the project.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- Function
  - BoundaryValuesU< dim > . . . . . **??**
  - BoundaryValuesV< dim > . . . . . **??**
  - InitialValuesU< dim > . . . . . **??**
  - InitialValuesV< dim > . . . . . **??**
  - RightHandSide< dim > . . . . . **??**
  - WaveSpeed< dim > . . . . . **??**
- ParameterReader . . . . . **??**
- WaveEquationBase . . . . . **??**
  - WaveNewmark . . . . . **??**
  - WaveTheta . . . . . **??**

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

BoundaryValuesU< dim >
: Default Dirichlet boundary data for u . . . . . **??**

BoundaryValuesV< dim >
: Default Dirichlet boundary data for $v = \partial_t u$ . . . . . **??**

InitialValuesU< dim >
: Default initial displacement: $u_0(\mathbf{x}) \equiv 0$ . . . . . **??**

InitialValuesV< dim >
: Default initial velocity: $v_0(\mathbf{x}) \equiv 0$ . . . . . **??**

ParameterReader
: Helper that declares, parses and loads simulation parameters . . . . . **??**

RightHandSide< dim >
: Default forcing term: $f(\mathbf{x}, t) \equiv 0$ . . . . . **??**

WaveEquationBase
: Abstract base class for 2-D wave-equation solvers . . . . . **??**

WaveNewmark
: Concrete wave-equation solver with the Newmark- β method . . . . . **??**

WaveSpeed< dim >
: Default wave speed: $c(\mathbf{x}) \equiv 1$ . . . . . **??**

WaveTheta
: Concrete wave-equation solver with the theta-method . . . . . **??**

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

/home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/ParameterReader.hpp
Wrapper around deal.II's ParameterHandler for parsing JSON/PRM parameter files used by the wave-equation solvers . . . . . ??

/home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveEquationBase.hpp
Abstract base class for parallel finite-element wave-equation solvers . . . . . ??

/home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveNewmark.hpp
Wave-equation solver using the Newmark-beta time-integration scheme . . . . . ??

/home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveTheta.hpp
Wave-equation solver using the theta-method time discretisation . . . . . ??

/home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/main-newmark.cpp
Entry point for the Newmark wave-equation solver . . . . . ??

/home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/main-theta.cpp
Entry point for the theta-method wave-equation solver . . . . . ??

/home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/ParameterReader.cpp
Implementation of the ParameterReader class and helper parsing utilities . . . . . ??

/home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/WaveEquationBase.cpp
Implementation of the WaveEquationBase common infrastructure . . . . . ??

/home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/WaveNewmark.cpp
Implementation of the WaveNewmark solver . . . . . ??

/home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/WaveTheta.cpp
Implementation of the WaveTheta solver . . . . . ??

Chapter 5

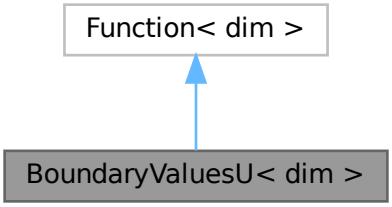
Class Documentation

5.1 BoundaryValuesU< dim > Class Template Reference

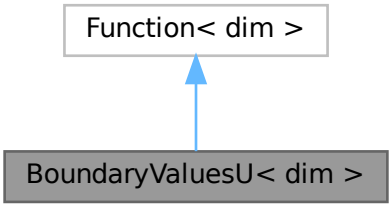
Default Dirichlet boundary data for u.

`#include <WaveTheta.hpp>`

Inheritance diagram for BoundaryValuesU< dim >:



Collaboration diagram for BoundaryValuesU< dim >:



Public Member Functions

- virtual double **value** (const Point< dim > &p, const unsigned int component=0) const override

5.1.1 Detailed Description

template<int dim>
class BoundaryValuesU< dim >

Default Dirichlet boundary data for u.
A sinusoidal pulse is applied on a portion of the left boundary $(x < 0.5,\ y \in (1/3, 2/3))$ for $t \leq 0.5$. Elsewhere the value is zero.

Template Parameters

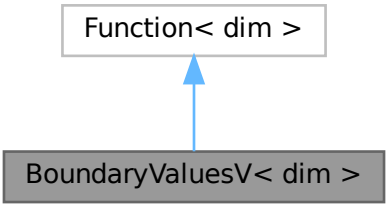
dim	Spatial dimension.

The documentation for this class was generated from the following file:

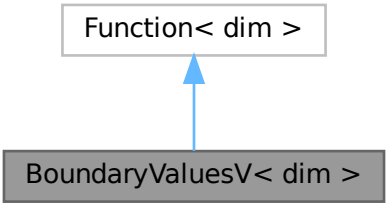
- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveTheta.hpp

5.2 BoundaryValuesV< dim > Class Template Reference

Default Dirichlet boundary data for $v = \partial_t u$.
`#include <WaveTheta.hpp>`
Inheritance diagram for BoundaryValuesV< dim >:



Collaboration diagram for BoundaryValuesV< dim >:



Public Member Functions

- virtual double **value** (const Point< dim > &p, const unsigned int component=0) const override

5.2.1 Detailed Description

template<int dim>
class BoundaryValuesV< dim >

Default Dirichlet boundary data for $v = \partial_t u$.
Time-derivative of BoundaryValuesU: a cosine pulse on the same boundary strip for $t \leq 0.5$, zero elsewhere.

Template Parameters

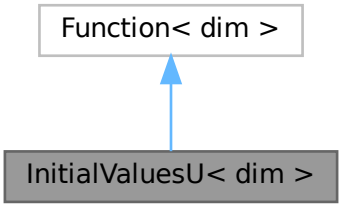
dim	Spatial dimension.

The documentation for this class was generated from the following file:

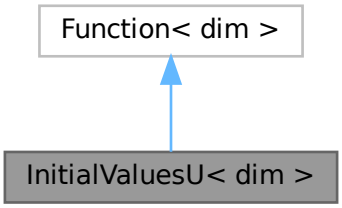
- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveTheta.hpp

5.3 InitialValuesU< dim > Class Template Reference

Default initial displacement: $u_0(\mathbf{x}) \equiv 0$.
`#include <WaveTheta.hpp>`
Inheritance diagram for InitialValuesU< dim >:



Collaboration diagram for InitialValuesU< dim >:



Public Member Functions

- virtual double **value** (const Point< dim > &, const unsigned int component=0) const override

5.3.1 Detailed Description

template<**int dim**>
class InitialValuesU< **dim** >

Default initial displacement: $u_0(\mathbf{x}) \equiv 0$.

Template Parameters

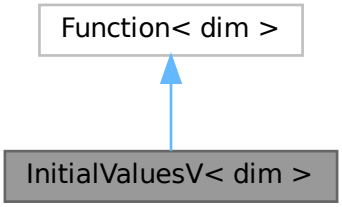
dim	Spatial dimension.

The documentation for this class was generated from the following file:

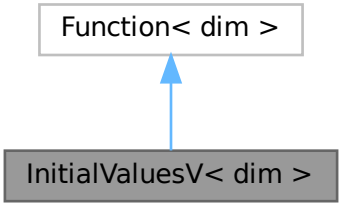
- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveTheta.hpp

5.4 InitialValuesV< dim > Class Template Reference

Default initial velocity: $v_0(\mathbf{x}) \equiv 0$.
`#include <WaveTheta.hpp>`
Inheritance diagram for InitialValuesV< dim >:



Collaboration diagram for InitialValuesV< dim >:



Public Member Functions

- virtual double **value** (const Point< dim > &, const unsigned int component=0) const override

5.4.1 Detailed Description

template<int dim>
class InitialValuesV< dim >

Default initial velocity: $v_0(\mathbf{x}) \equiv 0$.

Template Parameters

dim	Spatial dimension.

The documentation for this class was generated from the following file:

- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveTheta.hpp

5.5 ParameterReader Class Reference

Helper that declares, parses and loads simulation parameters.
`#include <ParameterReader.hpp>`

Public Member Functions

- ParameterReader (ParameterHandler &paramhandler)

  Construct a reader bound to an existing ParameterHandler.
- void declare (const std::vector< std::string > &function_names)

  Declare all scalar entries and function subsections.
- void parse (const std::string &filename)

  Parse a parameter file (JSON or PRM format).
- void load_functions (const std::vector< std::string > &names, const std::vector< FunctionParser< dim > * > &funcs)

  Initialise FunctionParser objects from the parsed subsections.
- std::pair< unsigned int, unsigned int > get_nel () const

  Read the "Nel" entry and return (N_el_x, N_el_y).
- std::pair< Point< dim >, Point< dim > > get_geometry () const

  Read the "Geometry" entry and return the bounding box.

5.5.1 Detailed Description

Helper that declares, parses and loads simulation parameters.
Usage pattern:

```
ParameterHandler prm;
ParameterReader reader(prm);
reader.declare({"C", "F", "U0", "V0", "G", "DGDT", "Solution"});
reader.parse("parameters.json");
reader.load_functions(names, func_ptrs);
```

5.5.2 Constructor & Destructor Documentation

5.5.2.1 ParameterReader()

```
ParameterReader::ParameterReader (
            ParameterHandler & paramhandler )
```

Construct a reader bound to an existing ParameterHandler.

Parameters

paramhandler	Reference to the deal.II ParameterHandler instance.

5.5.3 Member Function Documentation

5.5.3.1 declare()

```
void ParameterReader::declare (
            const std::vector< std::string > & function_names )
```

Declare all scalar entries and function subsections.

Parameters

function_names	Names of the function subsections to declare (e.g. "C", "F", "U0", ...).

5.5.3.2 get_geometry()

```
std::pair< Point< dim >, Point< dim > > ParameterReader::get_geometry ( ) const
```

Read the "Geometry" entry and return the bounding box.
Expected format: "`[x_min, x_max] x [y_min, y_max]`".

Returns

Pair of Point<dim> (bottom-left, top-right).

5.5.3.3 get_nel()

```
std::pair< unsigned int, unsigned int > ParameterReader::get_nel ( ) const
```

Read the "Nel" entry and return (N_el_x, N_el_y).
A single value is duplicated for a square grid; two comma-separated values give a rectangular grid.

Returns

Pair of element counts (x, y).

5.5.3.4 load_functions()

```
void ParameterReader::load_functions (
            const std::vector< std::string > & names,
            const std::vector< FunctionParser< dim > * > & funcs )
```

Initialise FunctionParser objects from the parsed subsections.
Each name in `names` must match a previously declared subsection. The corresponding FunctionParser pointer in `funcs` is initialised with the expression and constants read from the file. The special subsection "Solution" is silently skipped when its expression is empty.

