User Guide

Time-Harmonic Wave Modeling and Inversion using Hybridizable Discontinuous Galerkin Discretization

HAWEN version 1.3.0

December, 2023

Abstract

This document is the User Guide for the software HAWEN (Time-Harmonic Wave Modeling and Inversion using Hybridizable Discontinuous Galerkin Discretization) currently in its version 1.3.0. It is available at https://gitlab.com/ffaucher/hawen. One should refer to the following to acknowledge use of the software (see [14]):

> F. Faucher. hawen: time-harmonic wave modeling and inversion using hybridizable discontinuous Galerkin discretization, Journal of Open Source Software, 6 (2021). (https://doi.org/10.21105/joss.02699).

Software HAWEN can currently be used to solve the following problems:

- Forward and inverse problem for visco-acoustic wave propagation in two and three dimensions, corresponding to the Euler's equations.
- Forward and inverse problem for visco-elastic wave propagation in two and three dimensions for isotropic media.
- Forward and inverse problem for the modal scalar waves in helioseismology under spherical symmetry, that is, for one-dimensional ODE.
- Forward and inverse problem for the scalar waves in helioseismology in two and three dimensions.

One should follow these steps in order to use the software.

1. Compilation of the software, see Section 3.
2. Create your *parameter file*, appropriate to the problem you want to solve, see Section 5.
3. Run the software, it supports mpi and OpenMP parallelism and can be used on clusters.
4. Visualize your results (modeling or parameter reconstruction).

Website: https://ffaucher.gitlab.io/hawen-website/. It contains some additional information and illustrations, as well as a step-by-step tutorial for inverse problem.

Acknowledgments: The code is maintained by Florian Faucher, now in the project-team Makutu of INRIA Bordeaux Sud-Ouest, and University of Pau and Pays de l'Adour, formerly at the Faculty of Mathematics, University of Vienna, Austria funded by the Austrian Science Fund (FWF), under the Lise-Meitner fellowship M 2791-N. Contact information for any comments, suggestions or questions: florian.faucher@inria.fr.

Contents

1 Introduction

This software is designed to solve time-harmonic wave problems in different media. The two classes of problems solved are the *modeling* (the propagation of waves in a given medium) and the inversion (the quantitative identification of the medium's parameters), where we rely on an iterative minimization method. The following problem are currently available,

- acoustic isotropic medium,
- elastic isotropic medium,
- modal scalar wave problem in helioseismology under spherical symmetry,
- scalar wave problem in helioseismology.

The code uses the Hybridizable Discontinuous Galerkin discretization method, relying on unstructured domain meshing. It is written in `Fortran90` and combines `mpi` and `OpenMP` parallelism for efficiency. The code is open-source and available through gitlab at `https://gitlab.com/ffaucher/hawen`.

The software runs using a *parameter file* as input, where all the details on the problem considered are given, e.g., the dimension, the physical parameters to be used, where to find the mesh of the domain, etc. The parameter file is detailed in Section 5, the list of keywords can also be found in the Index section at the end of the document.

Documents for illustrations The software should be referred to using [14]. To illustrate the software range of applications, we further mention the following works, that have been carried out using the code.

- [28], forward problem for linear anisoptropic elastic wave propagation.
- [12], forward problem for visco-elastic wave propagation with laboratory-scale experiments.
- [19], forward and inverse problem for visco-acoustic wave propagation with attenuation model uncertainty.
- [18, 17], forward and inverse problem for the Euler's equations.
- [15], inverse problem for the Euler's equations using the reciprocity-gap misfit functional.
- [8, 7], for the scalar wave equation in helioseismology under spherical symmetry, computations of the Green's functions and helioseismic products (power spectrum).
- [16], inverse problem for elastic media using the reciprocity-gap misfit functional. The benchmark to reproduce the results of [16] is available at `http://phaidra.univie.ac.at/o:1079829`.

1.1 License

The software is distributed without warranty, under the GNU General Public License v3.0, see `https://www.gnu.org/licenses/gpl-3.0.en.html`.

1.2 Changelog

* **(2023/12)** `version 1.3.0`, main changes:
  - The 2D and 3D quadrature formulas have been updated and improved for accuracy, especially concerning high-order polynomials.
  - The elastic isotropic propagator has been reworked. It is now in terms of the displacement field, and uses the compliance tensor.
  - The scalar problem in helioseismology under spherical symmetry has been renamed.
* **(2023/01)** `version 1.2.2`, main changes:
  - Improve compilation using `CMake` and set up validation tests,
* **(2022/11)** `version 1.2.1`, main changes:
  - Enable compilation using `CMake`,
  - Enable usage of GPU with MUMPS linear system operations.
* **(2022/03)** `version 1.2.0`, main changes:
  - Add two new visco-acoustic models,
  - Enable quadruple precision for the polynomials,
  - Allow for delta-Dirac sources involving derivatives.
* **(2021/07)** `version 1.1.0`, main changes:
  - Enable model representation using Lagrange basis functions: one set of polynomial functions per cell.
  - Enable inversion of Lagrange basis model representation.
  - Enable circular restriction in inversion (gradient and data-sets).
* **(2021/01)** `version 1.0.0`, main changes:
  - Implement several visco-acoustic models and allow for their inversion.
  - Modification of the 3D absorbing boundary condition for elastic propagation.
* **(2020/08)** `version 0.1.2`, main changes:
  - Modification of the formulations of the Maxwell and Zener viscoelastic models.
* **(2020/07)** `version 0.1.1`, main changes:
  - Implementation of the Maxwell and Fractional viscoelastic models.
* **(2020/06)** `version 0.1.0`, main changes:
  - Implement the scalar wave problem using the conjugated equation in helioseismology: propagator `helio_scalar-conjugated`.
  - Allow to solve the eigen-problem that is, find the eigenvalues and eigenvectors of the discretization matrix.
  - Enrich the viscoelastic problem with the Zener or the Kelvin–Voigt models.
  - Guarantee the symmetry of the matrix in all cases.
  - New features: allow for `PGI` compilation; allow for a different low-rank tolerance for linesearch operations.
* **(2020/05)** first available version of the code.

1.3 Contributors and Acknowledgments

- Starting September 2021, Florian Faucher is a researcher in the team Makutu of Inria Bordeaux, University of Pau and Pays de l'Adour, (florian.faucher@inria.fr).
- 2024–2029: Florian Faucher was awarded an ERC Starting Grant, which contributes to the software evolution.
- November 2022: Marc Fuentes (Inria Bordeaux) developed the `CMake` installation procedure for `hawen`.
- From October 2019 to September 2021, The code was developed by Florian Faucher at the Faculty of Mathematics, University of Vienna, Austria; funded by the Austrian Science Fund (FWF), under the Lise-Meitner fellowship M 2791-N.
- For any questions whose answers cannot be found from this user guide, or if you wish to participate in the software development, please contact florian.faucher@inria.fr, indicating "hawen" in the email subject.

2 Wave problems considered

The code is designed to solve the time-harmonic wave propagation problems that we depict in this section. For each of the equations, we give the PDE and the list of expected models (input) and solutions (output), as well as the possibilities regarding boundary conditions, attenuation model, etc. Here, we consider the propagation in a domain Ω, which is discretized by the input mesh, and refer to Section 5 for the complete details regarding the parameter files formalism.

As we work with HDG discretization, we consider the *first-order* problems. Nonetheless, we remind that such a discretization has several benefits regarding the size of the global linear systems, and refer the readers to, e.g., [18] for more details and references.

For generality in the frequency domain, we work with the complex frequency $\tilde{\omega}$, given by

$$\tilde{\omega} = \mathrm{i}\,\omega - \mathfrak{s}\,, \qquad \text{with } \omega \text{ and } \mathfrak{s} \text{ in } \mathbb{R}, \tag{2.1}$$

where ω refers to the usual angular frequency (Fourier transform) such that $\omega = 2\pi\,\mathrm{f}$ with f in Hz), while $\mathfrak{s}$ is a damping coefficient (in s^{-1}) that relates to Laplace's transform. The case $\mathfrak{s} \neq 0$ is in particular used for the complex-frequency inversion, or to include a constant attenuation in the propagation. In the following, by setting $\mathfrak{s} = 0$, one retrieves the usual frequency-domain formulations.

In the next subsections, we review the wave problems that are currently implemented, and list their specificities regarding the parameter files, including:

- The specific keywords regarding the physical parameters that are involved (velocity, density, etc.).
- The model of attenuation currently implemented with the equation.
- The keywords associated with the solutions of the equation (pressure, stress, etc.).
- The meaning of the boundary conditions in terms of the solutions to the equation. We allow for Dirichlet, Neumann, absorbing, free surface and wall surface boundary conditions.

2.1 Acoustic wave problem: `helmholtz_acoustic-iso`

Here we consider the acoustic-wave problem written at first-order, [18, 19], also referred to as Bergmann's equation [26]. The medium is characterized by two physical properties: the density ρ and the wave speed v_p, and the waves consist in the scalar pressure field and velocity vector field, respectively p and $\boldsymbol{v}$. Note that the bulk modulus κ can also be used to characterize the medium, it is related with the density and wave speed as follow: $v_p = \sqrt{\kappa/\rho}$.

Given the medium parameters (v_p, ρ), the complex frequency $\tilde{\omega}$ and the sources (f_p, $\boldsymbol{f}_\mathbf{v}$), find ($p$, $\boldsymbol{v}$) that solve the Euler's equation,

$$\begin{cases} -\tilde{\omega}\,\rho(\mathbf{x})\boldsymbol{v}(\mathbf{x}) + \nabla p(\mathbf{x}) = \boldsymbol{f}_\mathbf{v}(\mathbf{x})\,, & \text{(2.2a)} \\ -\dfrac{\tilde{\omega}}{\rho(\mathbf{x})\,v_p(\mathbf{x})^2}\,p(\mathbf{x}) + \nabla\cdot\boldsymbol{v}(\mathbf{x}) = f_\mathrm{p}(\mathbf{x})\,. & \text{(2.2b)} \end{cases}$$

`implementation in dimension`: 1, 2 and 3.
`implementation of the forward problem`: ✓.
`implementation of the inverse problem`: ✓.

Here, we work with two physical parameters: the wave-speed (velocity) v_p and the density ρ while the solution is given by the scalar pressure field p and the vectorial velocity $\boldsymbol{v}$. For the specific details regarding the implementation using HDG, including the gradient computation, we refer to [18]. We provide below the specific keywords associated with this equation.

Physical models

`vp`	velocity v_p in (2.2).
`rho`	density ρ in (2.2).

Solutions

`p`	pressure field p in (2.2).
`vx`	vectorial velocity $\boldsymbol{v}$ in the direction x.
`vy`	vectorial velocity $\boldsymbol{v}$ in the direction y, only for dimension 3.
`vz`	vectorial velocity $\boldsymbol{v}$ in the direction z, only in dimensions higher than 1.

Models of attenuation

In the acoustic settings, we allow for the different models to represent the attenuation, described by complex-valued parameters. It can be apply onto the wave-speed, or onto the bulk modulus

$$\kappa = v_p^2\,\rho\,. \tag{2.3}$$

They are given below, where the symbol † indicates the complex-valued parameter and we refer to [32] for more details.

- **The *Koslky–Futterman* model**, uses a complex-valued wavespeed defined by, [25, 20, 32],

$$v_p^\dagger = v_p\left(1 - \frac{\mathrm{i}}{2\,Q}\right), \tag{2.4}$$

  where Q is the quality factor, that accounts for the attenuation.