Parameters

names	Subsection names (same order as `funcs`).
funcs	Pointers to FunctionParser objects to initialise.

5.5.3.5 parse()

```
void ParameterReader::parse (
            const std::string & filename )
```

Parse a parameter file (JSON or PRM format).

Parameters

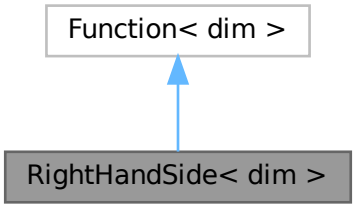
filename	Path to the parameter file.

The documentation for this class was generated from the following files:

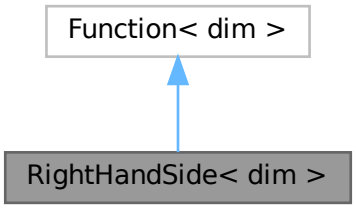
- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/ParameterReader.hpp
- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/ParameterReader.cpp

5.6 RightHandSide< dim > Class Template Reference

Default forcing term: $f(\mathbf{x}, t) \equiv 0$.
`#include <WaveTheta.hpp>`
Inheritance diagram for RightHandSide< dim >:



Collaboration diagram for RightHandSide< dim >:



Public Member Functions

- virtual double **value** (const Point< dim > &, const unsigned int component=0) const override

5.6.1 Detailed Description

template<int dim>
class RightHandSide< dim >

Default forcing term: $f(\mathbf{x}, t) \equiv 0$.

Template Parameters

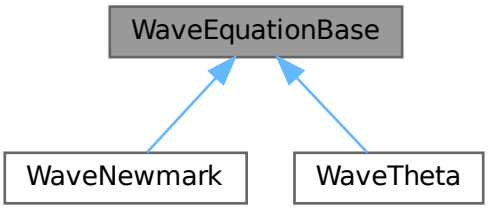
dim	Spatial dimension.

The documentation for this class was generated from the following file:

- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveTheta.hpp

5.7 WaveEquationBase Class Reference

Abstract base class for 2-D wave-equation solvers.
`#include <WaveEquationBase.hpp>`
Inheritance diagram for WaveEquationBase:



Public Member Functions

- virtual ~**WaveEquationBase** ()=default

  Virtual destructor for safe polymorphic deletion.
- virtual void run ()=0

  Execute the full simulation (pure-virtual).

Static Public Attributes

- static constexpr unsigned int **dim** = 2

  Spatial dimension (2-D).

Protected Member Functions

- WaveEquationBase (const std::string &problem_name_, const std::pair< unsigned int, unsigned int > &N↩ _el_, const std::pair< Point< dim >, Point< dim > > &geometry_, const unsigned int &r_, const double &T_, const double &delta_t_, const Function< dim > &c_, Function< dim > &f_, const Function< dim > &u0_, const Function< dim > &v0_, Function< dim > &g_, Function< dim > &dgdt_, const unsigned int log_every_, const unsigned int print_every_, Function< dim > *exact_solution_)

  Construct the base solver (called by derived-class constructors).
- void **setup_mesh** ()

  Create a simplicial rectangle, partition it and build the distributed triangulation. Also writes a VTK file of the mesh.
- void **setup_fe** ()

  Instantiate the FE_SimplexP element and Gauss quadrature rule.
- void **setup_dof_handler** ()

  Distribute DoFs on the mesh.
- void prepare_output_filename (const std::string &method_params)

  Build the output folder path and open convergence CSV.
- void **compute_and_log_energy** ()

  Compute the discrete energy $E^n = 12(v^T M v + u^T K u)$ and append it to the energy CSV log.
- void **log_point_probe** ()

Evaluate u at the centre of the domain and log to CSV.

- void log_iterations (const unsigned int n_iterations_1, const unsigned int n_iterations_2)

  Log the number of CG iterations for the current time step.
- void **compute_and_log_error** ()

  Compute L2 and H1 errors against the exact solution and log to CSV.
- void **compute_final_errors** ()

  Compute and log final errors (convenience overload with empty strings).
- void compute_final_errors (const std::string &theta_str, const std::string &beta_str, const std::string &gamma_str)

  Compute final errors and append a row to the convergence CSV.
- void **print_step_info** ()

  Print a one-line summary of the current time step to stdout.
- void **output** () const

  Write VTU/PVTU output files for the current time step.
- double compute_error (const VectorTools::NormType &cell_norm, const Function< dim > &exact_solution) const

  Integrate the error between the FE solution and an exact function.
- double compute_relative_error (const double error, const VectorTools::NormType &norm_type, const Function< dim > &exact_solution) const

  Compute a relative error $\|u_h - u\|/\|u\|$.
- bool check_divergence (const double norm_u, const double norm_v, const double threshold) const

  Check whether the solution norms exceed a threshold (divergence detector).

Protected Attributes

- const std::string **problem_name**

  Human-readable problem name (used in folder paths).
- std::string **output_folder**

  Full path of the output folder for this run.
- std::ofstream **energy_log_file**

  Output stream for the energy time-series CSV.
- std::ofstream **error_log_file**

  Output stream for the error time-series CSV.
- std::ofstream **convergence_file**

  Output stream for the convergence-study CSV (appended across runs).
- std::ofstream **iterations_log_file**

  Output stream for the CG-iteration count CSV.
- std::ofstream **point_probe_log_file**

  Output stream for the point-probe (mid-domain) CSV.
- std::pair< unsigned int, unsigned int > **N_el**

  Number of elements in the x- and y-directions.
- const std::pair< Point< dim >, Point< dim > > **geometry**

  Bounding box corners (bottom-left, top-right).
- const unsigned int **r**

  Polynomial degree of the finite-element space.
- const double **T**

  Final simulation time.
- const double **delta_t**

  Time-step size Δt.
- double **time** = 0.0

  Current simulation time.
- unsigned int **timestep_number** = 0

Current time-step index.

- double **current_energy**

  Discrete energy at the current time step.
- unsigned int **error_sample_count** = 0

  Counter for accumulated error samples (unused legacy).
- double **simulation_time** = 0.0

  Wall-clock simulation time in seconds.
- const Function< dim > & **c**

  Wave-speed function $c(\mathbf{x})$.
- Function< dim > & **f**

  Forcing term $f(\mathbf{x}, t)$.
- const Function< dim > & **u0**

  Initial displacement $u_0(\mathbf{x})$.
- const Function< dim > & **v0**

  Initial velocity $v_0(\mathbf{x})$.
- Function< dim > & **g**

  Dirichlet boundary data $g(\mathbf{x}, t)$ for u.
- Function< dim > & **dgdt**

  Time-derivative of the Dirichlet data $\partial_t g$.
- const unsigned int **log_every**

  Write energy / error CSVs every this many steps (0 = disabled).
- const unsigned int **print_every**

  Print step info to stdout every this many steps.
- Function< dim > * **exact_solution**

  Pointer to a manufactured exact solution (nullptr if unavailable).
- const unsigned int **mpi_size**

  Total number of MPI processes.
- const unsigned int **mpi_rank**

  Rank of this MPI process.
- parallel::fullydistributed::Triangulation< dim > **mesh**

  Fully-distributed MPI triangulation.
- std::unique_ptr< FiniteElement< dim > > **fe**

  Finite-element object (FE_SimplexP of degree r).
- std::unique_ptr< Quadrature< dim > > **quadrature**

  Gauss quadrature rule on simplices.
- DoFHandler< dim > **dof_handler**

  Degree-of-freedom handler.
- TrilinosWrappers::SparseMatrix **mass_matrix**

  Global mass matrix M.
- TrilinosWrappers::SparseMatrix **stiffness_matrix**

  Global stiffness matrix K (includes c^2 weighting).
- TrilinosWrappers::MPI::Vector **solution_u**

  Current displacement u^{n+1}.
- TrilinosWrappers::MPI::Vector **solution_v**

  Current velocity v^{n+1}.
- TrilinosWrappers::MPI::Vector **old_solution_u**

  Previous displacement u^n.
- TrilinosWrappers::MPI::Vector **old_solution_v**

  Previous velocity v^n.
- TrilinosWrappers::MPI::Vector **system_rhs**

  Right-hand side vector of the current linear system.
- ConditionalOStream **pcout**

  Conditional output stream (prints only on rank 0).

5.7.1 Detailed Description

Abstract base class for 2-D wave-equation solvers.
Solves the scalar wave equation

$$\partial_{tt} u - c^2 \Delta u = f \quad in \Omega \times (0, T]$$

on a rectangular simplicial mesh distributed across MPI ranks.
Derived classes (WaveTheta, WaveNewmark) implement the actual time-stepping loop by overriding run().

5.7.2 Constructor & Destructor Documentation

5.7.2.1 WaveEquationBase()

```
WaveEquationBase::WaveEquationBase (
            const std::string & problem_name_,
            const std::pair< unsigned int, unsigned int > & N_el_,
            const std::pair< Point< dim >, Point< dim > > & geometry_,
            const unsigned int & r_,
            const double & T_,
            const double & delta_t_,
            const Function< dim > & c_,
            Function< dim > & f_,
            const Function< dim > & u0_,
            const Function< dim > & v0_,
            Function< dim > & g_,
            Function< dim > & dgdt_,
            const unsigned int log_every_,
            const unsigned int print_every_,
            Function< dim > * exact_solution_ )  [inline], [protected]
```

Construct the base solver (called by derived-class constructors).

Parameters

*problem_name_*	Human-readable name used for output folder naming.
*N_el_*	Number of elements in x- and y-direction.
*geometry_*	Bounding box of the rectangular domain (bottom-left, top-right corners).
*r_*	Polynomial degree of the FE space.
*T_*	Final simulation time.
*delta_t_*	Time-step size.
*c_*	Wave-speed function $c(\mathbf{x})$.
*f_*	Forcing (source) term $f(\mathbf{x}, t)$.
*u0_*	Initial displacement $u(\mathbf{x}, 0)$.
*v0_*	Initial velocity $\partial_t u(\mathbf{x}, 0)$.
*g_*	Dirichlet boundary data for u.
*dgdt_*	Time derivative of the Dirichlet data $\partial_t g$ (needed by the theta-method for the velocity system).
*log_every_*	Write energy / error CSVs every n steps (0 = off).
*print_every_*	Print step info to stdout every n steps.
*exact_solution_*	Pointer to the manufactured exact solution (nullptr when not available).