– **The *Kelvin–Voigt* model** applies onto the bulk modulus with

$$\kappa^\dagger \;=\; \kappa \;-\; \mathrm{i}\omega\,\kappa\,\tau_\epsilon\,. \tag{2.5}$$

– **The *Maxwell* model** writes as

$$\kappa^\dagger \;=\; \frac{-\mathrm{i}\omega\,\kappa\eta}{\kappa \;-\; \mathrm{i}\omega\,\eta}\,. \tag{2.6}$$

– **The *Cole–Cole* model** is

$$\kappa^\dagger \;=\; \kappa\,\frac{1 \;+\; (-\mathrm{i}\omega\,\tau_\epsilon)^\beta}{1 \;+\; (-\mathrm{i}\omega\,\tau_\sigma)^\beta}\,. \tag{2.7}$$

– **The *Zener* model** is

$$\kappa^\dagger \;=\; \kappa\,\frac{1 \;-\; \mathrm{i}\omega\,\tau_\epsilon}{1 \;-\; \mathrm{i}\omega\,\tau_\sigma}\,. \tag{2.8}$$

– **The *modified-Szabo* model** is

$$\kappa^\dagger \;=\; \frac{\kappa}{1 \;+\; (-\mathrm{i}\,\omega\,\tau)^{\beta-1}}\,. \tag{2.9}$$

– **The *Kowar–Scherzer–Bonnefond* model (KSB)** is

$$\kappa^\dagger \;=\; \frac{\kappa}{\left(1 \;+\; \dfrac{\eta}{\sqrt{1 \;+\; (-\mathrm{i}\,\omega\tau)^\beta}}\right)^2}\,. \tag{2.10}$$

In the case of attenuation, we need the additional medium parameters:

`viscosity_model`	The choices are currently `Kolsky-Futterman` for (2.4) `Kelvin-Voigt` for (2.5), `Zener-model_simple` for (2.8), `Maxwell` for (2.6) and `Cole-Cole` for (2.7)
`freq-power`	Frequency power β in the case of viscoacoustic Cole–Cole model, see (2.7). This is only if `viscosity_model=Cole-Cole`.
`tau-sigma`	Attenuation model τ_σ in the case of viscoacoustic Zener model, see (2.8) and Cole–Cole model, see (2.7).
`tau-eps`	Attenuation model τ_ϵ in the case of viscoacoustic Zener model, see (2.8), Kelvin–Voigt, see (2.5), and Cole–Cole model, see (2.7).
`visco`	Attenuation model η in the Maxwell model, see (2.6), or Q in the case of the Kolsky–Futterman model, see (2.4).

Boundary conditions

Here, we list the choice of boundary conditions, and there meaning in terms of the solutions, assuming the conditions is imposed on a boundary Γ.

`dirichlet`	Dirichlet boundary condition in pressure: $p\,\vert_\Gamma = 0$.
`free-surface`	In acoustic, it is the Dirichlet boundary condition, imposing $p\,\vert_\Gamma = 0$.
`neumann`	Neumann boundary condition in pressure: $\partial_n p\,\vert_\Gamma = 0$.
`wall-surface`	In acoustic, it is the Neumann boundary condition, imposing $\partial_n p\,\vert_\Gamma = 0$.

`absorbing` It corresponds to the Robin boundary condition $\partial_n p = \mathtt{Z}\, p$, with the choice of `Z` given below.

`bc_impedance` Indicates the choice of `Z` for the absorbing condition. In acoustic, it can only be `Sommerfeld_0`, which amounts to $\mathtt{Z} = \tilde{\omega} c_p^{-1}$.

Inversion

In the context of inversion, one has to select the physical parameters which are inverted. In acoustic, two must be chosen among,

`velocity` The reconstruction is carried out with respect to the wave speed c_p.

`rho` The reconstruction is carried out with respect to the density ρ.

`bulk-modulus` The reconstruction is carried out with respect to the bulk modulus $\kappa = \rho\, c_p^2$.

`impedance` The reconstruction is carried out with respect to the impedance $I = \rho\, c_p$.

2.2 Elastic isotropic wave problem: `helmholtz_elastic-iso_Su`

Since version `v1.3.0`, the elastic isotropic wave problem is now written in terms of the displacement field $\boldsymbol{u}$ instead of the velocity field. We refer to [28] for more details.

Given the medium parameters ($v_p := \sqrt{(\lambda + 2\mu)/\rho}$, $v_s := \sqrt{\mu/\rho}$, ρ), the complex frequency $\tilde{\omega}$ and the sources ($\boldsymbol{f}_{\boldsymbol{\sigma}}$, $\boldsymbol{f}_{\mathbf{u}}$), find ($\boldsymbol{\sigma}$, $\boldsymbol{u}$) that solve

$$\begin{cases} -\nabla \cdot \boldsymbol{\sigma}(\mathbf{x}) + \tilde{\omega}^2\, \rho(\mathbf{x})\, \boldsymbol{u}(\mathbf{x}) = \boldsymbol{f}_{\mathbf{u}}(\mathbf{x})\,, & \text{(2.11a)} \\ -\boldsymbol{\sigma}(\mathbf{x}) \;-\; \lambda(\mathbf{x})\, Tr(\boldsymbol{\epsilon}(\mathbf{x}))I_d \;-\; 2\mu(\mathbf{x})\, \boldsymbol{\epsilon}(\mathbf{x}) = \boldsymbol{f}_{\boldsymbol{\sigma}}(\mathbf{x})\,. & \text{(2.11b)} \end{cases}$$

`implementation in dimension`: 2 and 3.
`implementation of the forward problem`: ✓.
`implementation of the inverse problem`: ✓.

Here, Tr denotes the trace and I_d the identity matrix. In elastic isotropy, there are three medium parameters: the Lamé parameters λ and μ, and the density ρ. They are related to the P and S-wavespeeds, with

$$v_p := \sqrt{\frac{\lambda + 2\mu}{\rho}}\,, \qquad v_s := \sqrt{\frac{\mu}{\rho}}\,. \tag{2.12}$$

The stress tensor $\boldsymbol{\sigma}$ is given in terms of the strain tensor $\boldsymbol{\epsilon}$, using the Hooke's law which is simplified above due to the isotropic medium: (2.11b). Furthermore, The strain is given by

$$\boldsymbol{\epsilon} = \frac{1}{2}\Big(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T\Big)\,. \tag{2.13}$$

Physical models

`vp` P-wavespeed v_p in (2.12).

`vs` S-wavespeed v_s in (2.12).

`rho` density ρ in (2.11).

Solutions

`ux`	vectorial displacement $\boldsymbol{u}$ in the direction x.
`uy`	vectorial displacement $\boldsymbol{u}$ in the direction y, only for dimension 3.
`uz`	vectorial displacement $\boldsymbol{u}$ in the direction z, only in dimensions higher than 1.
`sxx`	Diagonal component of the stress tensor $\boldsymbol{\sigma}$.
`syy`	Diagonal component of the stress tensor $\boldsymbol{\sigma}$, only for dimension 3.
`szz`	Diagonal component of the stress tensor $\boldsymbol{\sigma}$, for dimensions higher than 1.
`sxz`	Extra-diagonal component of the stress tensor $\boldsymbol{\sigma}$, for dimensions higher than 1.
`sxy`	Extra-Diagonal component of the stress tensor $\boldsymbol{\sigma}$, only for dimension 3.
`syz`	Extra-Diagonal component of the stress tensor $\boldsymbol{\sigma}$, only for dimension 3.

Models of attenuation

In the elastic isotropic, we allow for the attenuation represented via *complex-valued Lamé parameters*. We follow [11] which describes several models. We use : to denote the double contraction and $\boldsymbol{C}$ is the elasticity tensor which under isotropy, writes as

$$\boldsymbol{C} : \boldsymbol{\epsilon} \;=\; \lambda\, Tr(\boldsymbol{\epsilon}) I_d \;-\; 2\mu\, \boldsymbol{\epsilon}\,. \tag{2.14}$$

Referring to [11, Section 2] for more details, we have currently implemented the following models for viscoelasticity:

– **The *Kelvin–Voigt* model**, [11, Section 2.4.2] gives the constitutive law,

$$\boldsymbol{\sigma}(\mathbf{x},t) \;=\; \boldsymbol{C}(\mathbf{x}) : \boldsymbol{\epsilon}(\mathbf{x},t) + \boldsymbol{\eta}(\mathbf{x}) : \partial_t \boldsymbol{\epsilon}(\mathbf{x},t)\,, \tag{2.15}$$

which amounts, for time-harmonic waves, to,

$$\boldsymbol{\sigma}(\mathbf{x},\omega) \;=\; \Big(\boldsymbol{C}(\mathbf{x}) - \mathrm{i}\omega \boldsymbol{\eta}(\mathbf{x})\Big) \boldsymbol{\epsilon}(\mathbf{x},\omega)\,. \tag{2.16}$$

– **The *Zener* model**, [11, Section 2.4.3] gives the constitutive law,

$$\boldsymbol{\sigma}(\mathbf{x},t) + \tau_\sigma(\mathbf{x})\, \partial_t \boldsymbol{\sigma}(\mathbf{x},t) \;=\; \boldsymbol{C}(\mathbf{x}) : \boldsymbol{\epsilon}(\mathbf{x},t) + \tau_\epsilon(\mathbf{x})\, \boldsymbol{\eta}(\mathbf{x}) : \partial_t \boldsymbol{\epsilon}(\mathbf{x},t) \tag{2.17}$$

which amounts, for time-harmonic waves, to,

$$\boldsymbol{\sigma}(\mathbf{x},\omega) \;=\; \left(\frac{\boldsymbol{C}(\mathbf{x}) \;-\; \mathrm{i}\omega\, \tau_\epsilon(\mathbf{x})\, \boldsymbol{\eta}(\mathbf{x})}{1 \;-\; \mathrm{i}\omega\, \tau_\sigma(\mathbf{x})} \right) : \boldsymbol{\epsilon}(\mathbf{x},\omega)\,. \tag{2.18}$$

It requires that

$$\tau_\epsilon \lambda_\eta \;\geq\; \tau_\sigma \lambda \qquad \text{and} \qquad \tau_\epsilon \mu_\eta \;\geq\; \tau_\sigma \mu\,. \tag{2.19}$$

– **The *Maxwell* model**, [11, Section 2.4.1] gives the constitutive law,

$$\boldsymbol{\eta}(\mathbf{x})^{-1} : \boldsymbol{\sigma}(\mathbf{x},t) + \boldsymbol{C}(\mathbf{x})^{-1} : \partial_t \boldsymbol{\sigma}(\mathbf{x},t) \;=\; \partial_t \boldsymbol{\epsilon}(\mathbf{x},t)\,, \tag{2.20}$$

which amounts, for time-harmonic waves, to,

$$\boldsymbol{\sigma}(\mathbf{x},\omega) \;=\; -\mathrm{i}\omega \left(\boldsymbol{\eta}(\mathbf{x})^{-1} \;-\; \mathrm{i}\omega\, \boldsymbol{C}(\mathbf{x})^{-1} \right)^{-1} : \boldsymbol{\epsilon}(\mathbf{x},\omega) \tag{2.21}$$

– **The *Fractional* model**, gives, for time-harmonic waves,

$$\boldsymbol{\sigma}(\mathbf{x},\omega) \;=\; \Big(\boldsymbol{C}(\mathbf{x}) \;+\; (-\,\mathrm{i}\omega)^{\alpha}\,\boldsymbol{\eta}(\mathbf{x})\Big) : \boldsymbol{\epsilon}(\mathbf{x},\omega)\,. \tag{2.22}$$

Therefore, the encoding of attenuation leads to the introduction of new parameters, and results in working with *complex-valued* Lamé parameters, $\lambda^{\dagger}$ and $\mu^{\dagger}$. For instance, with the Kelvin–Voigt model, we have,

$$\lambda^{\dagger} \;=\; \lambda \,-\tilde{\omega}\,\eta_{\lambda} \qquad \text{and} \quad \mu^{\dagger} \;=\; \mu \,-\tilde{\omega}\,\eta_{\mu}\,, \tag{2.23}$$

hence we need the two additional models: η_{λ} and η_{μ}. Proceeding similarly with the other models, we give below the corresponding keywords for the parameter file.

`viscosity_model`	The choices are currently `Kelvin-Voigt` for (2.16), `Zener-model_simple` for (2.18). `Maxwell` for (2.21) and `Fractional` for (2.22)
`eta-lambda`	Attenuation model η_{λ} in the case of viscoelasticity with Kelvin–Voigt, Maxwell, Fractional and Zener models.
`eta-mu`	Attenuation model η_{μ} in the case of viscoelasticity with Kelvin–Voigt, Maxwell, Fractional and Zener models.
`freq-power`	Frequency power α in the case of viscoelastic Fractional model, see (2.22). This is only if `viscosity_model=Fractional`.
`tau-sigma`	Attenuation model τ_{σ} in the case of viscoelastic Zener model, see (2.18). This is only if `viscosity_model=Zener-model_simple`.
`tau-eps`	Attenuation model τ_{ϵ} in the case of viscoelastic Zener model, see (2.18). This is only if `viscosity_model=Zener-model_simple`.

Boundary conditions

Here, we list the choice of boundary conditions, and there meaning in terms of the solutions, assuming the conditions is imposed on a boundary Γ.

`dirichlet`	Dirichlet boundary condition in velocity: $\boldsymbol{v}\,|_{\Gamma} = 0$.
`neumann`	Neumann boundary condition imposes $\boldsymbol{\sigma}\cdot n \;=\; 0$.
`free-surface`	In elastic isotropic, it corresponds to $\boldsymbol{\sigma}\cdot n \;=\; 0$, which is actually a Neumann boundary condition.
`wall-surface`	In elastic isotropic, it is the Dirichlet boundary condition, imposing $\boldsymbol{v}\,|_{\Gamma} = 0$.
`absorbing`	It corresponds to the Robin type boundary condition.
`bc_impedance`	Indicates the choice for the absorbing condition. In elastic, it can only be `Sommerfeld_0`, given by, e.g., [21, 24].

Inversion

In the context of inversion, one has to select the physical parameters which are inverted. In elastic isotropy, the following keywords are allowed.

`lambda`	The reconstruction is carried out for the first Lamé parameter λ.
`mu`	The reconstruction is carried out for the second Lamé parameter μ.

`rho`	The reconstruction is carried out with respect to the density ρ.
`eta-lambda`	For viscoelastic medium, the reconstruction is carried out with respect to the parameter η_λ. This is for the Kelvin–Voigt, Maxwell, Fractional and Zener attenuation models.
`eta-mu`	For viscoelastic medium, the reconstruction is carried out with respect to the parameter η_μ. This is for the Kelvin–Voigt, Maxwell, Fractional and Zener attenuation models.
`freq-power`	For viscoelastic medium, the reconstruction is carried out with respect to the frequency power α. This is only for the Fractional attenuation model, i.e., only if `viscosity_model=Fractional`.
`tau-sigma`	For viscoelastic medium, the reconstruction is carried out with respect to the parameter τ_σ. This is only for the Zener attenuation model, i.e., only if `viscosity_model=Zener-model_simple`.
`tau-eps`	For viscoelastic medium, the reconstruction is carried out with respect to the parameter τ_ϵ. This is only for the Zener attenuation model, i.e., only if `viscosity_model=Zener-model_simple`.

2.3 Scalar wave in helioseismology: `helio_scalar-conjugated_spherical1d`

Given the medium parameters (v_p, α, $\alpha' := \partial_x \alpha$), the complex frequency $\tilde{\omega}$, the mode ℓ and the sources (f_w, f_v), find (w, v) that solve

$$\begin{cases} -x\,(x\,v(x))' - x^2\dfrac{\tilde{\omega}^2}{v_p(x)^2} + x^2\dfrac{\alpha(x)^2}{4} + x\,\alpha(x) + \ell\,(\ell+1) = x^2\,f_v(x)\,, & \text{(2.24a)} \\ x\,w'(x) - w(x) - v(x) = x^2\,f_\mathrm{w}(x). & \text{(2.24b)} \end{cases}$$

`implementation in dimension`: 1.
`implementation of the forward problem`: ✓.
`implementation of the inverse problem`: ✓.

Here, v_p is the velocity and α is the inverse density scaled heights, with $\alpha = -\rho'/\rho$. We refer to [7, 10, 8, 9] for more details on this equation, occurring for the propagation of scalar waves in helioseismology under spherical symmetry. In particular, the precise derivation of the HDG discretization is given in [8]. This equation is in one-dimension only.

Physical models

`vp`	wavespeed v_p in (2.24).
`alpha`	inverse density scaled height α in (2.24).
`dalpha`	derivative of the inverse density scaled height α in (2.24).

Solutions

`w`	solution w in (2.24).
`v`	solution v in (2.24).

Model of attenuation

The attenuation is given by a model parameter γ, which modifies the frequency with

$$\tilde{\omega}_\dagger^2 \;=\; \tilde{\omega}^2 \;+\; 2\mathrm{i}\tilde{\omega}\gamma(x) \tag{2.25}$$

`viscosity_model` One can either give directly the value of γ in (2.25), with keyword `gamma`. Alternatively, one can use the power law, with keyword `gamma0_power-law`, where γ is computed by

$$\gamma(\omega) \;=\; 2\pi\gamma_0 \left| \frac{\omega}{2\pi\, 3\times 10^{-3}} \right|^{5.77}, \qquad \text{power-law}\,. \tag{2.26}$$

`gamma` Value of the attenuation factor γ when `viscosity_model=gamma`.

`gamma0_power-law` Value of γ_0 in (2.26) when `viscosity_model=gamma0_power-law`.

Boundary conditions

Here, we list the choice of boundary conditions, and there meaning in terms of the solutions, assuming the conditions is imposed on a boundary Γ.

`dirichlet` Dirichlet boundary condition for w: $w = 0$.

`neumann` Neumann boundary condition for w: $w' = 0$.

`free-surface` Arbitrarily chosen to correspond with the Dirichlet condition, $w = 0$.

`wall-surface` Arbitrarily chosen to correspond with the Neumann condition, $v = 0$.

`absorbing` It corresponds to the Radiation boundary conditions, cf [7, 8, 10, 9].

`bc_impedance` Indicates the choice of `Z` for the absorbing condition. These are given in [7, 8, 10, 9], with the choices `ZDtN`, `Znonlocal`, `ZSHF1b`, `ZSSAI1`, `ZAHF1`, `ZSHF0`, `ZSHF1a`, `ZSSAI0`, `ZAHF0`, `Znaive`, `ZARBC1`.

Inversion

In the context of inversion, one has to select the physical parameters which are inverted, for now, it can only be,

`velocity` The reconstruction is carried out with respect to the wave speed v_p.

2.4 Scalar wave in helioseismology: `helio_scalar-conjugated`

Given the medium parameters (v_p, α, $\alpha' := \partial_r\alpha$), with α spherically symmetric, the complex frequency $\tilde{\omega}$, and the sources (f_w, f_v), find $(w, \boldsymbol{v})$ that solve

$$\begin{cases} -\,\tilde{\omega}\,\boldsymbol{v}(\mathbf{x}) + \nabla w(\mathbf{x}) \;=\; |\mathbf{x}|\,\boldsymbol{f}_\mathbf{v}(\mathbf{x})\,, & \text{(2.27a)} \\ -\,\tilde{\omega}\,|\mathbf{x}| \left(\dfrac{\eta^2(\mathbf{x},\omega)}{v_p(\mathbf{x})^2} + \dfrac{\mathsf{q}(\mathbf{x})}{(\mathrm{i}\omega)^2} \right) w(\mathbf{x}) + |\mathbf{x}|\,\nabla\cdot\boldsymbol{v}(\mathbf{x}) \;=\; |\mathbf{x}|\,f_\mathrm{w}(\mathbf{x})\,, & \text{(2.27b)} \end{cases}$$

$$\text{with} \qquad \mathsf{q}(\mathbf{x}) \;=\; \frac{\alpha(\mathbf{x})^2}{4} + \frac{\alpha'(\mathbf{x})}{2} + \frac{\alpha(\mathbf{x})}{|\mathbf{x}|} \qquad \text{and} \qquad \eta(\mathbf{x},\omega) = \sqrt{1 + \frac{2\mathrm{i}\,\gamma(\mathbf{x})}{\omega}}\,. \tag{2.28}$$

Therefore, α is a spherical parameter, that *only* depends on $r = |x|$.

`implementation in dimension`: 2 and 3.
`implementation of the forward problem`: ✓.
`implementation of the inverse problem`: ✓.

Here, v_p is the velocity and α is the inverse density scaled heights, with $\alpha = -\rho'/\rho$. The parameter γ stands in case of attenuation only, that is, it is set to zero if a non-attenuating medium. We refer to [7, 10, 8, 9] for more details on this equation, occurring for the propagation of scalar waves in helioseismology.

Physical models

`vp`	wavespeed v_p in (2.27).
`alpha`	inverse density scaled height α in (2.28).
`dalpha`	derivative of the inverse density scaled height α in (2.28).

Solutions

`w`	solution w in (2.27).
`v`	solution $\boldsymbol{v}$ in (2.27).

Model of attenuation

The attenuation is given by a model parameter γ, which modified η in (2.27) and (2.28). In the absence of attenuation, γ is automatically set to zero such that $\eta = 1$.

`viscosity_model`	One can either give directly the value of γ, with keyword `gamma`. Alternatively, one can use the power law, with keyword `gamma0_power-law`, where γ is computed following (2.26).
`gamma`	Value of the attenuation factor γ when `viscosity_model=gamma`.
`gamma0_power-law`	Value of γ_0 in (2.26) when `viscosity_model=gamma0_power-law`.

Boundary conditions

Here, we list the choice of boundary conditions, and there meaning in terms of the solutions, assuming the conditions is imposed on a boundary Γ.

`dirichlet`	Dirichlet boundary condition for w: $w = 0$.
`neumann`	Neumann boundary condition for w: $w' = 0$.
`free-surface`	Arbitrarily chosen to correspond with the Dirichlet condition, $w = 0$.
`wall-surface`	Arbitrarily chosen to correspond with the Neumann condition, $\boldsymbol{v} = 0$.
`absorbing`	It corresponds to the Radiation boundary conditions, cf [7, 8, 10, 9], but only the ones that are not dependent on the mode are available.
`bc_impedance`	Indicates the choice of `Z` for the absorbing condition. These are given in [7, 8, 10, 9], where only the ones that are independent of the mode are available: `ZSHF0`, `ZSHF1a`, `ZSSAI0`, `ZAHF0`, `Znaive`, `ZARBC1`.

Inversion

In the context of inversion, one has to select the physical parameters which are inverted, for now, it can only be,

`velocity`	The reconstruction is carried out with respect to the wave speed v_p.

3 Installation and utilization

The code is available on gitlab and can be downloaded using a `git clone` or direct download, from https://gitlab.com/ffaucher/hawen.

3.1 Dependencies

3.1.1 Mandatory dependencies

The code requires the following libraries to be used.

- The direct solver MUMPS, [4, 5, 3], is used to solve the linear system. It is designed for the factorization of large-scaled sparse matrix, and allows for multiple right-hand sides. It is available at http://mumps.enseeiht.fr/. Two libraries of MUMPS must be installed prior to HAWEN: `libcmumps` (for simple precision complex matrices) and `libzmumps` (for double precision complex matrices). It is recommended to use MUMPS at least of version `5.3.3`.
- As part of MUMPS, LAPACK and ScaLAPACK are also required (https://www.netlib.org/scalapack/), they are also used by the code.
- The partitioner METIS is used to split the mesh among the `mpi` processors, that is, every processor retains a list of cell. METIS is available at http://glaros.dtc.umn.edu/gkhome/metis/metis/overview. METIS can be compiled using 32 or 64 bits. Depending on your compilation, one must indicate the corresponding precision (i.e., simple or double) in the `src/m_define_precision.f90` file, cf. Subsection 3.2.3.

Remark 1. *If several partitioners are linked with* MUMPS*, they also have to be linked* HAWEN*. Namely, all compilation dependencies of* MUMPS *must be included in* HAWEN*.*

3.1.2 Optional dependencies

The following libraries are optional.

- The library `arb`, [22, 23], can be linked with HAWEN. It is used to precisely compute special functions (e.g., Bessel's functions are not stable depending on the parameters). In particular, it is used to compute the exact Dirichlet-to-Neumann map condition for the propagator `helio_scalar_conjugated_spherical1d`, by the means of Whittaker's special functions, cf. [7, 10, 9, 8]. It is available at http://arblib.org/.
- The libraries ARPACK and PARPACK can be linked. They serve for the computation of eigenvectors and eigenvalues, these are not used in HAWEN for now. It is available as https://www.caam.rice.edu/software/ARPACK/.

3.2 Compilation using `Makefile`

Once the code HAWEN has been retrieved from https://gitlab.com/ffaucher/hawen, and that the dependencies have been installed, here are the steps for the compilation. You first need to go into the folder `code`:

`cd code`

The compilation needs the following files to be edited, these are further detailed in the below subsections.

- the file `make.version_config` has to be created in the `config` folder.
- the compilation options are selected in the file `config/make.version`.
- (optional) the arithmetic precision is selected in the the file `src/m_define_precision.f90`.

Once those files are properly edited, the compilation follows with command
`make`

In order to clean the installation, one has to run
`make clean`
or
`make cleanall`

Remark 2. *It is* necessary *to use*
`make clean`
in between successive compilations of different propagators. For instance in order to obtain the acoustic and elastic executables, one must compile the first version, and run ***`make clean`*** *before compiling the second version.*

3.2.1 Configuration file, `config/make.version_config`

The first step is to create the file `config/make.version_config` which contains the link towards the dependency libraries: MUMPS, METIS, LAPACK and ScaLAPACK. Templates for this file can be found in the folder: `config/config-files_make/`.

The code has been successfully tested using GNU, `intel` and `PGI` based compilers, and linking with the MKL or `OpenBLAS` libraries instead of the LAPACK and ScaLAPACK ones.