5.7.3 Member Function Documentation

5.7.3.1 check_divergence()

```
bool WaveEquationBase::check_divergence (
```

```
            const double norm_u,
            const double norm_v,
            const double threshold ) const  [protected]
```

Check whether the solution norms exceed a threshold (divergence detector).

Parameters

norm_u	L2 norm of u.
norm_v	L2 norm of v.
threshold	Blow-up threshold.

Returns

True if the solution has diverged.

5.7.3.2 compute_error()

```
double WaveEquationBase::compute_error (
            const VectorTools::NormType & cell_norm,
            const Function< dim > & exact_solution ) const  [protected]
```

Integrate the error between the FE solution and an exact function.

Parameters

cell_norm	Norm type (L2_norm, H1_norm, ...).
exact_solution	Reference exact-solution function.

Returns

Global (reduced across MPI ranks) error value.

5.7.3.3 compute_final_errors()

```
void WaveEquationBase::compute_final_errors (
            const std::string & theta_str,
            const std::string & beta_str,
            const std::string & gamma_str )  [protected]
```

Compute final errors and append a row to the convergence CSV.

Parameters

theta_str	String representation of theta (or empty).
beta_str	String representation of beta (or empty).
gamma_str	String representation of gamma (or empty).

5.7.3.4 compute_relative_error()

```
double WaveEquationBase::compute_relative_error (
            const double error,
            const VectorTools::NormType & norm_type,
            const Function< dim > & exact_solution ) const  [protected]
```

Compute a relative error $\|u_h - u\| / \|u\|$.

Parameters

error	Absolute error (previously computed).
norm_type	Norm type used for the denominator.
exact_solution	Exact-solution function.

Returns

Relative error (returns absolute error when the exact norm is $<$ 1e-14).

5.7.3.5 log_iterations()

```
void WaveEquationBase::log_iterations (
            const unsigned int n_iterations_1,
            const unsigned int n_iterations_2 )  [protected]
```

Log the number of CG iterations for the current time step.

Parameters

n_iterations_1	Iteration count for the first linear system.
n_iterations_2	Iteration count for the second system (0 if N/A).

5.7.3.6 prepare_output_filename()

```
void WaveEquationBase::prepare_output_filename (
            const std::string & method_params )  [protected]
```

Build the output folder path and open convergence CSV.

Parameters

method_params	Suffix encoding method-specific parameters (e.g. "-theta0_5" or "-gamma0_5-beta0_25").

5.7.3.7 run()

```
virtual void WaveEquationBase::run ( )  [pure virtual]
```

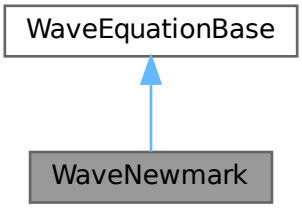
Execute the full simulation (pure-virtual).
Concrete solvers set up the mesh, assemble matrices, march in time and produce output.
Implemented in WaveNewmark, and WaveTheta.
The documentation for this class was generated from the following files:

- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveEquationBase.hpp
- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/WaveEquationBase.cpp

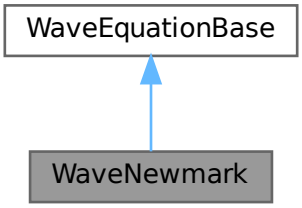
5.8 WaveNewmark Class Reference

Concrete wave-equation solver with the Newmark- β method.
`#include <WaveNewmark.hpp>`

Inheritance diagram for WaveNewmark:



Collaboration diagram for WaveNewmark:



Public Member Functions

- WaveNewmark (const std::string &problem_name_, const std::pair< unsigned int, unsigned int > &N_↩ el_, const std::pair< Point< dim >, Point< dim > > &geometry_, const unsigned int &r_, const double &T_, const double &gamma_, const double &beta_, const double &delta_t_, const Function< dim > &c_, Function< dim > &f_, const Function< dim > &u0_, const Function< dim > &v0_, Function< dim > &g_, Function< dim > &dgdt_, const unsigned int log_every_=10, const unsigned int print_every_=10, Function< dim > ∗exact_solution_=nullptr)

  Construct the Newmark solver.
- void run () override

  Run the full Newmark time-stepping simulation.

Public Member Functions inherited from WaveEquationBase

- virtual ∼**WaveEquationBase** ()=default

  Virtual destructor for safe polymorphic deletion.

Protected Member Functions

- void **setup** ()

  Initialise mesh, FE, DoF handler, sparsity pattern and vectors.
- void **assemble_matrices** ()

  Assemble the global mass and stiffness matrices and form the clean (no-BC) system matrix $M + \beta\,\Delta t^2\,K$.

- void assemble_rhs ()

  Assemble the right-hand side for the acceleration system.
- void solve_a ()

  Solve the linear system for a^{n+1} and apply Dirichlet BCs.
- void update_u_v ()

  Update displacement and velocity from the newly computed acceleration.
- void apply_dirichlet_bc ()

  Apply Dirichlet boundary conditions.

Protected Member Functions inherited from WaveEquationBase

- WaveEquationBase (const std::string &problem_name_, const std::pair< unsigned int, unsigned int > &N↩ _el_, const std::pair< Point< dim >, Point< dim > > &geometry_, const unsigned int &r_, const double &T_, const double &delta_t_, const Function< dim > &c_, Function< dim > &f_, const Function< dim > &u0_, const Function< dim > &v0_, Function< dim > &g_, Function< dim > &dgdt_, const unsigned int log_every_, const unsigned int print_every_, Function< dim > *exact_solution_)

  Construct the base solver (called by derived-class constructors).
- void **setup_mesh** ()

  Create a simplicial rectangle, partition it and build the distributed triangulation. Also writes a VTK file of the mesh.
- void **setup_fe** ()

  Instantiate the FE_SimplexP element and Gauss quadrature rule.
- void **setup_dof_handler** ()

  Distribute DoFs on the mesh.
- void prepare_output_filename (const std::string &method_params)

  Build the output folder path and open convergence CSV.
- void **compute_and_log_energy** ()

  Compute the discrete energy $E^n = 12(v^T M v + u^T K u)$ and append it to the energy CSV log.
- void **log_point_probe** ()

  Evaluate u at the centre of the domain and log to CSV.
- void log_iterations (const unsigned int n_iterations_1, const unsigned int n_iterations_2)

  Log the number of CG iterations for the current time step.
- void **compute_and_log_error** ()

  Compute L2 and H1 errors against the exact solution and log to CSV.
- void **compute_final_errors** ()

  Compute and log final errors (convenience overload with empty strings).
- void compute_final_errors (const std::string &theta_str, const std::string &beta_str, const std::string &gamma_str)

  Compute final errors and append a row to the convergence CSV.
- void **print_step_info** ()

  Print a one-line summary of the current time step to stdout.
- void **output** () const

  Write VTU/PVTU output files for the current time step.
- double compute_error (const VectorTools::NormType &cell_norm, const Function< dim > &exact_solution) const

  Integrate the error between the FE solution and an exact function.
- double compute_relative_error (const double error, const VectorTools::NormType &norm_type, const Function< dim > &exact_solution) const

  Compute a relative error $\|u_h - u\|/\|u\|$.
- bool check_divergence (const double norm_u, const double norm_v, const double threshold) const

  Check whether the solution norms exceed a threshold (divergence detector).

Protected Attributes

- const double **gamma**

  Newmark parameter γ (velocity weighting).
- const double **beta**

  Newmark parameter β (displacement weighting).
- TrilinosWrappers::MPI::Vector **solution_a**

  Current acceleration a^{n+1}.
- TrilinosWrappers::MPI::Vector **old_solution_a**

  Previous acceleration a^n.
- TrilinosWrappers::SparseMatrix matrix_a

  Clean system matrix $M + \beta\,\Delta t^2\,K$ (no BCs).
- TrilinosWrappers::SparseMatrix system_matrix_a

  BC-modified system matrix.
- TrilinosWrappers::PreconditionAMG **preconditioner_a**

  Cached AMG preconditioner for the acceleration system.
- bool **preconditioner_a_initialized** = false

  Whether preconditioner_a has been initialised.
- unsigned int **current_iterations**

  Number of CG iterations in the last solve_a() call.

Protected Attributes inherited from WaveEquationBase

- const std::string **problem_name**

  Human-readable problem name (used in folder paths).
- std::string **output_folder**

  Full path of the output folder for this run.
- std::ofstream **energy_log_file**

  Output stream for the energy time-series CSV.
- std::ofstream **error_log_file**

  Output stream for the error time-series CSV.
- std::ofstream **convergence_file**

  Output stream for the convergence-study CSV (appended across runs).
- std::ofstream **iterations_log_file**

  Output stream for the CG-iteration count CSV.
- std::ofstream **point_probe_log_file**

  Output stream for the point-probe (mid-domain) CSV.
- std::pair< unsigned int, unsigned int > **N_el**

  Number of elements in the x- and y-directions.
- const std::pair< Point< dim >, Point< dim > > **geometry**

  Bounding box corners (bottom-left, top-right).
- const unsigned int **r**

  Polynomial degree of the finite-element space.
- const double **T**

  Final simulation time.
- const double **delta_t**

  Time-step size Δt.
- double **time** = 0.0

  Current simulation time.
- unsigned int **timestep_number** = 0

  Current time-step index.
- double **current_energy**

Discrete energy at the current time step.

- unsigned int **error_sample_count** = 0

  Counter for accumulated error samples (unused legacy).
- double **simulation_time** = 0.0

  Wall-clock simulation time in seconds.
- const Function< dim > & **c**

  Wave-speed function $c(\mathbf{x})$.
- Function< dim > & **f**

  Forcing term $f(\mathbf{x}, t)$.
- const Function< dim > & **u0**

  Initial displacement $u_0(\mathbf{x})$.
- const Function< dim > & **v0**

  Initial velocity $v_0(\mathbf{x})$.
- Function< dim > & **g**

  Dirichlet boundary data $g(\mathbf{x}, t)$ for u.
- Function< dim > & **dgdt**

  Time-derivative of the Dirichlet data $\partial_t g$.
- const unsigned int **log_every**

  Write energy / error CSVs every this many steps (0 = disabled).
- const unsigned int **print_every**

  Print step info to stdout every this many steps.
- Function< dim > * **exact_solution**

  Pointer to a manufactured exact solution (nullptr if unavailable).
- const unsigned int **mpi_size**

  Total number of MPI processes.
- const unsigned int **mpi_rank**

  Rank of this MPI process.
- parallel::fullydistributed::Triangulation< dim > **mesh**

  Fully-distributed MPI triangulation.
- std::unique_ptr< FiniteElement< dim > > **fe**

  Finite-element object (FE_SimplexP of degree r).
- std::unique_ptr< Quadrature< dim > > **quadrature**

  Gauss quadrature rule on simplices.
- DoFHandler< dim > **dof_handler**

  Degree-of-freedom handler.
- TrilinosWrappers::SparseMatrix **mass_matrix**

  Global mass matrix M.
- TrilinosWrappers::SparseMatrix **stiffness_matrix**

  Global stiffness matrix K (includes c^2 weighting).
- TrilinosWrappers::MPI::Vector **solution_u**

  Current displacement u^{n+1}.
- TrilinosWrappers::MPI::Vector **solution_v**

  Current velocity v^{n+1}.
- TrilinosWrappers::MPI::Vector **old_solution_u**

  Previous displacement u^n.
- TrilinosWrappers::MPI::Vector **old_solution_v**

  Previous velocity v^n.
- TrilinosWrappers::MPI::Vector **system_rhs**

  Right-hand side vector of the current linear system.
- ConditionalOStream **pcout**

  Conditional output stream (prints only on rank 0).