We provide below the most important variables to link

```
# F90 indicates the compiler, e.g., mpif90 or mpiifort.
F90  := mpif90
# ARCH indicates the architecture, it can either be
# GNU, INTEL or PGI
ARCH := GNU

# metis is a mandatory dependency, both the library
# and include folder must be linked.
METISDIR  = /usr/lib
LMETIS    = -L$(METISDIR) -lmetis
IMETIS    = -I/usr/include/

# MUMPS is a mandatory dependency, both the library
# -lcmumps -lzmumps -lmumps_common and -lpord must
# be linked, as well as the include folder
```

```
MUMPSDIR= /usr/lib
LMUMPS  = -L$(MUMPSDIR) -lcmumps -lzmumps
LMUMPS += -L$(MUMPSDIR) -lmumps_common -lpord
IMUMPS  = -I/usr/include/

# additional dependencies for Mumps for linear algebra
# are required with LAPACK and SCALAPACK
LMUMPS += -L/usr/lib -lblas -lscalapack-openmpi -llapack
```

We remind that any additional dependencies of MUMPS, e.g., partitioners, must also be included here.

3.2.2 Choice of problem, `config/make.version`

The choice of problem is made prior to the compilation, such that one executable is linked with one problem (forward or inverse, choice of medium). This is selected in the file `config/make.version`, together with the choice of debug mode. The main options are

- `DEBUG` and `OPTIMIZATION` to select either the compilation in debug or with optimization flags (e.g., `-O3` flag). One should be `0` and the other `1`.
- `PROBLEM` indicates the type of problem: this can be `forward` or `inversion` respectively to compile the forward or the inverse problem.
- `PROPAGATOR` indicates the type of choice of PDE: this can be `helmholtz_acoustic-iso`, `helmholtz_elastic-iso_Su`, `helio_scalar-conjugated_spherical1d`, or `helio_scalar-conjugated`, cf. Section 2.
- `MUMPS_LATEST`: should be set to `1` if you have compiled MUMPS version 5.3.3 or higher.
- `DEPENDENCY_ARB`: should be `0` if `arb` library is not linked, or `1` if it has been compiled and the proper path given in the `config/make.version_config` file.
- `DEPENDENCY_ARPACK`: should be `0` if `ARPACK` library is not linked, or `1` if it has been compiled and the proper path given in the `config/make.version_config` file.
- `DEPENDENCY_PARPACK`: should be `0` if `PARPACK` library is not linked, or `1` if it has been compiled and the proper path given in the `config/make.version_config` file.

The main compilation options are selected in the file `config/make.version`.

```
DEBUG           :=0
OPTIMIZATION    :=1

## problem can be "forward" or "inversion"
PROBLEM        := forward
## list of propagators:
##  (o) helmholtz_acoustic-iso
##  (o) helmholtz_elastic-iso_Su
##  (o) helio_scalar-conjugated_spherical1d
PROPAGATOR     := helmholtz_acoustic-iso

# DEPENDENCIES
```

```
MUMPS_LATEST=0
DEPENDENCY_ARB=0
DEPENDENCY_ARPACK=0
DEPENDENCY_PARPACK=0
```

3.2.3 Precision file, `src/m_define_precision.f90`

One can choose the arithmetic precision (simple or double precision) for the different objects. This is controlled in the file `src/m_define_precision.f90`, which can be modified prior to the compilation. This file does not need to be modified if you are not sure of what to choose.

The precision is selected by an integer, which must be either `4` (simple precision) or `8` (double precision). The choice concerns the following:

- `RKIND_POL`: precision for the polynomials (recommended `8`).
- `IKIND_MESH`: size for the integers related to the mesh (e.g., the number of cells), *only `4` is supported for now.*
- `RKIND_MESH`: size for the reals related to the mesh (e.g., the nodes coordinates), recommended `8`.
- `IKIND_METIS`: size of the integers at which `METIS` has been compiled.
- `RKIND_METIS`: size of the reals at which `METIS` has been compiled.
- `IKIND_MAT` and `RKIND_MAT`: precision for the matrix operations. They coincide with the precision of `MUMPS` and must be the same, recommended `8`.

We provide here below the template with the default values.

Choose the arithmetic precision with the file `m_define_precision.f90`

```
!> for polynomials computation, should be 8 (double)
integer, parameter :: RKIND_POL = 8

!> mesh informations: should be 4 or 8 and
integer, parameter :: IKIND_MESH = 4
integer, parameter :: RKIND_MESH = 8

!> this is to chose depending on the compilation of
!> Metis: 4 if simple precision (32 bits)
!>        8 if double precision (64 bits)
!> careful that even if compiled with int 64 bits,
!> the size of reals is usually 32 bits nonetheless.
integer, parameter :: IKIND_METIS = 4
integer, parameter :: RKIND_METIS = 4

!> matrix informations: should be 4 or 8
integer, parameter :: IKIND_MAT  = 8
integer, parameter :: RKIND_MAT  = IKIND_MAT
```

3.3 Compilation using CMake

It is possible to compile the code using CMake. It is however an experimental implemtation, please send any comments, questions and remarks to florian.faucher@inria.fr in case of complications and for potential improvements.

After the code dependencies are installed, compiling with CMake is done in two step. First to retrieve the links to the dependencies with
`cmake -B build`
performed in the root directory. Here it means that all installation file needed by CMake are generated in folder build/. By default, the propagator compiled is helmholtz_acoustic-iso and the problem is forward. These can be changed using -D options of CMake, e.g., run
`cmake -B build -DPROPAGATOR=helmholtz_elastic-iso_Su -DPROBLEM=inversion`
to compile the inversion code for the helmholtz_elastic-iso_Su propgator.

One can also specify the path to the dependencies in the case they have been manually installed. For instance using a file preload.cmake to indicate the path (see below) and then by running `cmake -B build -C preload.cmake`
An example of a preload file can be found in cmake/preload.cmake.Template

Illustration of a file preload.cmake file to link towards manually installed libraries, such as MUMPS and ParMETIS.

```
# choice of arithmetic precision
set(IKIND_METIS 8 CACHE STRING "metis int. size") # double precision
set(RKIND_METIS 4 CACHE STRING "metis real size") # simple precision
set(IKIND_MAT   4 CACHE STRING "mumps real size") # simple precision

# choice of problem: forward or inversion
set(PROBLEM     forward                CACHE STRING "")

# choice of propagator
set(PROPAGATOR  helmholtz_acoustic-iso CACHE STRING "")


# paths to mumps/metis/scotch
set(MUMPS_PREFIX    /your/path/to/MUMPS)
set(MUMPS_ROOT       ${MUMPS_PREFIX}                CACHE STRING "")

set(METIS_PREFIX    /your/path/to/parmetis)
set(PARMETIS_LIB_DIR ${METIS_PREFIX}                CACHE STRING "")
set(PARMETIS_INC_DIR ${METIS_PREFIX}/include        CACHE STRING "")
set(METIS_LIB_DIR    ${METIS_PREFIX}                CACHE STRING "")
set(METIS_INC_DIR    ${METIS_PREFIX}/metis/include  CACHE STRING "")

set(SCOTCH_PREFIX   /your/path/to/scotch)
set(SCOTCH_LIB_DIR   ${SCOTCH_PREFIX}               CACHE STRING "")
```

Once the command `CMake -B build` has been successfully run (with choices of options), the compilation can be done in parallel, e.g., using 5 cores:
`make -C build -j5`
The executable files are generated in folder `build/bin/`. In addition, the installation can verified using validation tests by running `ctest` in the CMake installation folder:
`cd build/`

`ctest`

3.4 Launching the code

The code supports `mpi` and `OpenMP` parallelism The number of `mpi` and threads is chosen by the user when launching the code. Assuming the parameter file is correct (see Section 5) and that the compilation is successful, one can use, for example, for elastic isotropic wave propagation with HDG discretization:

`export OMP_NUM_THREADS=1`
`mpirun -np 1 ./forward_helmholtz_elastic-iso_Su_hdg.out parameter=parameter-file.txt`
Here, `parameter-file.txt` is the parameter file (text file) created in the formalism specified in Section 5.

The number of thread can be given directly with
`mpirun -np 1 -x OMP_NUM_THREADS=1`

When using `OpenMP`on a single node on a cluster, it is recommended to use the option
`mpirun --bind-to none`

4 Tutorial

In this section we describe the utilization of the code in the two configurations: (1) modeling the propagation of waves and (2) inverse problem.

4.1 Modeling benchmark

We illustrate the modeling of time-harmonic waves by describing the experiment that is available in the `examples/` folder (at the same level of the `code/` retrieved to install the software). This folder also contains a self-sufficient `readme` file.

In the folder `examples/`, untar the archive containing the benchmark and go to the extracted folder:
`tar -Jxf benchmark.tar.xz`
`cd benchmark`

Two parameter files are in the folder, to solve the propagation in an acoustic or elastic medium respectively: these are `par.modeling_acoustic` and `par.modeling_elastic`. They contain the

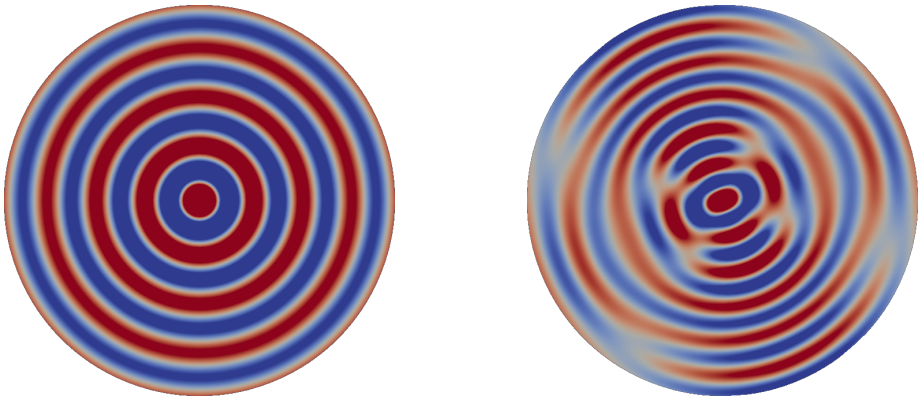
information to run the problem of interest, for instance the values of the medium parameters, the frequency, etc., which are all detailed in Section 5.

We refer to `$HAWENBIN` for the path where the executables have been compiled (which should be `../../bin`). To launch the code, following the previous section, one can use

```
mpirun -np 2 $HAWENBIN/forward_helmholtz_elastic-iso_Su_hdg.out  parameter=par.modeling_elastic
mpirun -np 2 $HAWENBIN/forward_helmholtz_acoustic-iso_hdg.out parameter=par.modeling_acoustic
```

where we use 2 `mpi` processors for the computations.

Two additional folders appear after the successful computations: `results_elastic` and `results_acoustic`, that contain the wavefields. The solutions are visualized using ParaView (https://www.paraview.org/, see Subsection 5.8), see Figure 1.



(a) Real part of the acoustic pressure field at 0.4 Hz with a scale between -0.1 and 0.1.

(b) Real part of the elastic velocity field v_z at 0.4 Hz with a scale between -20 and 20.

Figure 1: Wavefield solution obtained from the tutorial benchmark available in the `examples/` folder. These are visualized with Paraview and correspond to the files `results_acoustic/wavefield/wavefield_frequency_0.00000E+00_4.00000E-01Hz_src000001.pvtu` and `results_elastic/wavefield/wavefield_frequency_0.00000E+00_4.00000E-01Hz_src000001.pvtu`.

4.2 Inversion benchmark

A tutorial for inversion with the step-by-step actions is available from the website https://ffaucher.gitlab.io/hawen-website/, in the Tutorial webpage. The data (target and initial velocity models, meshes) for this test-case have to be downloaded from this webpage. It is the reconstruction of the acoustic Marmousi model.

5 Parameter file description

We provide the structures and keywords to use in the parameter files to use the software. The parameters are in an input text file such that

```
keyword=value
```

where `keyword` is the name of the parameter and `value` is its input values. Regarding the format, it can be:

– `logical`: `true` or `false`

– `text`: a chain of character (e.g., the path towards the mesh files),
– `integer`: integer number,
– `real`: real number,

In the following, the symbol † indicates that it can be a list of values. In this case, the values are separated by comma `,`. For instance,
`frequency_fourier=1,2,3,4.2`
indicates that the selection of frequency is 1, 2, 3 and 4.2 Hz.

In the parameter file, the character `#` indicates a comment: the line will not be read if it starts with `#`. For example,
`# this is a commentary line that will not be read`

We provide in Subsection 5.11 a template of parameter file.

5.1 Dimension

`dimension` [`integer`]

indicates the dimension of the problem to be solved: it must be 1, 2 or 3. Per convention, 2D problem considers the (x, z) plane, and 1D problem the axis x. For instance, if `dimension=2` is used for you parameter file, nothing related to the y-axis should be present.

5.2 Frequency and mode

For the equations that depend on the frequency (e.g., acoustic or elastic mechanical waves) and on a mode (e.g., one-dimensional ODE for the scalar wave equation in helioseismology under spherical symmetry), these must be given in the parameter file. We use a format of *complex frequency*, $\tilde{\omega}$, given by,

$$\tilde{\omega} = \mathrm{i}\,\omega - \mathfrak{s} = 2\mathrm{i}\,\pi\,\mathrm{f} - \mathfrak{s}, \tag{5.1}$$

where f is the 'Fourier' component, in Hz and $\mathfrak{s}$ the 'Laplace' component, in s^{-1}. The parameter file contains the list of f and $\mathfrak{s}$. In the case of lists of values, all combinations will be computed.

In the parameter file, there are two ways to give the frequency/mode: (1) with a list of values, and (2) specifying an interval with (2a) the initial value, (2b) the final value and (2c) the step between consecutive values, leading to an equi-partitioned interval. If both configurations (list and interval) are present in the parameter files, *the list prevails*.

Method 1: giving the list of values

`frequency_fourier` [$\texttt{real}_\dagger$]

List of Fourier frequency in Hz, corresponding to f in (5.1).

`frequency_laplace` [$\texttt{real}_\dagger$]

List of Laplace damping factors, in s^{-1}, corresponding to $\mathfrak{s}$ in (5.1).

`mode_list` [$\texttt{integer}_\dagger$]

List of modes.

Method 2: giving an interval

`frequency_fourier_min` [real]

First value for equi-partitioned interval of Fourier frequency in Hz.

`frequency_fourier_max` [real]

Last value for equi-partitioned interval of Fourier frequency in Hz.

`frequency_fourier_step` [real]

Step between consecutive Fourier frequency in Hz.

`frequency_laplace_min` [real]

First value for equi-partitioned interval of Laplace damping in s^{-1}.

`frequency_laplace_max` [real]

Last value for equi-partitioned interval of Laplace damping in s^{-1}.

`frequency_laplace_step` [real]

Step between consecutive Laplace damping in s^{-1}.

`mode_list_min` [integer]

First value for equi-partitioned interval of modes.

`mode_list_max` [integer]

Last value for equi-partitioned interval of modes.

`mode_list_step` [integer]

Step between consecutive modes.

Inversion only

In the context of inversion, one can specify a group of frequencies and modes to be inverted at the same time.

`fwi_frequency_group` [integer]

Number of frequency to be treated at the same time in the inversion algorithm, that is, the gradient is computed for all frequencies in the group. By default, it is equal to 1 so that each frequency is independently treated.

`fwi_mode_group` [integer]

Number of modes to be treated at the same time in the inversion algorithm, that is, the gradient is computed for all modes in the group. By default, it is equal to 1 so that each

mode is independently treated.

5.3 Acquisition (source and receivers)

5.3.1 Sources and receivers

The positions of the sources and receivers are read from an external text file. We provide below an example of files, where we specify the formalism. While the coordinates are specified in those file, the depth (z coordinates) of the receivers and of the sources can also be given directly in the parameter file, in which case it prevails.

Example of source file

```
3   # total number of sources
1   1   4000.  0.     20.
2   1   3000.  0.   1000.
3   2   5000.  0.   1000.  7000.  0.  1000.
```

- the first line indicates the total number of sources,
- then there is one line per source with the following:
  - the index of the current source,
  - the number of point for the source (for simultaneous point-sources),
  - the positions of the point(s): in x, y, and z. In 2D, y should be 0. In 1D, y and z should be 0. In the case of a planewave source, the direction of the source is given by the normalization of the x, y, z coordinates (see Subsection 5.3.2).

The receiver file follow the same formalism, that we give here.

Example of receiver file

```
5  # total number of receivers
  1          50.  0  100.
  2         100.  0  100.
  3         150.  0  100.
  4         200.  0  100.
  5         250.  0  100.
```

- the first line indicates the total number of receivers
- then there is one line per receiver with the following:
  - the index of the current receiver,
  - the positions of the point: in x, y, and z. In 2D, y should be 0. In 1D, y and z should be 0.

In the parameter file, the following keywords are used to link the sources and receivers external text files. When launching the test case, one can specify which of the sources in the file are to be taken into account.

`file_acquisition_source` [text]

Path towards the text source file.

`file_acquisition_receiver` [text]

Path towards the text receiver file.

`acquisition_receiver_fixed` [logical]

Indicates if the receivers are fixed, i.e., if the position of the receivers is the same for all sources.
If `acquisition_receiver_fixed=false`, the file with the receiver positions is given by `file_acquisition_receiver` (same for all source).
If `acquisition_receiver_fixed=true`, there must be one file per source to indicate the receivers position, file whose base name is contained in `file_acquisition_receiver`, and whose suffix is the source number. For instance, if `file_acquisition_receiver=rcv.txt`, the positions of the receivers related to the first source have to be in the file `rcv.txt1`, for the second source `rcv.txt2`, etc.

`src_depth` [real]

It forces the value of the source depth (coordinates z), replacing the one that is indicated in the source file. Consequently, all sources have the same depth if it is specified.

`rcv_depth` [real]

It forces the value of the receiver depth (coordinates z), replacing the one that is indicated in the receiver file. Consequently, all receivers have the same depth if it is specified.

`source_first` [integer]

Indicates the index of the first source to be taken into account when running the software. By default, it is set to 1, such that the first source in the source file is considered.

`source_last` [integer]

Indicates the index of the last source to be taken into account when running the software. By default, it is set to 10^9, and reduced to the total number of sources in the source file.

`source_step` [integer]

Indicates the step between two indexes for the sources to be taken into account. By default, it is set to 1, such that all sources are considered.

`acquisition_verbose` [logical]

When set to `true`, the output on the screen when launching the code will show the details of the acquisition that is read (sources positions, etc.) It is set to `false` by default.

`acquisition_rcv_normal` [text]

In the context of inversion with reciprocity misfit functional, the normal to the receivers position must be given. By default, it is set to `flat_z`, which means that it assumes a flat surface and the normal is the z axis. As an alternative, `flat_x` and `flat_y` can be used to specify a normal along x or y. The keyword `text-file` indicates that the normal for each receiver is read from the file given by `file_acquisition_receiver_normal`.

`file_acquisition_receiver_normal` [text]

In the case where the normal is specified in a text file, i.e., when `acquisition_rcv_normal` =`text-file`, it is the base filename that contains the normal information of each re-

ceiver. This base is extended with a number referring to the receiver. For instance, if `acquisition_rcv_normal=normal.txt`, the file `normal.txt1` contains the information for the first receiver, file `normal.txt2` for the second receiver, etc. The file structure is the same as the one for the receivers (the total number and then one line per receiver with its index and the 3 coordinates). Here, the 3 coordinates represent the normal directions at the current receiver.

5.3.2 Value and type of the source

Two types of sources are possible in the software: a delta-Dirac (point or multiple-point sources, depending on the source file), and a planewave source (possibly a sum of planewaves with different directions). The planewave is imposed on (some of) the boundary, which much be specified, cf. Subsection 5.4.2. The planewave is given by

$$f_{\text{plane-wave}}(x) = \alpha \, \exp\Big(-\mathrm{i} \, \frac{\omega}{c_{\text{pw}}} \, |x \cdot x_{\text{pw}}| \Big), \tag{5.2}$$

where x_{pw} is the direction of the planewave (read from the source file), c_{pw} is the speed of the planewave, which corresponds to the wave speed at the boundary where the planewave is applied. The source is scaled by a complex factor, α, which can either be directly given in the parameter file, or obtained from the Fourier transform of a Ricker wavelet source.

Popular in seismic, the Ricker wavelet is given by

$$f_{\text{Ricker}}(t) \; = \; a \left(1 \; - \; 2\,\pi^2 \, f_{\text{p}}^2 \, (t_0 + t - t_{\text{p}})^2\right) e^{-\,\pi^2 \, f_{\text{p}}^2 \, (t_0 + t - t_{\text{p}})^2} \, , \tag{5.3}$$

with t_0 the starting time, f_{p} the frequency peak, t_{p} the time peak and a the amplitude, to be given in the parameter file.

`source_type` [`text`]

Indicates the type of source to be considered, `dirac` by default. It can be `dirac` or `planewave`. One can also use the derivatives of the basis functions in the specified direction with `dirac-deriv_x`, `dirac-deriv_y` or `dirac-deriv_z`. For a divergence, one case use `dirac-divergence`. For a gradient, one can use `dirac-deriv`.

`source_component` [`text`$_\dagger$]

Indicates on which equation is the source applied. For instance, in acoustic, it can be applied as a pressure: `p` or the velocity `vx,vy,vz`, it corresponds to either f_{p} or $\boldsymbol{f}_{\mathbf{v}}$ in (2.2). The possibilities depend on the equation and we refer to Section 2:
`helmholtz_acoustic-iso`: can be `p,vx,vy,vz`;
`helmholtz_elastic-iso_Su`: can be `ux,uy,uz,sxx,syy,szz,sxy,sxz,syz`;
`helio_scalar-conjugated_spherical1d`: can be `w,v`;
`helio_scalar-conjugated`: can be `w,vx,vy,vz`.

While there is usually no ambiguity between the components, we note that for the propagator `helio_scalar-conjugated_spherical1d` for scalar waves under spherical symmetry in helioseismology, both equations of the system (2.24) are scalar, `v` is arbitrarily taking to correspond to the first equation and `w` to the second one.

`source_planewave_vel` [real]

Indicates the speed of the planewave, given by c_{pw} in (5.2)

`source_constant` [logical]

Indicates if the source is directly given as a constant complex number (`true`) or via the Fourier transform of the Ricker function (`false`). It is set to `false` by default.

Source coefficient given as a complex value

`source_constant_real` [real]

Real part of the source coefficient when given as a complex number, i.e., only used when `source_constant=true`.

`source_constant_imag` [real]

Imaginary part of the source coefficient when given as a complex number, i.e., only used when `source_constant=true`.

Source coefficient given as a Fourier transform of a Ricker function

`source_time0` [real]

Initial time of the Ricker function: t_0 in (5.3), set to 0 by default.

`source_timepeak` [real]

Time peak of the Ricker function: t_{peak} in (5.3), set to 0 by default.

`source_freqpeak` [real]

Frequency peak of the Ricker function: f_{peak} in (5.3), set to 0 by default.

`source_ampli` [real]

Amplitude of the Ricker function: a in (5.3), set to 0 by default, such as we have no source if it is not specified.

5.4 Discretization domain (mesh)

The code works with simplex cells, that is, triangles in two dimensions and tetrahedra in three dimensions.

5.4.1 Mesh format

We recommend the followings:

– For meshing two-dimensional domains, there is the software `triangle`, [29, 30], cf. https://www.cs.cmu.edu/~quake/triangle.html.

– For meshing three-dimensional domains, there is the software `TetGen`, [31], cf. http://wias-berlin.de/software/index.jsp?id=TetGen&lang=1.

– In addition, we also allow for the `.mesh` format in 2 and 3D.

– In one-dimension, we use a format called `segment`: here, the interval is decomposed following a text file that contains: the total number of nodes in the first line. Next, for every line, we have three values: the index of the segment (from 1 to the number of points), the position of this point, and the tag of the point (should be 0 for interior point and higher than 0 for the first and last ones).

`mesh_format` [text]

Indicates the format of the mesh, the followings are allowed:
1D: only the format `segment` is supported (see above). 2D: `triangle` for mesh generated with `triangle` (see above); `mesh` for `.mesh` format.
3D: `tetgen` for mesh generated with `tetgen` (see above); `mesh` for `.mesh` format.

`mesh_file` [text]

This is the path towards the mesh files. The extensions should not be given. For instance, using `triangle`, several mesh files are generated: e.g., `mesh.ele`, `mesh.node`, etc. Here the path should link towards the main name only, that is, `mesh`.

`mesh_partition` [text]

This is the partitioner used to split the mesh domain among the `mpi` processors. Only `metis` is supported for now. It is a required dependency of the code, cf. Subsection 3.1.

5.4.2 Mesh boundaries

Regarding the boundary conditions, each of the boundary faces of the mesh must have a specific tags (i.e., an integer), and all of the boundary face tags must be referenced in the parameter file.