Additional Inherited Members

Static Public Attributes inherited from WaveEquationBase

- static constexpr unsigned int **dim** = 2

  Spatial dimension (2-D).

5.8.1 Detailed Description

Concrete wave-equation solver with the Newmark- β method.
The Newmark update reads: $u^{n+1} = u^n + \Delta t\, v^n + \Delta t^2 \big[(12-\beta)\, a^n + \beta\, a^{n+1}\big]$,
$v^{n+1} = v^n + \Delta t \big[(1-\gamma)\, a^n + \gamma\, a^{n+1}\big]$, where a^{n+1} is obtained from the linear system $(M + \beta\, \Delta t^2\, K)\, a^{n+1} = f^{n+1} - K\, z$ with $z = u^n + \Delta t\, v^n + \Delta t^2 (12-\beta)\, a^n$.
Common parameter choices:

- $\gamma = 1/2,\ \beta = 1/4$ (average-acceleration, unconditionally stable)
- $\gamma = 1/2,\ \beta = 0$ (explicit central differences)

5.8.2 Constructor & Destructor Documentation

5.8.2.1 WaveNewmark()

```
WaveNewmark::WaveNewmark (
            const std::string & problem_name_,
            const std::pair< unsigned int, unsigned int > & N_el_,
            const std::pair< Point< dim >, Point< dim > > & geometry_,
            const unsigned int & r_,
            const double & T_,
            const double & gamma_,
            const double & beta_,
            const double & delta_t_,
            const Function< dim > & c_,
            Function< dim > & f_,
            const Function< dim > & u0_,
            const Function< dim > & v0_,
            Function< dim > & g_,
            Function< dim > & dgdt_,
            const unsigned int log_every_ = 10,
            const unsigned int print_every_ = 10,
            Function< dim > * exact_solution_ = nullptr )  [inline]
```

Construct the Newmark solver.

Parameters

*problem_↩ name_*	Human-readable problem name.
*N_el_*	Number of mesh elements (x, y).
*geometry_*	Domain bounding box.
*r_*	FE polynomial degree.
*T_*	Final time.
*gamma_*	Newmark parameter γ.
*beta_*	Newmark parameter β.
*delta_t_*	Time-step size.
*c_*	Wave-speed function.
*f_*	Forcing term.
*u0_*	Initial displacement.
*v0_*	Initial velocity.

Parameters

*g_*	Dirichlet BC for u.
*dgdt_*	Time-derivative of the Dirichlet data.
*log_every_*	Logging frequency (0 = off).
*print_every_*	Console output frequency.
*exact_↩ solution_*	Pointer to exact solution (nullptr if unavailable).

5.8.3 Member Function Documentation

5.8.3.1 apply_dirichlet_bc()

`void WaveNewmark::apply_dirichlet_bc ( ) [protected]`
Apply Dirichlet boundary conditions.

Note

BCs are applied inside solve_a(); this is a legacy stub.

5.8.3.2 assemble_rhs()

`void WaveNewmark::assemble_rhs ( ) [protected]`
Assemble the right-hand side for the acceleration system.
Computes $\mathrm{rhs} = f^{n+1} - K\,z$ where $z = u^n + \Delta t\,v^n + \Delta t^2(12-\beta)\,a^n$.

5.8.3.3 run()

`void WaveNewmark::run ( ) [override], [virtual]`
Run the full Newmark time-stepping simulation.
Sets up mesh and FE space, assembles M and K, computes a consistent initial acceleration a^0, then marches in time until $t = T$.
Implements WaveEquationBase.

5.8.3.4 solve_a()

`void WaveNewmark::solve_a ( ) [protected]`
Solve the linear system for a^{n+1} and apply Dirichlet BCs.
Uses CG with an AMG preconditioner. The preconditioner is built once on the first call and reused for subsequent time steps.

5.8.3.5 update_u_v()

`void WaveNewmark::update_u_v ( ) [protected]`
Update displacement and velocity from the newly computed acceleration.
Applies the algebraic Newmark update formulas for u^{n+1} and v^{n+1}.

5.8.4 Member Data Documentation

5.8.4.1 matrix_a

`TrilinosWrappers::SparseMatrix WaveNewmark::matrix_a [protected]`
Clean system matrix $M + \beta\,\Delta t^2\,K$ (no BCs).
Copied into system_matrix_a before applying boundary values.

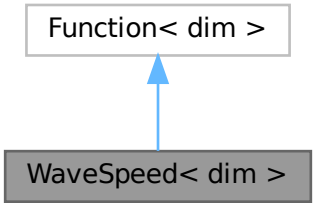
5.8.4.2 system_matrix_a

`TrilinosWrappers::SparseMatrix WaveNewmark::system_matrix_a [protected]`
BC-modified system matrix.
The AMG preconditioner keeps an internal pointer to this matrix, so it must not be reinitialised after the preconditioner is built.
The documentation for this class was generated from the following files:

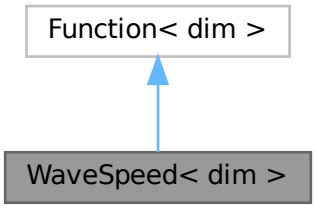
- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveNewmark.hpp
- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/WaveNewmark.cpp

5.9 WaveSpeed< dim > Class Template Reference

Default wave speed: $c(\mathbf{x}) \equiv 1$.
`#include <WaveTheta.hpp>`
Inheritance diagram for WaveSpeed< dim >:



Collaboration diagram for WaveSpeed< dim >:



Public Member Functions

- virtual double **value** (const Point< dim > &, const unsigned int=0) const override

5.9.1 Detailed Description

template<int dim>
class WaveSpeed< dim >

Default wave speed: $c(\mathbf{x}) \equiv 1$.

Template Parameters

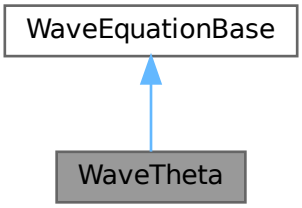
dim	Spatial dimension.

The documentation for this class was generated from the following file:

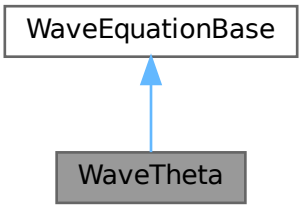
- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveTheta.hpp

5.10 WaveTheta Class Reference

Concrete wave-equation solver with the theta-method.
`#include <WaveTheta.hpp>`
Inheritance diagram for WaveTheta:



Collaboration diagram for WaveTheta:



Public Member Functions

- WaveTheta (const std::string &problem_name_, const std::pair< unsigned int, unsigned int > &N_el_, const std::pair< Point< dim >, Point< dim > > &geometry_, const unsigned int &r_, const double &T_, const double &theta_, const double &delta_t_, const Function< dim > &c_, Function< dim > &f_, const Function< dim > &u0_, const Function< dim > &v0_, Function< dim > &g_, Function< dim > &dgdt_, const unsigned int log_every_=10, const unsigned int print_every_=10, Function< dim > *exact_solution_=nullptr)

  Construct the theta-method solver.
- void run () override

  Run the full theta-method time-stepping simulation.

Public Member Functions inherited from WaveEquationBase

- virtual ~**WaveEquationBase** ()=default

  Virtual destructor for safe polymorphic deletion.

Protected Member Functions

- void **setup** ()

  Initialise mesh, FE, DoF handler, sparsity pattern and vectors.
- void assemble_matrices ()

  Assemble M and K, then form the clean (no-BC) system matrices for u and v.
- void assemble_rhs_u ()

  Assemble the right-hand side for the displacement system.
- void assemble_rhs_v ()

  Assemble the right-hand side for the velocity system.
- void solve_u ()

  Solve the displacement system for u^{n+1}.
- void solve_v ()

  Solve the velocity system for v^{n+1}.

Protected Member Functions inherited from WaveEquationBase

- WaveEquationBase (const std::string &problem_name_, const std::pair< unsigned int, unsigned int > &N↩ _el_, const std::pair< Point< dim >, Point< dim > > &geometry_, const unsigned int &r_, const double &T_, const double &delta_t_, const Function< dim > &c_, Function< dim > &f_, const Function< dim > &u0_, const Function< dim > &v0_, Function< dim > &g_, Function< dim > &dgdt_, const unsigned int log_every_, const unsigned int print_every_, Function< dim > *exact_solution_)

  Construct the base solver (called by derived-class constructors).
- void **setup_mesh** ()

  Create a simplicial rectangle, partition it and build the distributed triangulation. Also writes a VTK file of the mesh.
- void **setup_fe** ()

  Instantiate the FE_SimplexP element and Gauss quadrature rule.
- void **setup_dof_handler** ()

  Distribute DoFs on the mesh.
- void prepare_output_filename (const std::string &method_params)

  Build the output folder path and open convergence CSV.
- void **compute_and_log_energy** ()

  Compute the discrete energy $E^n = 12(v^T M v + u^T K u)$ and append it to the energy CSV log.
- void **log_point_probe** ()

  Evaluate u at the centre of the domain and log to CSV.
- void log_iterations (const unsigned int n_iterations_1, const unsigned int n_iterations_2)

  Log the number of CG iterations for the current time step.
- void **compute_and_log_error** ()

  Compute L2 and H1 errors against the exact solution and log to CSV.
- void **compute_final_errors** ()

  Compute and log final errors (convenience overload with empty strings).
- void compute_final_errors (const std::string &theta_str, const std::string &beta_str, const std::string &gamma_str)

  Compute final errors and append a row to the convergence CSV.
- void **print_step_info** ()

  Print a one-line summary of the current time step to stdout.
- void **output** () const

  Write VTU/PVTU output files for the current time step.