`mesh_tag_dirichlet` [integer$_\dagger$]

Indicates the list of boundary mesh tags that correspond to a Dirichlet boundary condition. The formula depends on the equation, cf. Section 2

`mesh_tag_neumann` [integer$_\dagger$]

Indicates the list of boundary mesh tags that correspond to a Neumann boundary condition. The formula depends on the equation, cf. Section 2

`mesh_tag_absorbing` [integer$_\dagger$]

Indicates the list of boundary mesh tags that correspond to an absorbing boundary condition. The formula depends on the equation, cf. Section 2

`bc_impedance` [text]

This keyword decides the formula for the absorbing boundary condition (if any). It is by default `Sommerfeld_0`, the choices depend on the equation, cf. Section 2.

`mesh_tag_free-surface` [`integer`$_\dagger$]

Indicates the list of boundary mesh tags that correspond to a free surface boundary condition. The formula depends on the equation, cf. Section 2

`mesh_tag_wall-surface` [`integer`$_\dagger$]

Indicates the list of boundary mesh tags that correspond to a wall surface boundary condition. The formula depends on the equation, cf. Section 2

`mesh_tag_planewave` [`integer`$_\dagger$]

Indicates the list of boundary mesh tags that correspond to a plane-wave boundary condition. Then the boundary source is consequently applied, cf. Subsection 5.3.2. For this keyword to be valid, one must use the `planewave` source type.

5.5 Polynomial order and method

Using the HDG discretization, the solutions are represented with piecewise-polynomials functions, where we use the Lagrange basis. The order of the polynomial can be different for each cell, as indicated below (i.e., p-adaptivity).

`polynomial_order` [`integer`]

Depending on the values, we have three cases:
$>$ **0**: constant order for the polynomials given by this value.
$-$**1**: the order on each cell is read from the file which path is given by the keyword `polynomial_order_file`.
$-$**2**: the order on each cell is automatically computed depending on the wavelength for the current problem.

`polynomial_order_max` [`integer`]

In case of the automatic computation of the orders (`polynomial_order=-2`), this is the maximal allowed order for the polynomials.

`polynomial_order_min` [`integer`]

In case of the automatic computation of the orders (`polynomial_order=-2`), this is the minimal allowed order for the polynomials.

`polynomial_order_file` [`text`]

In case of the orders given by a file (`polynomial_order=-1`), this is the path towards the file. The first line of the file is the number of cells, and then we have one line for each cell with an integer value for its order. Here, the ordering of the cells follows the input mesh.

`evaluate_int_quadrature` [`logical`]

Indicates if the integral evaluation uses the reference element (`false`) or it uses the Gauss-Lobatto approximation (`true`). The Gauss-Lobatto is mandatory for some equation

and the piecewise-polynomial model representations.

`hdg_penalization` [real]

In case one wants to fix a constant value of the penalization for HDG, otherwise it is computed automatically depending on the equation.

5.6 Reading and representation of the physical parameters

The physical parameters are read from the input parameter files, they depend on the equation. We allow for different format for the inputs, such that

- a constant value, which is specified in the parameter file,
- a text file that gives the value on each cells,
- a binary file which represent the model on a Cartesian grid,
- we can also force a spherical representation for the model.

See below for more details and the specificity of each situation.

After reading the model from the parameter file, several possibilities exist for its numerical representation.

`model_representation` [text]

This keyword decides on the model representation within the run. We have the following possibilities:
`piecewise-constant`: this the default type, the models are represented with one value per mesh cell.
`piecewise-polynomial`: the models are represented by a polynomial on each cell, with order defined depending on the physical parameter, see below. It only makes sense if one uses the quadrature rules to evaluate the integral, with `evaluate_int_quadrature=true`.
`sep-original`: for the models read from a `sep` format, the full grid is kept during the run, and each quadrature point is read from the original models. It only makes sense if one uses the quadrature rules to evaluate the integral, with `evaluate_int_quadrature=true` and if a `sep` file is read for the model.
`dof`: the model is represented in a basis of Lagrange function. Therefore, on each cell, the model is represented via a sum of polynomial functions, analogous to the wavefield solution. The order is the same for all cells, and read from `model_representation_polynomial_order`. In particular, it is appropriate for `sep` file model, which are then converted to a more flexible (non-cartesian) representation. It only makes sense if one uses the quadrature rules to evaluate the integral, with `evaluate_int_quadrature=true`.

Remark 3. *One should be careful with the piecewise polynomial representation which, depending on the order, can give oscillating representation on the cell.*

5.6.1 Viscosity

It is possible to consider viscosity in the equation, see Section 2, in which case it is specified in the parameter file.

`viscosity` [logical]

Logical to indicates if viscosity is encoded in the equation (`false` by default). We refer to Section 2 for the encoding of viscosity depending on the equation.

`viscosity_model` [text]

In the case of viscosity (`viscosity=true`), this is the model of viscosity to consider. Depending on the equation, different model can be available, we refer to Section 2:
`helmholtz_acoustic-iso`: only the `kolsky-futterman` model is available.
`helmholtz_elastic-iso_Su`: can be `Kelvin-Voigt` or `Zener-model_simple`.
`helio_scalar-conjugated_spherical1d`: the available models are `gamma0_power-law` and `gamma`.
`helio_scalar-conjugated`: the available models are `gamma0_power-law` and `gamma`.

5.6.2 Representation of input physical models

We refer by `MODEL` to the generic model name, which possibilities depend on the equation, cf Section 2:

- `helmholtz_acoustic-iso`: we have `MODEL`={`vp`, `rho`}.
- `helmholtz_elastic-iso_Su`: we have `MODEL`={`vp`, `vs`, `rho`}.
- `helio_scalar-conjugated_spherical1d`: we have `MODEL`={`vp`, `alpha`, `dalpha`}.
- `helio_scalar-conjugated`: we have `MODEL`={`vp`, `alpha`, `dalpha`}.

In the case of attenuation, the corresponding model parameters must also be included, cf. Section 2 for the possibilities with the equation.

Each of the physical model must be given in the parameter file, with the possibilities described below. Obviously, each model must only have one of the following representation given.

`model_constant_MODEL` [real]

In the case of constant model, the value is given here.

`model_ascii_MODEL` [text]

Here it is the path towards a text file that give the list of constant value per cell: the first line is the number of cells (which must coincide with the mesh that is read). Then we have one line per cell with value of the parameter `MODEL`.

`model_sep_MODEL` [text]

When the model is read from a `sep` file, this is the path towards the header file.

`model_sep-spherical1D_MODEL` [text]

This link towards the header of a `sep` file, which should represent a one-dimensional

model, which is then spherically projected in the case of a 2D or 3D domains.

If more than one situation is fulfilled: the priority is given to the `sep` format, then the `sep` spherical, then the ascii and finally the constant model.

In addition, we have the followings.

`model_scale_MODEL` [`real`]

Apply a scaling to the model, which is thus multiplied by a constant value.

`model_representation_polynomial_order_MODEL` [`integer`]

In the case of a piecewise-polynomial model representation, each physical parameter has its own order of polynomial per cell, defined by this keyword.

5.6.3 Model compressed representation

In addition, it is possible to represent the model via a compressed representation, gluing together the values.

`model_representation_compression_MODEL` [`logical`]

Compression is applied if set to `true`, it is `false` by default

`model_representation_compression_format_MODEL` [`text`]

Indicates the format of the compression when it is applied. The possibilities are:
`neighbors`: the compression is selected from neighbor values.
`box`: this performs a structured compression on a grid.

`model_representation_compression_coeff_MODEL` [`real`]

with the `neighbors` format, the reduction of the number of values is given here. For instance, when set to 0.1, the final number of coefficients should represent about 10% of the total number of cells

`model_representation_compression_boxx_MODEL` [`real`]

with the `box` format, it is the (metric) size of the box that will use the same values, in the x axis.

`model_representation_compression_boxy_MODEL` [`real`]

with the `box` format, it is the (metric) size of the box that will use the same values, in the y axis.

`model_representation_compression_boxz_MODEL` [`real`]

with the `box` format, it is the (metric) size of the box that will use the same values, in the z axis.

5.7 Linear Algebra

The software uses the direct solver MUMPS to solve the linear system, we refer to its user manual for more information on the different options. It is a required dependency of the code, cf. Subsection 3.1. In particular, in recent versions of MUMPS, one can use the block low rank option to reduce the memory consumption. For non-expert user, the only important keyword is the first one that indicates how many right-hand sides are solved the same time. Because we use a direct solver, the resolution for many right-hand sides is very efficient, and one should solve as many as possible at the same time (the available memory being the limit here).

`parallel_rhs` [integer]

the number of parallel right-hand sides during the solve phase, i.e., how many sources are solved at the same time. This value should by as large as the memory allows you to.

`mumps_analysis` [integer]

MUMPS analysis phase: 0 (automatic decision), 1 (sequential analysis), 2 (parallel analysis). Due to the observed instability of the parallel partitioner, it is recommended to use the sequential analysis, `mumps_analysis=1`. For parallel partitioner (`parmetis` and `pt-scotch`), MUMPS must have been accordingly compiled, and those libraries must be linked with HAWEN as well.

`mumps_partitioner_par` [integer]

selection of partitioner for MUMPS in case of parallel analysis (depends on how MUMPS has been compiled): 0 (automatic decision), 1 (PT Scotch), 2 (Parmetis).

`mumps_partitioner_seq` [integer]

selection of partitioner for MUMPS in case of sequential analysis (depends on how MUMPS has been compiled): 0 (AMD), 1 (Pivot), 2 (AMF), 3 (Scotch), 4 (Pord), 5 (metis), 6 (Mindeg), 7 (automatic decision).

`mumps_verbose` [integer]

print MUMPS infos on screen: 0 (no output), 1 (output infos).

`mumps_memory_increase_percent` [real]

memory workspace for MUMPS in percent: it must be at least 30 (default value if keyword is missing). It is recommended to be increased when using the block low rank option.

`mumps_memory_limit` [real]

With version of MUMPS higher than 5.2, one can specify the memory limit that MUMPS can use. It is given in MegaBytes.

`mumps_lowrank` [integer]

Indicates if the MUMPS low rank option is employed (for memory reduction): 0 (inactive, by default), 1 (for factorization only), 2 (for factorization and solve phases).

`mumps_lowrank_tol` [real]

If the low rank option is activated, it is the value of the threshold ($0 <$ `tol` < 1).

`mumps_lowrank_variant` [integer]

If the low rank option is activated, one can use its variant version. It corresponds to ICNTL(36) in MUMPS user's guide, which can be 0 (by default) or 1 (activated).

`mumps_advanced_experimental` [integer]

It corresponds to the KEEP(370-371) in MUMPS user's guide, which can be 0 (by default) or 1 (activated).

`mumps_multithread_tree` [integer]

For MUMPS version higher or equal to 5.3.3, it is possible to use an improved multi-threading using tree parallelism: it corresponds to the KEEP(401) in MUMPS user's guide, which can be 0 (by default) or 1 (activated).

`mumps_block_structure` [integer]

For MUMPS version higher or equal to 5.3.3, this option exploits the analysis by block: it corresponds to the ICNTL(15) in MUMPS user's guide, which can be 0 (by default) or 1 (activated).

`mumps_root_parallel` [integer]

This option indicates is the root node factorization is done in parallel (0) or sequential (1): it corresponds to the ICNTL(13) in MUMPS user's guide.

`mumps_cb_compress` [integer]

This option indicates is the contribution blocks are compressed (1) or not (0): it corresponds to the ICNTL(37) in MUMPS user's guide. This option is only used if the block low-rank option is activated.

`mumps_crit_pivoting` [real]

This option indicates the threshold for numerical pivoting: it corresponds to the CNTL(1) in MUMPS user's guide. This option is only used if the block low-rank option is activated.

`mumps_relaxed_pivoting` [integer]

This option indicates the relaxed pivoting option and corresponds to KEEP(268) in MUMPS user's guide. It is either not activated (0, per default), applied on all matrices (1) or applied only on large matrices (-2).

`mumps_mpi_to_komp` [integer]

This option remaps the original number of `mpi` processes towards threads. It is not activated if it is less than or equal to 1. Otherwise, the original number of `mpi` processes will be divided by this factor, and that many threads will be used for MUMPS operations.

`mumps_mixed_precision` [integer]

To activate mixed precision Block Low-Rank feature, it can be 0 (by default) or 1 (activated).

`mumps_ngpu_per_mpi` [integer]

To activate the number of GPU used per `mpi` process. 0 (by default) means that no GPU

is used.

`mumps_gpu_mpi_tuning` [integer]

Tuning option in the case where GPU are used with MUMPS. It corresponds to the KEEP(66) in MUMPS user's guide.

5.8 Outputs

This is the format in which the outputs are saved, one can use a Cartesian (structured) grid, which implies that a projection will be computed from the unstructured mesh towards the grid. Note that this step of projection can take some time because all node of the grid would need the degree of freedom weights. Therefore, for mesh composed of a high number of cells, it is faster to use the unstructured format only. We allow the followings:

- Unstructured model representation rely on the software ParaView formalism, we refer to [1, 6] and https://www.paraview.org/.
- By `sep` format, we refer to a binary file saved in a Cartesian grid, in the spirit of the format defined in seismic (Stanford Exploration Project), http://sep.stanford.edu/lib/exe/fetch.php?media=sep:software:sepman.pdf. The binary file uses the extension `.H@` while a header, a text file summarizing the dimensions and origins of the grid, has the extension `.H`.

Remark 4. *In the unstructured format, which uses Paraview formalism, the three-dimensional domain does not support order higher than* 5*.*

Shared (modeling and inversion) output information

`workpath` [text]

Path toward the directory where the results are saved (note that sub-folder will be created when running the code).

`save_unstructured_vtk` [logical]

Indicates if the results are saved on an unstructured grid using `vtk` (to read with Paraview) format.

`save_structured_vtk` [logical]

Indicates if the results are saved on a structured Cartesian grid using `vtk` (to read with Paraview) format.

`save_structured_sep` [logical]

Indicates if the results are saved on a structured Cartesian grid using `sep` format (binary file).

`dx` [real]

In the case where structured grid is saved, this is the step specification of the Cartesian grid along x axis.

`dy` [real]

In the case where structured grid is saved, this is the step specification of the Cartesian grid along y axis.

`dz` [real]

In the case where structured grid is saved, this is the step specification of the Cartesian grid along z axis.

Specific to modeling

`save_wavefield` [logical]

Indicates if the total wavefield is saved (`true`) or not (`false`), that is, the solution on the entire domain of interest.

`save_receivers` [logical]

Indicates if the soution at the receiver positions is saved (`true`) or not (`false`), here, the positions are obtained from the receiver file given by keyword `file_acquisition_receiver`.

`save_wavefield_component` [text$_\dagger$]

In the case where the wavefield is saved (`save_wavefield=true`), it is the list of components that will be saved. The possibilities depend on the type of equation, we refer to Section 2:
`helmholtz_acoustic-iso`: can be `p,vx,vy,vz`;
`helmholtz_elastic-iso_Su`: can be `ux,uy,uz,sxx,syy,szz,sxy,sxz,syz`;
`helio_scalar-conjugated_spherical1d`: can be `w,v`;
`helio_scalar-conjugated`: can be `w,v`.

`save_receivers_component` [text$_\dagger$]

In the case where the solution at the receiver positions is saved (`save_receivers=true`), it is the list of components that will be saved. The possibilities depend on the type of equation, we refer to Section 2:
`helmholtz_acoustic-iso`: can be `p,vx,vy,vz`;
`helmholtz_elastic-iso_Su`: can be `ux,uy,uz,sxx,syy,szz,sxy,sxz,syz`;
`helio_scalar-conjugated_spherical1d`: can be `w,v`;
`helio_scalar-conjugated`: can be `w,v`.

5.9 Inversion

We refer to [13] and the reference therein for the details on inversion procedure using iterative minimization, following the full waveform inversion approach. The misfit functional is defined by

$$\mathcal{J}(\boldsymbol{m}) = \frac{1}{2}\|\mathcal{F}(\boldsymbol{m}) - \boldsymbol{d}\|\,, \tag{5.4}$$

where $\boldsymbol{m}$ stands for the physical model to be inverted, e.g., velocity and density in acoustics. The observations are denoted $\boldsymbol{d}$: these are read from the disk, the path towards these measurements

must be given in the parameter file. The forward problem $\mathcal{F}(\boldsymbol{m})$ solves the wave equation of choice, cf. Section 2, and keeps the solution at the receivers location only (which coordinates are read from the input file given in the keyword `file_acquisition_receiver`). The choice of norm can also be specified in the parameter file, see below.

The minimization of the misfit functional follows an iterative minimization, we refer to [27] for the review of methods, and to [13] and the reference therein for details in seismic imaging. In addition, the following references have been using HAWEN: [15, 18, 16].

`fwi_niter` [integer]

Number of iterations per (group of) frequency.

`fwi_niter_min` [integer]

Minimal number of iterations per (group of) frequency. The stagnation is not being tested as long as this number of iterations has not been reached.

`fwi_stagnation_tol` [real]

This is a stagnation tolerance for last five iterations. Once `fwi_niter_min` is reached, the evolution of the misfit function is computed compared to the last five iterations, if it is less than the stagnation (relative difference), we force to the next (group of) frequency.

`fwi_path_data` [text]

The path towards the observed data.

`fwi_parameter` [text$_\dagger$]

List of physical parameter to be inverted, these depend on the equation, as indicated in Section 2.

`fwi_parameter_function` [text$_\dagger$]

Each selected parameter is allowed a function, such as the inverse, for instance, if `fwi_parameter=cp,rho` in acoustic, that is inverting the wave speed and density, options are
`id`: inverting the parameter: v_p.
`squared`: inverting the parameter squared: v_p^2.
`inv`: inverting the inverse: v_p^{-1}.
`inv-sq`: inverting the inverse squared v_p^{-2}.
`sqrt`: inverting the square root: $v_p^{1/2}$.
`sqrt-inv`: inverting the inverse square root $v_p^{-1/2}$.
`log`: inverting the log of the parameter: $\log(v_p)$.
`ignore`: the parameter is not inverted.

For instance, combining
`fwi_parameter=cp,rho`
`fwi_parameter_function=inv-sq,ignore`
means that the inversion procedure is carried with respect to v_p^{-2} only.

5.9.1 Model representation for inversion

`fwi_gradient_model_param` [text]

Indicate how the unknown is represented during inversion, with the following choices.
`piecewise-constant`: by default, the unknown is represented with a piecewise-constant, that is, one value per cell.
`dof`: the model is represented in a basis of lagrange function per cell, see the keyword `model_representation`.

`fwi_gradient_model_param_doforder` [integer]

In the case where `fwi_gradient_model_param=dof`, it is order of the basis for each node. It is a constant on the entire mesh. Currently, all of the unknowns have the same order, and it must be the same as the order given in the `model_representation`. Furthermore, only the Lagrange basis of order 1 is available for now.

5.9.2 Misfit function and search direction

`fwi_misfit` [text]

Choice of misfit functional, among the following, we recommend `l2`, `log` or `reciprocity`.
`l2`: standard misfit functional (difference squared).
`reciprocity`: reciprocity-gap misfit functional, cf. [2, 15, 16].
`l2_real` standard misfit functional only using the real parts of the signals.
`l2_imag` standard misfit functional only using the imaginary parts of the signals.
`log` standard misfit functional using the log of the signals.
`l2_modulus` standard misfit functional using the modulus of the signals.
`log_modulus` standard misfit functional using the log of the modulus of the signals.
`l2_phase` standard misfit functional using the phase of the signals.
`l2_power2` standard misfit functional using the squared of the signals.
`l2_power3` standard misfit functional using the signals at power 3.

`fwi_component` [$\texttt{text}_\dagger$]

List of component that are among the observations, the selection depends on the equation, see Section 2.
`helmholtz_acoustic-iso`: can be `p,vx,vy,vz`;
`helmholtz_elastic-iso_Su`: can be `ux,uy,uz,sxx,syy,szz,sxy,sxz,syz`;
`helio_scalar-conjugated_spherical1d`: can be `w,v`;
`helio_scalar-conjugated`: can be `w,v`.

`fwi_reciprocity_nobs` [integer]

Number of observational sources for the reciprocity-gap waveform inversion. By nature, it does not have to be the same as the number of computational ones, see [15, 2].

`fwi_gradient_metric` [text]

Indicate the metric for the scalar product for the gradient, by default it set to `identity`. Other possibilities are:
`identity`: usual: $< u, v >= u^* v$.
`area` using the area of the cell as a scaling: $< u, v >= u^* v/\text{area(cell)}$.
`mass` using the mass matrix $< u, v >= u^* M v$.
`mass_inv` using the inverse of the mass matrix $< u, v >= u^* M^{-1} v$.
In general, we recommend the use of `mass_inv`.

`fwi_search_direction` [text]

Choice of search direction method, options are the followings, we refer to [27, 13] for additional information.
`gradient-descent`: gradient descent method.
`l-bfgs`: limited-memory BFGS formula.
`nlcg_FR`: Non-linear conjugate gradient method with Fletcher–Reeves formula.
`nlcg_PR`: Non-linear conjugate gradient method with Polak–Ribière formula.
`nlcg_HS`: Non-linear conjugate gradient method with Hestenes–Stiefel formula.
`nlcg_DY`: Non-linear conjugate gradient method with Dai–Yuan formula.

`fwi_pseudo_hessian_scale` [logical]

Should not be used for now.

5.9.3 Source reconstruction

`fwi_source_inversion` [logical]

Indicates if the source characteristic has to be recovered in parallel to the physical parameters. Note that this is not needed with reciprocity-gap misfit function, which does not use the observational source functions.

`fwi_source_inversion_shared` [logical]

In the case of source inversion, it indicates if all sources have the same source functions. This is the only supported feature for now (i.e., `true`).

5.9.4 Linesearch method

`fwi_linesearch_method` [text]

This is the lineasearch method: for now we have: `backtracking` for the standard method, or `none` if no linesearch is applied.

`fwi_linesearch_niter` [integer]

In the case of `backtracking` linesearch, this is the number of iterations that is performed at most.

`fwi_linesearch_stepX` [real$_\dagger$]

Initial linesearch step for the model `X`. Here, `X` is an integer from 1 to the number of models (i.e., physical parameters) for this equation. One can give a list of values: one for each (group of) frequency.

`fwi_linesearch_reset_iter` [logical]

Indicates if the linesearch step is reset to its default value at each iteration, it is `false` by default: it re-uses the one from the previous iteration. In any case, the step is reset fro each (group of) frequency.

`fwi_linesearch_control` [text]

Possibility of control criterion for the linesearch, it can be either `Goldstein` or `Armijo`. Otherwise, it does not use any control method.

`fwi_linesearch_control_tol` [real]

Indicates the tolerance for the control method: it must be $0 <$ `tol` < 0.5 for `Goldstein` and $0 <$ `tol` < 1 for `Armijo`.

`fwi_linesearch_lowrank` [real]

Allow for block low-rank tolerance specific to linesearch, in order to reduce the cost of the factorization, using the solve `MUMPS`.

5.9.5 Receiver offsets

`fwi_offset_rcv` [text]

Indicates if there is a limiting offset for the receivers (`false` by default). Receivers too far from the source are discarded if set to `true`.

`fwi_offset_rcv_xmin` [real]

If `fwi_offset_rcv=true`, receivers with the distance to the source lower than the given value in the x direction are not taken into account.

`fwi_offset_rcv_xmax` [real]

If `fwi_offset_rcv=true`, receivers with the distance to the source higher than the given value in the x direction are not taken into account.

`fwi_offset_rcv_ymin` [real]

If `fwi_offset_rcv=true`, receivers with the distance to the source lower than the given value in the y direction are not taken into account.

`fwi_offset_rcv_ymax` [real]

If `fwi_offset_rcv=true`, receivers with the distance to the source higher than the given value in the y direction are not taken into account.

`fwi_offset_rcv_zmin` [real]

If `fwi_offset_rcv=true`, receivers with the distance to the source lower than the given value in the z direction are not taken into account.

`fwi_offset_rcv_zmax` [`real`]

If `fwi_offset_rcv=true`, receivers with the distance to the source higher than the given value in the z direction are not taken into account.

`fwi_offset_rcv_rmin` [`real`]

If `fwi_offset_rcv=true`, receivers with a radial distance to the source lower than the given value are not taken into account.

`fwi_offset_rcv_rmax` [`real`]

If `fwi_offset_rcv=true`, receivers with a radial distance to the source higher than the given value are not taken into account.

5.9.6 Restriction of the domain and of the parameter values

`model_min_MODEL` [`real`]

Minimal value allowed for model `MODEL`. `MODEL` takes the name of the models associated with the equation, see Section 2.

`model_max_MODEL` [`real`]

Maximal value allowed for model `MODEL`. `MODEL` takes the name of the models associated with the equation, see Section 2.