- double compute_error (const VectorTools::NormType &cell_norm, const Function< dim > &exact_solution) const

  Integrate the error between the FE solution and an exact function.
- double compute_relative_error (const double error, const VectorTools::NormType &norm_type, const Function< dim > &exact_solution) const

  Compute a relative error $\|u_h - u\|/\|u\|$.
- bool check_divergence (const double norm_u, const double norm_v, const double threshold) const

  Check whether the solution norms exceed a threshold (divergence detector).

Protected Attributes

- const double **theta**

  Theta parameter $\theta \in [0, 1]$ *for the time-stepping scheme.*
- TrilinosWrappers::SparseMatrix **matrix_u**

  Clean system matrix for u*:* $M + (\theta\,\Delta t)^2\,K$ *(no BCs).*
- TrilinosWrappers::SparseMatrix **matrix_v**

  Clean system matrix for v*:* M *(no BCs).*
- TrilinosWrappers::SparseMatrix system_matrix_u

  BC-modified system matrix for the displacement solve.
- TrilinosWrappers::SparseMatrix **system_matrix_v**

  BC-modified system matrix for the velocity solve.
- TrilinosWrappers::PreconditionAMG **preconditioner_u**

  Cached AMG preconditioner for the displacement system.
- TrilinosWrappers::PreconditionAMG **preconditioner_v**

  Cached AMG preconditioner for the velocity system.
- bool **preconditioner_u_initialized** = false

  Whether preconditioner_u has been initialised.
- bool **preconditioner_v_initialized** = false

  Whether preconditioner_v has been initialised.
- unsigned int **current_iterations_u**

  Number of CG iterations in the last solve_u() call.
- unsigned int **current_iterations_v**

  Number of CG iterations in the last solve_v() call.

Protected Attributes inherited from WaveEquationBase

- const std::string **problem_name**

  Human-readable problem name (used in folder paths).
- std::string **output_folder**

  Full path of the output folder for this run.
- std::ofstream **energy_log_file**

  Output stream for the energy time-series CSV.
- std::ofstream **error_log_file**

  Output stream for the error time-series CSV.
- std::ofstream **convergence_file**

  Output stream for the convergence-study CSV (appended across runs).
- std::ofstream **iterations_log_file**

  Output stream for the CG-iteration count CSV.
- std::ofstream **point_probe_log_file**

  Output stream for the point-probe (mid-domain) CSV.
- std::pair< unsigned int, unsigned int > **N_el**

  Number of elements in the x- and y-directions.

- const std::pair< Point< dim >, Point< dim > > **geometry**

  Bounding box corners (bottom-left, top-right).
- const unsigned int **r**

  Polynomial degree of the finite-element space.
- const double **T**

  Final simulation time.
- const double **delta_t**

  Time-step size Δt.
- double **time** = 0.0

  Current simulation time.
- unsigned int **timestep_number** = 0

  Current time-step index.
- double **current_energy**

  Discrete energy at the current time step.
- unsigned int **error_sample_count** = 0

  Counter for accumulated error samples (unused legacy).
- double **simulation_time** = 0.0

  Wall-clock simulation time in seconds.
- const Function< dim > & **c**

  Wave-speed function $c(\mathbf{x})$.
- Function< dim > & **f**

  Forcing term $f(\mathbf{x}, t)$.
- const Function< dim > & **u0**

  Initial displacement $u_0(\mathbf{x})$.
- const Function< dim > & **v0**

  Initial velocity $v_0(\mathbf{x})$.
- Function< dim > & **g**

  Dirichlet boundary data $g(\mathbf{x}, t)$ for u.
- Function< dim > & **dgdt**

  Time-derivative of the Dirichlet data $\partial_t g$.
- const unsigned int **log_every**

  Write energy / error CSVs every this many steps (0 = disabled).
- const unsigned int **print_every**

  Print step info to stdout every this many steps.
- Function< dim > * **exact_solution**

  Pointer to a manufactured exact solution (nullptr if unavailable).
- const unsigned int **mpi_size**

  Total number of MPI processes.
- const unsigned int **mpi_rank**

  Rank of this MPI process.
- parallel::fullydistributed::Triangulation< dim > **mesh**

  Fully-distributed MPI triangulation.
- std::unique_ptr< FiniteElement< dim > > **fe**

  Finite-element object (FE_SimplexP of degree r).
- std::unique_ptr< Quadrature< dim > > **quadrature**

  Gauss quadrature rule on simplices.
- DoFHandler< dim > **dof_handler**

  Degree-of-freedom handler.
- TrilinosWrappers::SparseMatrix **mass_matrix**

  Global mass matrix M.
- TrilinosWrappers::SparseMatrix **stiffness_matrix**

Global stiffness matrix K (includes c^2 weighting).

- TrilinosWrappers::MPI::Vector **solution_u**

  Current displacement u^{n+1}.
- TrilinosWrappers::MPI::Vector **solution_v**

  Current velocity v^{n+1}.
- TrilinosWrappers::MPI::Vector **old_solution_u**

  Previous displacement u^n.
- TrilinosWrappers::MPI::Vector **old_solution_v**

  Previous velocity v^n.
- TrilinosWrappers::MPI::Vector **system_rhs**

  Right-hand side vector of the current linear system.
- ConditionalOStream **pcout**

  Conditional output stream (prints only on rank 0).

Additional Inherited Members

Static Public Attributes inherited from WaveEquationBase

- static constexpr unsigned int **dim** = 2

  Spatial dimension (2-D).

5.10.1 Detailed Description

Concrete wave-equation solver with the theta-method.
Rewrites the second-order wave equation as a first-order system $\partial_t u = v$,
$M\,\partial_t v = -K\,u + f$, and applies the theta-method ($\theta \in [0,1]$) for time integration: (M + $\theta^2 \Delta t^2\,K)\,u^{n+1} = M\,u^n + \Delta t\,M\,v^n - \theta(1-\theta)\,\Delta t^2\,K\,u^n + \theta\,\Delta t^2\,F_\theta$,
$M\,v^{n+1} = M\,v^n - \Delta t\big[(1-\theta)\,K\,u^n + \theta\,K\,u^{n+1}\big] + \Delta t\,F_\theta$, where $F_\theta = \theta f^{n+1} + (1-\theta) f^n$.
Special cases: $\theta = 0$ (explicit forward Euler), $\theta = 1/2$ (Crank–Nicolson), $\theta = 1$ (implicit backward Euler).

5.10.2 Constructor & Destructor Documentation

5.10.2.1 WaveTheta()

```
WaveTheta::WaveTheta (
            const std::string & problem_name_,
            const std::pair< unsigned int, unsigned int > & N_el_,
            const std::pair< Point< dim >, Point< dim > > & geometry_,
            const unsigned int & r_,
            const double & T_,
            const double & theta_,
            const double & delta_t_,
            const Function< dim > & c_,
            Function< dim > & f_,
            const Function< dim > & u0_,
            const Function< dim > & v0_,
            Function< dim > & g_,
            Function< dim > & dgdt_,
            const unsigned int log_every_ = 10,
            const unsigned int print_every_ = 10,
            Function< dim > * exact_solution_ = nullptr )  [inline]
```

Construct the theta-method solver.

Parameters

*problem_↩ name_*	Human-readable problem name.

Parameters

*N_el_*	Number of mesh elements (x, y).
*geometry_*	Domain bounding box.
*r_*	FE polynomial degree.
*T_*	Final time.
*theta_*	Theta parameter $\theta \in [0, 1]$.
*delta_t_*	Time-step size.
*c_*	Wave-speed function.
*f_*	Forcing term.
*u0_*	Initial displacement.
*v0_*	Initial velocity.
*g_*	Dirichlet BC for u.
*dgdt_*	Time-derivative of the Dirichlet data.
*log_every_*	Logging frequency (0 = off).
*print_every_*	Console output frequency.
*exact_solution_*	Pointer to exact solution (nullptr if unavailable).

5.10.3 Member Function Documentation

5.10.3.1 assemble_matrices()

`void WaveTheta::assemble_matrices ( ) [protected]`

Assemble M and K, then form the clean (no-BC) system matrices for u and v.

- matrix_u $= M + (\theta\,\Delta t)^2\,K$
- matrix_v $= M$

5.10.3.2 assemble_rhs_u()

`void WaveTheta::assemble_rhs_u ( ) [protected]`

Assemble the right-hand side for the displacement system.
Includes contributions from Mu^n, Mv^n, the stiffness term and the theta-weighted forcing.

5.10.3.3 assemble_rhs_v()

`void WaveTheta::assemble_rhs_v ( ) [protected]`

Assemble the right-hand side for the velocity system.
Uses the freshly computed u^{n+1} in the stiffness contribution.

5.10.3.4 run()

`void WaveTheta::run ( ) [override], [virtual]`

Run the full theta-method time-stepping simulation.
Sets up mesh and FE space, assembles M and K, sets initial conditions, then marches in time until $t = T$.
Implements WaveEquationBase.

5.10.3.5 solve_u()

`void WaveTheta::solve_u ( ) [protected]`

Solve the displacement system for u^{n+1}.
Applies Dirichlet BCs $u|_{\partial\Omega} = g(t^{n+1})$ and solves with CG + AMG.

5.10.3.6 solve_v()

`void WaveTheta::solve_v ( ) [protected]`
Solve the velocity system for v^{n+1}.
Applies Dirichlet BCs $v|_{\partial\Omega} = \partial_t g(t^{n+1})$ and solves with CG + AMG.

5.10.4 Member Data Documentation

5.10.4.1 system_matrix_u

`TrilinosWrappers::SparseMatrix WaveTheta::system_matrix_u [protected]`
BC-modified system matrix for the displacement solve.
The AMG preconditioner keeps an internal pointer to this matrix.
The documentation for this class was generated from the following files:

- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveTheta.hpp
- /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/WaveTheta.cpp

Chapter 6

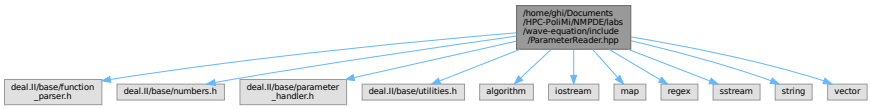
File Documentation

6.1 /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/ParameterReader.hpp File Reference

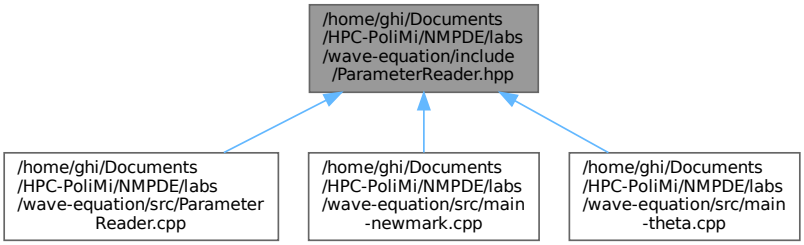
Wrapper around deal.II's ParameterHandler for parsing JSON/PRM parameter files used by the wave-equation solvers.

```
#include <deal.II/base/function_parser.h>
#include <deal.II/base/numbers.h>
#include <deal.II/base/parameter_handler.h>
#include <deal.II/base/utilities.h>
#include <algorithm>
#include <iostream>
#include <map>
#include <regex>
#include <sstream>
#include <string>
#include <vector>
```

Include dependency graph for ParameterReader.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class ParameterReader

  Helper that declares, parses and loads simulation parameters.

6.1.1 Detailed Description

Wrapper around deal.II's ParameterHandler for parsing JSON/PRM parameter files used by the wave-equation solvers.
Declares scalar parameters (mesh size, polynomial degree, time-stepping constants, ...) and function subsections (wave speed, forcing, initial / boundary data) and loads them into FunctionParser objects.

6.2 ParameterReader.hpp

Go to the documentation of this file.

```
00001
00011 #ifndef PARAMETER_READER_HPP
00012 #define PARAMETER_READER_HPP
00013
00014 #include <deal.II/base/function_parser.h>
00015 #include <deal.II/base/numbers.h>
00016 #include <deal.II/base/parameter_handler.h>
00017 #include <deal.II/base/utilities.h>
00018
00019 #include <algorithm>
00020 #include <iostream>
00021 #include <map>
00022 #include <regex>
00023 #include <sstream>
00024 #include <string>
00025 #include <vector>
00026
00027 using namespace dealii;
00028
00030 static constexpr unsigned int dim = 2;
00031
00045 class ParameterReader
00046 {
00047   public:
00052     ParameterReader(ParameterHandler& paramhandler);
00053
00059     void declare(const std::vector<std::string>& function_names);
00060
00065     void parse(const std::string& filename);
00066
00078     void load_functions(const std::vector<std::string>& names,
00079                         const std::vector<FunctionParser<dim>*>& funcs);
00080
00089     std::pair<unsigned int, unsigned int> get_nel() const;
00090
00098     std::pair<Point<dim>, Point<dim>> get_geometry() const;
00099
00100   private:
00102     void declare_scalar_parameters();
00103
00108     void declare_function_subsections(const std::vector<std::string>& names);
00109
00111     ParameterHandler& prm;
00112 };
00113
00114 #endif // PARAMETER_READER_HPP
```

6.3 /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveEquationBase.hpp File Reference

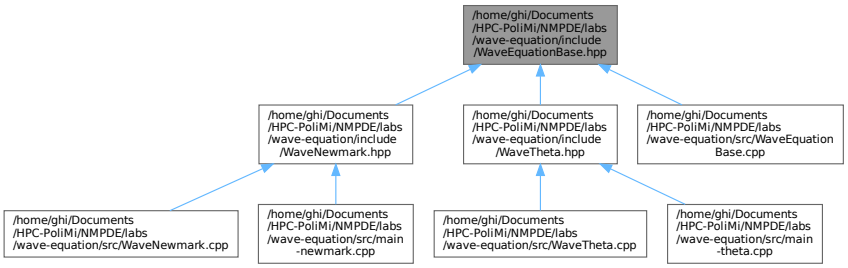
Abstract base class for parallel finite-element wave-equation solvers.

```
#include <deal.II/base/conditional_ostream.h>
#include <deal.II/base/quadrature_lib.h>
#include <deal.II/distributed/fully_distributed_tria.h>
#include <deal.II/dofs/dof_handler.h>
#include <deal.II/dofs/dof_tools.h>
#include <deal.II/fe/fe_simplex_p.h>
```

```
#include <deal.II/fe/fe_values.h>
#include <deal.II/fe/mapping_fe.h>
#include <deal.II/grid/grid_generator.h>
#include <deal.II/grid/grid_out.h>
#include <deal.II/grid/grid_tools.h>
#include <deal.II/lac/trilinos_precondition.h>
#include <deal.II/lac/trilinos_sparse_matrix.h>
#include <deal.II/numerics/data_out.h>
#include <deal.II/numerics/matrix_tools.h>
#include <deal.II/numerics/vector_tools.h>
#include <chrono>
#include <cstdlib>
#include <filesystem>
#include <fstream>
#include <iomanip>
#include <iostream>
```

Include dependency graph for WaveEquationBase.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class WaveEquationBase

  Abstract base class for 2-D wave-equation solvers.

Functions

- std::string clean_double (double x, int precision=6)

  Convert a double to a filesystem-safe string.

6.3.1 Detailed Description

Abstract base class for parallel finite-element wave-equation solvers.
Provides the mesh setup, FE space initialisation, assembly infrastructure, energy/error logging, VTU output, and MPI bookkeeping that are shared by every time-stepping scheme (theta-method, Newmark, ...). Concrete solvers inherit from WaveEquationBase and implement the pure-virtual run() method.
Built on top of the deal.II library with Trilinos linear-algebra back-end.

6.3.2 Function Documentation

6.3.2.1 clean_double()

```
std::string clean_double (
            double x,
            int precision = 6 )
```

Convert a double to a filesystem-safe string.
Trailing zeros and decimal points are removed; the dot is replaced by an underscore so the result can be used in file/folder names.

Parameters

x	The value to convert.
precision	Number of decimal digits (default 6).

Returns

Sanitised string representation (e.g. 0.25 → "0_25").

6.4 WaveEquationBase.hpp

Go to the documentation of this file.

```
00001
00013 #ifndef WAVE_EQUATION_BASE_HPP
00014 #define WAVE_EQUATION_BASE_HPP
00015
00016 #include <deal.II/base/conditional_ostream.h>
00017 #include <deal.II/base/quadrature_lib.h>
00018 #include <deal.II/distributed/fully_distributed_tria.h>
00019 #include <deal.II/dofs/dof_handler.h>
00020 #include <deal.II/dofs/dof_tools.h>
00021 #include <deal.II/fe/fe_simplex_p.h>
00022 #include <deal.II/fe/fe_values.h>
00023 #include <deal.II/fe/mapping_fe.h>
00024 #include <deal.II/grid/grid_generator.h>
00025 #include <deal.II/grid/grid_out.h>
00026 #include <deal.II/grid/grid_tools.h>
00027 #include <deal.II/lac/trilinos_precondition.h>
00028 #include <deal.II/lac/trilinos_sparse_matrix.h>
00029 #include <deal.II/numerics/data_out.h>
00030 #include <deal.II/numerics/matrix_tools.h>
00031 #include <deal.II/numerics/vector_tools.h>
00032
00033 #include <chrono>
00034 #include <cstdlib>
00035 #include <filesystem>
00036 #include <fstream>
00037 #include <iomanip>
00038 #include <iostream>
00039
00040 using namespace dealii;
00041
00055 class WaveEquationBase
00056 {
00057   public:
00059     static constexpr unsigned int dim = 2;
00060
00062     virtual ~WaveEquationBase() = default;
00063
00070     virtual void run() = 0;
00071
00072   protected:
00096     WaveEquationBase(
00097         const std::string& problem_name_,
00098         const std::pair<unsigned int, unsigned int>& N_el_,
00099         const std::pair<Point<dim>, Point<dim»& geometry_,
00100         const unsigned int& r_,
00101         const double& T_,
00102         const double& delta_t_,
00103         const Function<dim>& c_,
00104         Function<dim>& f_,
00105         const Function<dim>& u0_,
00106         const Function<dim>& v0_,
00107         Function<dim>& g_,
00108         Function<dim>& dgdt_,
```

```
00109         const unsigned int log_every_,
00110         const unsigned int print_every_,
00111         Function<dim>* exact_solution_)
00112         : problem_name(problem_name_), N_el(N_el_), geometry(geometry_),
00113           r(r_), T(T_), delta_t(delta_t_), c(c_), f(f_),
00114           u0(u0_), v0(v0_), g(g_), dgdt(dgdt_),
00115           log_every(log_every_), print_every(print_every_),
00116           exact_solution(exact_solution_),
00117           mpi_size(Utilities::MPI::n_mpi_processes(MPI_COMM_WORLD)),
00118           mpi_rank(Utilities::MPI::this_mpi_process(MPI_COMM_WORLD)),
00119           mesh(MPI_COMM_WORLD), pcout(std::cout, mpi_rank == 0)
00120     {
00121     }
00122
00123     // ---- Mesh & FE initialisation ----------------------------------------
00124
00127     void setup_mesh();
00128
00130     void setup_fe();
00131
00133     void setup_dof_handler();
00134
00135     // ---- Output & logging ------------------------------------------------
00136
00142     void prepare_output_filename(const std::string& method_params);
00143
00147     void compute_and_log_energy();
00148
00150     void log_point_probe();
00151
00157     void log_iterations(const unsigned int n_iterations_1, const unsigned int n_iterations_2);
00158
00160     void compute_and_log_error();
00161
00163     void compute_final_errors();
00164
00171     void compute_final_errors(const std::string& theta_str,
00172                               const std::string& beta_str,
00173                               const std::string& gamma_str);
00174
00176     void print_step_info();
00177
00179     void output() const;
00180
00181     // ---- Error computation -----------------------------------------------
00182
00189     double compute_error(const VectorTools::NormType& cell_norm, const Function<dim>& exact_solution)
      const;
00190
00198     double compute_relative_error(const double error,
00199                                   const VectorTools::NormType& norm_type,
00200                                   const Function<dim>& exact_solution) const;
00201
00209     bool check_divergence(const double norm_u,
00210                           const double norm_v,
00211                           const double threshold) const;
00212
00213     // ---- Problem description ---------------------------------------------
00214
00216     const std::string problem_name;
00217
00219     std::string output_folder;
00220
00222     std::ofstream energy_log_file;
00223
00225     std::ofstream error_log_file;
00226
00228     std::ofstream convergence_file;
00229
00231     std::ofstream iterations_log_file;
00232
00234     std::ofstream point_probe_log_file;
00235
00236     // ---- Mesh parameters -------------------------------------------------
00237
00239     std::pair<unsigned int, unsigned int> N_el;
00240
00242     const std::pair<Point<dim>, Point<dim» geometry;
00243
00245     const unsigned int r;
00246
00247     // ---- Time parameters -------------------------------------------------
00248
00250     const double T;
00251
00253     const double delta_t;
00254
```

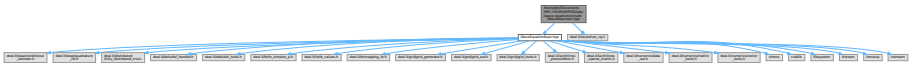
```
00256     double time = 0.0;
00257
00259     unsigned int timestep_number = 0;
00260
00261     // ---- Monitoring variables ------------------------------------------
00262
00264     double current_energy;
00265
00267     unsigned int error_sample_count = 0;
00268
00270     double simulation_time = 0.0;
00271
00272     // ---- Problem data (functions) --------------------------------------
00273
00275     const Function<dim>& c;
00276
00278     Function<dim>& f;
00279
00281     const Function<dim>& u0;
00282
00284     const Function<dim>& v0;
00285
00287     Function<dim>& g;
00288
00290     Function<dim>& dgdt;
00291
00292     // ---- Logging parameters --------------------------------------------
00293
00295     const unsigned int log_every;
00296
00298     const unsigned int print_every;
00299
00300     // ---- Exact solution ------------------------------------------------
00301
00303     Function<dim>* exact_solution;
00304
00305     // ---- MPI -----------------------------------------------------------
00306
00308     const unsigned int mpi_size;
00309
00311     const unsigned int mpi_rank;
00312
00313     // ---- Mesh & FE objects ---------------------------------------------
00314
00316     parallel::fullydistributed::Triangulation<dim> mesh;
00317
00319     std::unique_ptr<FiniteElement<dim» fe;
00320
00322     std::unique_ptr<Quadrature<dim» quadrature;
00323
00325     DoFHandler<dim> dof_handler;
00326
00327     // ---- Matrices (common to both methods) -----------------------------
00328
00330     TrilinosWrappers::SparseMatrix mass_matrix;
00331
00333     TrilinosWrappers::SparseMatrix stiffness_matrix;
00334
00335     // ---- Solution vectors ----------------------------------------------
00336
00338     TrilinosWrappers::MPI::Vector solution_u;
00339
00341     TrilinosWrappers::MPI::Vector solution_v;
00342
00344     TrilinosWrappers::MPI::Vector old_solution_u;
00345
00347     TrilinosWrappers::MPI::Vector old_solution_v;
00348
00350     TrilinosWrappers::MPI::Vector system_rhs;
00351
00352     // ---- Output --------------------------------------------------------
00353
00355     ConditionalOStream pcout;
00356 };
00357
00368 std::string clean_double(double x, int precision = 6);
00369
00370 #endif
```

6.5 /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveNewmark.hpp File Reference

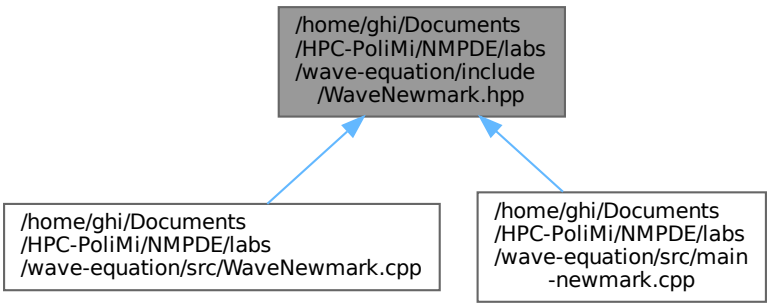
Wave-equation solver using the Newmark-beta time-integration scheme.

```
#include "WaveEquationBase.hpp"
#include <deal.II/lac/solver_cg.h>
```

Include dependency graph for WaveNewmark.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class WaveNewmark

  Concrete wave-equation solver with the Newmark- β method.

6.5.1 Detailed Description

Wave-equation solver using the Newmark-beta time-integration scheme.
Implements the classical Newmark family of methods parameterised by γ and β. The unknowns are displacement u, velocity v and acceleration a. At each time step one symmetric positive-definite system for a^{n+1} is solved with CG + AMG, then u and v are updated algebraically.

6.6 WaveNewmark.hpp

Go to the documentation of this file.

```
00001
00012 #ifndef WAVE_NEWMARK_HPP
00013 #define WAVE_NEWMARK_HPP
00014
00015 #include "WaveEquationBase.hpp"
00016 #include <deal.II/lac/solver_cg.h>
00017
00018 using namespace dealii;
00019
00038 class WaveNewmark : public WaveEquationBase
00039 {
00040   public:
00062     WaveNewmark(
00063         const std::string& problem_name_,
00064         const std::pair<unsigned int, unsigned int>& N_el_,
00065         const std::pair<Point<dim>, Point<dim»& geometry_,
00066         const unsigned int& r_,
00067         const double& T_,
00068         const double& gamma_,
00069         const double& beta_,
00070         const double& delta_t_,
00071         const Function<dim>& c_,
00072         Function<dim>& f_,
```

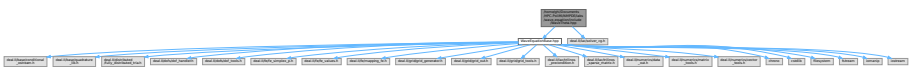
```
00073         const Function<dim>& u0_,
00074         const Function<dim>& v0_,
00075         Function<dim>& g_,
00076         Function<dim>& dgdt_,
00077         const unsigned int log_every_ = 10,
00078         const unsigned int print_every_ = 10,
00079         Function<dim>* exact_solution_ = nullptr)
00080         : WaveEquationBase(problem_name_, N_el_, geometry_, r_, T_, delta_t_,
00081                            c_, f_, u0_, v0_, g_, dgdt_, log_every_, print_every_, exact_solution_),
00082           gamma(gamma_), beta(beta_)
00083     {
00084     }
00085
00093     void run() override;
00094
00095   protected:
00096     // ---- Newmark-specific methods ----------------------------------------
00097
00099     void setup();
00100
00105     void assemble_matrices();
00106
00113     void assemble_rhs();
00114
00121     void solve_a();
00122
00129     void update_u_v();
00130
00135     void apply_dirichlet_bc();
00136
00137     // ---- Newmark parameters ----------------------------------------------
00138
00140     const double gamma;
00141
00143     const double beta;
00144
00145     // ---- Additional vectors for acceleration -----------------------------
00146
00148     TrilinosWrappers::MPI::Vector solution_a;
00149
00151     TrilinosWrappers::MPI::Vector old_solution_a;
00152
00153     // ---- System matrices -------------------------------------------------
00154
00160     TrilinosWrappers::SparseMatrix matrix_a;
00161
00168     TrilinosWrappers::SparseMatrix system_matrix_a;
00169
00170     // ---- Preconditioner --------------------------------------------------
00171
00173     TrilinosWrappers::PreconditionAMG preconditioner_a;
00174
00176     bool preconditioner_a_initialized = false;
00177
00179     unsigned int current_iterations;
00180 };
00181
00182 #endif
```

6.7 /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/include/WaveTheta.hpp File Reference

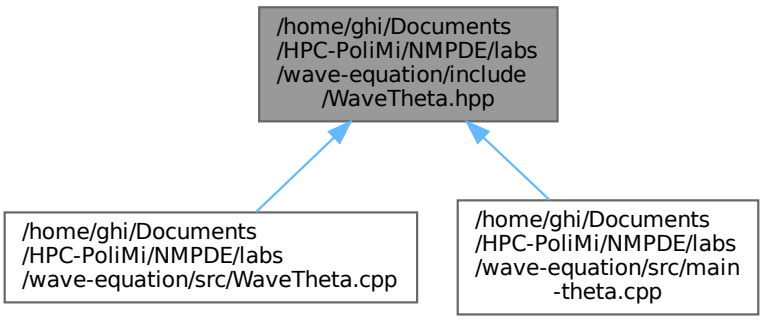
Wave-equation solver using the theta-method time discretisation.

```
#include "WaveEquationBase.hpp"
#include <deal.II/lac/solver_cg.h>
```

Include dependency graph for WaveTheta.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class WaveTheta

  Concrete wave-equation solver with the theta-method.
- class WaveSpeed< dim >

  Default wave speed: $c(\mathbf{x}) \equiv 1$.
- class InitialValuesU< dim >

  Default initial displacement: $u_0(\mathbf{x}) \equiv 0$.
- class InitialValuesV< dim >

  Default initial velocity: $v_0(\mathbf{x}) \equiv 0$.
- class RightHandSide< dim >

  Default forcing term: $f(\mathbf{x}, t) \equiv 0$.
- class BoundaryValuesU< dim >

  Default Dirichlet boundary data for u.
- class BoundaryValuesV< dim >

  Default Dirichlet boundary data for $v = \partial_t u$.

6.7.1 Detailed Description

Wave-equation solver using the theta-method time discretisation.
Implements a second-order-in-time theta-method where the unknowns are displacement u and velocity v. Two symmetric positive-definite linear systems (one for u^{n+1}, one for v^{n+1}) are solved at each time step with CG + AMG.
This file also contains simple template helper classes for equation data (wave speed, initial conditions, forcing, boundary values) that can be used for quick stand-alone tests without a parameter file.

6.8 WaveTheta.hpp

Go to the documentation of this file.

```
00001
00015 #ifndef WAVE_THETA_HPP
00016 #define WAVE_THETA_HPP
00017
00018 #include "WaveEquationBase.hpp"
00019 #include <deal.II/lac/solver_cg.h>
00020
00021 using namespace dealii;
00022
00046 class WaveTheta : public WaveEquationBase
00047 {
```

```
00048   public:
00069     WaveTheta(const std::string& problem_name_,
00070               const std::pair<unsigned int, unsigned int>& N_el_,
00071               const std::pair<Point<dim>, Point<dim»& geometry_,
00072               const unsigned int& r_,
00073               const double& T_,
00074               const double& theta_,
00075               const double& delta_t_,
00076               const Function<dim>& c_,
00077               Function<dim>& f_,
00078               const Function<dim>& u0_,
00079               const Function<dim>& v0_,
00080               Function<dim>& g_,
00081               Function<dim>& dgdt_,
00082               const unsigned int log_every_ = 10,
00083               const unsigned int print_every_ = 10,
00084               Function<dim>* exact_solution_ = nullptr)
00085         : WaveEquationBase(problem_name_, N_el_, geometry_, r_, T_, delta_t_,
00086                            c_, f_, u0_, v0_, g_, dgdt_, log_every_, print_every_, exact_solution_),
00087           theta(theta_)
00088     {
00089     }
00090
00097     void run() override;
00098
00099   protected:
00100     // ---- Theta-specific methods ------------------------------------------
00101
00103     void setup();
00104
00112     void assemble_matrices();
00113
00120     void assemble_rhs_u();
00121
00127     void assemble_rhs_v();
00128
00135     void solve_u();
00136
00143     void solve_v();
00144
00145     // ---- Theta parameter -------------------------------------------------
00146
00148     const double theta;
00149
00150     // ---- System matrices -------------------------------------------------
00151
00156     TrilinosWrappers::SparseMatrix matrix_u;
00157
00161     TrilinosWrappers::SparseMatrix matrix_v;
00162
00168     TrilinosWrappers::SparseMatrix system_matrix_u;
00169
00173     TrilinosWrappers::SparseMatrix system_matrix_v;
00174
00175     // ---- Preconditioners -------------------------------------------------
00176
00178     TrilinosWrappers::PreconditionAMG preconditioner_u;
00179
00181     TrilinosWrappers::PreconditionAMG preconditioner_v;
00182
00184     bool preconditioner_u_initialized = false;
00185
00187     bool preconditioner_v_initialized = false;
00188
00190     unsigned int current_iterations_u;
00191
00193     unsigned int current_iterations_v;
00194 };
00195
00196 // ============================================================================
00197 // Default equation data (stand-alone test helpers)
00198 // ============================================================================
00199
00204 template <int dim>
00205 class WaveSpeed : public Function<dim>
00206 {
00207   public:
00208     virtual double value(const Point<dim>& /*p*/,
00209                          const unsigned int /*component*/ = 0) const override
00210     {
00211         return 1.0;
00212     }
00213 };
00214
00219 template <int dim>
00220 class InitialValuesU : public Function<dim>
00221 {
```

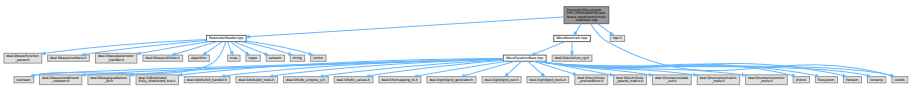
```
00222   public:
00223     virtual double value(const Point<dim>& /*p*/,
00224                          const unsigned int component = 0) const override
00225     {
00226         (void)component;
00227         Assert(component == 0, ExcIndexRange(component, 0, 1));
00228         return 0;
00229     }
00230 };
00231
00236 template <int dim>
00237 class InitialValuesV : public Function<dim>
00238 {
00239   public:
00240     virtual double value(const Point<dim>& /*p*/,
00241                          const unsigned int component = 0) const override
00242     {
00243         (void)component;
00244         Assert(component == 0, ExcIndexRange(component, 0, 1));
00245         return 0;
00246     }
00247 };
00248
00253 template <int dim>
00254 class RightHandSide : public Function<dim>
00255 {
00256   public:
00257     virtual double value(const Point<dim>& /*p*/,
00258                          const unsigned int component = 0) const override
00259     {
00260         (void)component;
00261         Assert(component == 0, ExcIndexRange(component, 0, 1));
00262         return 0;
00263     }
00264 };
00265
00275 template <int dim>
00276 class BoundaryValuesU : public Function<dim>
00277 {
00278   public:
00279     virtual double value(const Point<dim>& p,
00280                          const unsigned int component = 0) const override
00281     {
00282         (void)component;
00283         Assert(component == 0, ExcIndexRange(component, 0, 1));
00284
00285         if ((this->get_time() <= 0.5) && (p[0] < 0.5) && (p[1] > 1. / 3) &&
00286             (p[1] < 2. / 3))
00287             return std::sin(this->get_time() * 4 * numbers::PI);
00288         else
00289             return 0;
00290     }
00291 };
00292
00301 template <int dim>
00302 class BoundaryValuesV : public Function<dim>
00303 {
00304   public:
00305     virtual double value(const Point<dim>& p,
00306                          const unsigned int component = 0) const override
00307     {
00308         (void)component;
00309         Assert(component == 0, ExcIndexRange(component, 0, 1));
00310
00311         if ((this->get_time() <= 0.5) && (p[0] < 0.5) && (p[1] > 1. / 3) &&
00312             (p[1] < 2. / 3))
00313             return (std::cos(this->get_time() * 4 * numbers::PI) * 4 * numbers::PI);
00314         else
00315             return 0;
00316     }
00317 };
00318
00319 #endif
```

6.9 /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/main-newmark.cpp File Reference

Entry point for the Newmark wave-equation solver.

```
#include "ParameterReader.hpp"
#include "WaveNewmark.hpp"
#include <cstdlib>
```

```
#include <mpi.h>
```
Include dependency graph for main-newmark.cpp:



Functions

- int **main** (int argc, char ∗argv[ ])

6.9.1 Detailed Description

Entry point for the Newmark wave-equation solver.
Parses a parameter file (JSON or PRM), initialises the function data (wave speed, forcing, initial / boundary conditions, optional exact solution) via ParameterReader, constructs a WaveNewmark instance and runs the simulation. MPI is initialised and finalised automatically through deal.II's MPI_InitFinalize helper.

Usage

```
mpirun -np <N>  ./main-newmark  [path/to/parameters.json]
```

If no parameter file is given the default ../parameters/sine-membrane.json is used.

6.10 /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/main-theta.cpp File Reference

Entry point for the theta-method wave-equation solver.
```
#include "ParameterReader.hpp"
#include "WaveTheta.hpp"
#include <cstdlib>
```
Include dependency graph for main-theta.cpp:

Functions

- int **main** (int argc, char ∗argv[ ])

6.10.1 Detailed Description

Entry point for the theta-method wave-equation solver.
Parses a parameter file (JSON or PRM), initialises the function data (wave speed, forcing, initial / boundary conditions, optional exact solution) via ParameterReader, constructs a WaveTheta instance and runs the simulation. MPI is initialised and finalised automatically through deal.II's MPI_InitFinalize helper.

Usage

```
mpirun -np <N>  ./main-theta  [path/to/parameters.json]
```

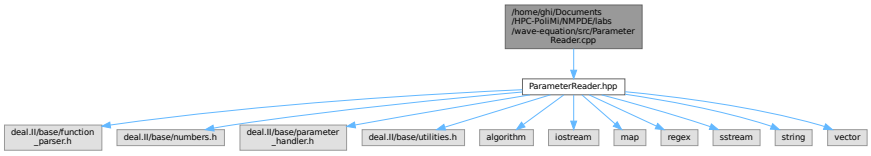
If no parameter file is given the default ../parameters/sine-membrane.json is used.

6.11 /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/↩ ParameterReader.cpp File Reference

Implementation of the ParameterReader class and helper parsing utilities.

```
#include "ParameterReader.hpp"
```

Include dependency graph for ParameterReader.cpp:



Functions

- double parse_value_with_pi (std::string value)

  Parse a numeric string that may contain the symbol "pi".
- std::map< std::string, double > parse_constants_with_pi_and_multiplication (const std::string &s)

  Parse a comma-separated "key=value" constant list.

6.11.1 Detailed Description

Implementation of the ParameterReader class and helper parsing utilities.
Handles declaration, parsing and loading of simulation parameters and function expressions from JSON / PRM files. Two free-standing helpers (parse_value_with_pi, parse_constants_with_pi_and_multiplication) support the use of symbolic "pi" constants inside parameter files.

6.11.2 Function Documentation

6.11.2.1 parse_constants_with_pi_and_multiplication()

```
std::map< std::string, double > parse_constants_with_pi_and_multiplication (
            const std::string & s )
```

Parse a comma-separated "key=value" constant list.
Values may use the "pi" symbol (see parse_value_with_pi()).

Parameters

s	The raw constant string from the parameter file.

Returns

Map of constant names to numeric values.

6.11.2.2 parse_value_with_pi()

```
double parse_value_with_pi (
            std::string value )
```

Parse a numeric string that may contain the symbol "pi".
Parse a numeric string that may contain the symbol "pi".
Recognised forms: `"pi"`, `"3.0*pi"`, or a plain numeric literal.

Parameters

value	The string to parse.

Returns

Numeric value.

Recognised forms: `"pi"`, `"3.0*pi"`, or a plain numeric literal.

Parameters

value	The string to parse.

Returns

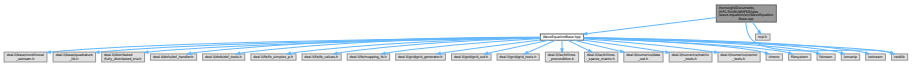
Numeric value.

6.12 /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/↩ WaveEquationBase.cpp File Reference

Implementation of the WaveEquationBase common infrastructure.

```
#include "WaveEquationBase.hpp"
#include <cstdlib>
#include <mpi.h>
```

Include dependency graph for WaveEquationBase.cpp:



Functions

- std::string clean_double (double x, int precision)

  Convert a double to a filesystem-safe string.

6.12.1 Detailed Description

Implementation of the WaveEquationBase common infrastructure.
Contains mesh creation, FE setup, DoF distribution, energy / error computation, logging helpers, VTU output and the clean_double utility.

6.12.2 Function Documentation

6.12.2.1 clean_double()

```
std::string clean_double (
            double x,
            int precision = 6 )
```

Convert a double to a filesystem-safe string.
Trailing zeros and decimal points are removed; the dot is replaced by an underscore so the result can be used in file/folder names.

Parameters

x	The value to convert.
precision	Number of decimal digits (default 6).

Returns

Sanitised string representation (e.g. 0.25 → "0_25").

6.13 /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/WaveNewmark.cpp File Reference

Implementation of the WaveNewmark solver.

```
#include "WaveNewmark.hpp"
```

Include dependency graph for WaveNewmark.cpp:

6.13.1 Detailed Description

Implementation of the WaveNewmark solver.
Contains the setup, matrix assembly, right-hand side assembly, linear solve for the acceleration, Newmark update and the main time-stepping loop including consistent initial-acceleration computation.

6.14 /home/ghi/Documents/HPC-PoliMi/NMPDE/labs/wave-equation/src/WaveTheta.cpp File Reference

Implementation of the WaveTheta solver.

```
#include "WaveTheta.hpp"
```

Include dependency graph for WaveTheta.cpp:

6.14.1 Detailed Description

Implementation of the WaveTheta solver.
Contains the setup, matrix assembly, right-hand side assembly for both displacement and velocity systems, the two linear solves and the main time-stepping loop.