`fwi_restrict_box` [`logical`]

Indicates if some part of the domain is not inverted (`false` by default).

`fwi_xmin` [`real`]

If `fwi_restrict_box=true`, the portion of the domain for which $x <$ `fwi_xmin` is not inverted.

`fwi_xmax` [`real`]

If `fwi_restrict_box=true`, the portion of the domain for which $x >$ `fwi_xmax` is not inverted.

`fwi_ymin` [`real`]

If `fwi_restrict_box=true`, the portion of the domain for which $y <$ `fwi_ymin` is not inverted.

`fwi_ymax` [`real`]

If `fwi_restrict_box=true`, the portion of the domain for which $y >$ `fwi_ymax` is not inverted.

`fwi_zmin` [real]

If `fwi_restrict_box=true`, the portion of the domain for which $z <$ `fwi_zmin` is not inverted.

`fwi_zmax` [real]

If `fwi_restrict_box=true`, the portion of the domain for which $z >$ `fwi_zmax` is not inverted.

`fwi_r0x` [real]

If `fwi_restrict_box=true`, inversion is only carried out in a circular portion of the domain, which radius is positioned in $(r0_x, r0_y, r0_z)$. Hence, `fwi_r0x` indicates the x-coordinate of the center. It is zero by default.

`fwi_r0y` [real]

If `fwi_restrict_box=true`, inversion is only carried out in a circular portion of the domain, which radius is positioned in $(r0_x, r0_y, r0_z)$. Hence, `fwi_r0y` indicates the y-coordinate of the center. It is zero by default.

`fwi_r0z` [real]

If `fwi_restrict_box=true`, inversion is only carried out in a circular portion of the domain, which radius is positioned in $(r0_x, r0_y, r0_z)$. Hence, `fwi_r0z` indicates the z-coordinate of the center. It is zero by default.

`fwi_rmin` [real]

If `fwi_restrict_box=true`, inversion is only carried out in a circular portion of the domain where the radial position $r = \sqrt{x^2 + y^2 + z^2}$ is larger than or equal to `fwi_rmin`.

`fwi_rmax` [real]

If `fwi_restrict_box=true`, inversion is only carried out in a circular portion of the domain where the radial position $r = \sqrt{x^2 + y^2 + z^2}$ is smaller than or equal to `fwi_rmax`.

5.9.7 Cleaning, additional outputs

`fwi_debug_save_gradient` [logical]

For debug purposes, allows to save the gradients for each iteration onto the disk (`false` by default).

`fwi_output_clean` [integer]

Allows to automatically clean the output directory at the end of the algorithm (gradient files, etc.)
`0`: nothing is deleted (default option)
`1`: gradient and search direction arrays at each iteration are deleted.
$>$ 1: gradient, search direction and temporary models at each iteration are deleted.

5.9.8 Regularization

Regularization is not yet validated.

5.10 Eigenproblem

If the eigenproblem version of HAWENis compiled, providing ARPACK and PARPACK have been linked, these are the options to allow the computation of the highest or lowest eigenvalues of the discretization matrix.

`eigenproblem_format` [text]

Choice of eigenproblem: either `highest` or `lowest` which indicates if the eigenvalues with highest or of lowest magnitudes are to be computed.

`eigenproblem_nev` [integer]

Number of eigenvalues to be computed, set to 10 by default.

`eigenproblem_tol` [real]

Tolerance for the computations, set to 0 by default, for exact computations

`eigenproblem_verbose` [integer]

Indicates the level of screen information, can be 0 (no info), 1 (basic info), or 2 (print the output eigenvalues). It is set to 2 by default.

5.11 Template parameter file: HDG 2D elastic propagation

We give here an illustration of a parameter file, containing the main keywords, in the case of an elastic isotropic 2D problem.

```
1  ###############################################################################
2  ## PARAMETER FILE: 2D ELASTIC HDG DISCRETIZATION
3  ###############################################################################
4  ## 1) Problem information ######################################################
5  dimension=2
6  frequency_fourier=4     # 4Hz frequency
7  frequency_laplace=0     # no damping
8
9  ## 2) Acquisition #############################################################
10 file_acquisition_source=./acquisition/source.txt
11 file_acquisition_receiver=./acquisition/receiver.txt
12 src_depth=1000.         # force here or read in file
13 # rcv_depth=100.        # force here or read in file
14 source_type=dirac       # could be a planewave too.
15 source_component=vx,vz  # e.g., {p,vx,vy,vz,sxx,syy,szz,...}
16 source_first=1          # only the first source is solved
17 source_last=1           # only the first source is solved
```

```
source_step=1           # only the first source is solved
acquisition_verbose=true

## source characterization from a Ricker wavelet
source_time0=0.
source_timepeak=0.
source_freqpeak=10.
source_ampli=1.e3
source_constant=false     # possibility to give source coeff. instead
# source_constant_real=1.
# source_constant_imag=1.
##########################################################################

## 3) model parameters ###################################################
# model_sep_vp=                 # read from sep file
# model_ascii_vp=               # read list of values per mesh cell
# model_scale_vp=               # to scale the parameter
model_constant_vp=2500.         # constant value for vp
model_constant_vs=1000.         # constant value for vs
model_constant_rho=1.0          # constant value for rho
viscosity=false                 # viscous behavious: none here
viscosity_model=Kelvin-Voigt    # viscous model name
model_constant_eta-lambda=0.    # unused because viscosity=false
model_constant_eta-mu=1e5       # unused because viscosity=false
##########################################################################

## 4) mesh information ###################################################
mesh_file=./mesh/mesh_5k/mesh.1
mesh_format=triangle
mesh_partition=metis
# boundary tag infos: flag if any and list of tags
mesh_tag_absorbing=2,3,4    # the mesh tag where ABC           is applied
mesh_tag_free-surface=1     # the mesh tag where Free-Surface  is applied
mesh_tag_dirichlet=         # the mesh tag where Dirichlet     is applied
mesh_tag_neumann=           # the mesh tag where Neumann       is applied
mesh_tag_wall-surface=      # the mesh tag where Wall-Surface  is applied

# HDG formulation
evaluate_int_quadrature=false # using reference element for integrals
polynomial_order=-2           # automatically computed from wavelength
polynomial_order_min=3        # at least order 3 polynomials
polynomial_order_max=6        # at most  order 6 polynomials

##########################################################################

## 5) solver #############################################################

```

```
## number of parallel rhs
parallel_rhs=64

## information for Mumps solver
mumps_analysis=1              #0=AUTO, 1=SEQUENTIAL, 2=PARALLEL
mumps_partitioner_seq=4       #0=AMD, 1=PIVOT, 2=AMF, 3=SCOTCH, 4=PORD,
                              #5=METIS, 6=MINDEG, 7=AUTO
mumps_partitioner_par=1       #0=AUTO, 1=PT_SCOTCH,  2=PARMETIS
## low-rank options
mumps_lowrank=0               # low-rank is not used
mumps_lowrank_tol=1e-9
mumps_lowrank_variant=0
mumps_advanced_experimental=0
mumps_memory_increase_percent=50
mumps_memory_limit=
mumps_verbose=0

#############################################################################

## 6) io ####################################################################
workpath=./results                                  # folder to save results
save_wavefield=true                                 # to save wavefields
save_receivers=true                                 # to save solution at rcv.
save_structured_vtk=false
save_structured_sep=false
save_unstructured_vtk=true
save_wavefield_component=vx,vz,sxx,szz,sxz  # list of components to save
save_receivers_component=vx,vz,sxx,szz,sxz  # list of components to save
dx=100. # for structured save, we need the discretization
dy=100. # for structured save, we need the discretization
dz=100. # for structured save, we need the discretization
#############################################################################
## END OF PARAMETER FILE
#############################################################################
```

Index

References

[1] J. Ahrens, B. Geveci, and C. Law, *ParaView: An end-user tool for large data visualization*, The visualization handbook, 717 (2005).

[2] G. Alessandrini, M. V. De Hoop, F. Faucher, R. Gaburro, and E. Sincich, *Inverse problem for the Helmholtz equation with Cauchy data: reconstruction with conditional well-posedness driven iterative regularization*, ESAIM: M2AN, 53 (2019), pp. 1005–1030, https://doi.org/10.1051/m2an/2019009.

[3] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. A. Mary, *Bridging the gap between flat and hierarchical low-rank matrix formats: The multilevel block low-rank format*, SIAM Journal on Scientific Computing, 41 (2019), pp. A1414–A1442, https://doi.org/10.1137/18M1182760.

[4] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal on Matrix Analysis and Applications, 23 (2001), pp. 15–41, https://doi.org/10.1137/S0895479899358194.

[5] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, *Hybrid scheduling for the parallel solution of linear systems*, Parallel computing, 32 (2006), pp. 136–156, https://doi.org/10.1016/j.parco.2005.07.004.

[6] U. Ayachit, *The ParaView guide: a parallel visualization application*, Kitware, Inc., 2015.

[7] H. Barucq, F. Faucher, D. Fournier, L. Gizon, and H. Pham, *Efficient and accurate algorithm for the full modal Green's kernel of the scalar wave equation in helioseismology*, SIAM Journal on Applied Mathematics, 80 (2020), pp. 2657–2683, https://doi.org/10.1137/20M1336709.

[8] H. Barucq, F. Faucher, D. Fournier, L. Gizon, and H. Pham, *Efficient computation of the modal outgoing Green's kernel for the scalar wave equation in helioseismology*, Research Report RR-9338, Inria Bordeaux Sud-Ouest ; Magique 3D ; Max-Planck Institute for Solar System Research, April 2020, https://hal.archives-ouvertes.fr/hal-02544701.

[9] H. Barucq, F. Faucher, and H. Pham, *Outgoing solutions to the scalar wave equation in helioseismology*, Research Report RR-9280, Inria Bordeaux Sud-Ouest ; Project-Team Magique3D, August 2019, https://hal.archives-ouvertes.fr/hal-02168467.

[10] H. Barucq, F. Faucher, and H. Pham, *Outgoing solutions and radiation boundary conditions for the ideal atmospheric scalar wave equation in helioseismology*, ESAIM: Mathematical Modelling and Numerical Analysis, 54 (2020), pp. 1111–1138, https://doi.org/10.1051/m2an/2019088.

[11] J. M. Carcione, *Wave fields in real media: Wave propagation in anisotropic, anelastic, porous and electromagnetic media*, Elsevier, Third ed., 2015.

[12] M. Deheuvels, F. Faucher, and D. Brito, *Numerical and experimental study of ultrasonic seismic waves propagation and attenuation on high quality factor samples*, Geophysical Prospecting, (2023), https://doi.org/10.1111/1365-2478.13465.

[13] F. Faucher, *Contributions to seismic full waveform inversion for time harmonic wave equations: Stability estimates, convergence analysis, numerical experiments involving large scale optimization algorithms*, PhD thesis, Université de Pau et Pays de l'Ardour, 2017, https://hal.archives-ouvertes.fr/tel-01807861.

[14] F. Faucher, ***hawen:*** *time-harmonic wave modeling and inversion using hybridizable discontinuous Galerkin discretization*, Journal of Open Source Software, 6 (2021), https://doi.org/10.21105/joss.02699, https://joss.theoj.org/papers/aaac99edf0ce2d26a3e22b69558b0b55.

[15] F. Faucher, G. Alessandrini, H. Barucq, M. de Hoop, R. Gaburro, and E. Sincich, *Full Reciprocity-gap Waveform Inversion, enabling sparse-source acquisition*, Geophysics, 85 (2020), pp. R461–R476, https://doi.org/10.1190/geo2019-0527.1.

[16] F. Faucher, M. V. de Hoop, and O. Scherzer, *Reciprocity-gap misfit functional for distributed acoustic sensing, combining data from passive and active sources*, Geophysics, 86 (2021), pp. R211–R220, https://doi.org/10.1190/geo2020-0305.1.

[17] F. Faucher, C. Kirisits, M. Quellmalz, O. Scherzer, and E. Setterqvist, *Diffraction tomography, fourier reconstruction, and full waveform inversion*, arXiv preprint, (2021), https://arxiv.org/abs/2110.07921.

[18] F. Faucher and O. Scherzer, *Adjoint-state method for Hybridizable Discontinuous Galerkin discretization, application to the inverse acoustic wave problem*, Computer Methods in Applied Mechanics and Engineering, 372 (2020), p. 113406, https://doi.org/10.1016/j.cma.2020.113406.

[19] F. Faucher and O. Scherzer, *Quantitative inverse problem in visco-acoustic media under attenuation model uncertainty*, Journal of Computational Physics, 472 (2023), p. 111685, https://doi.org/10.1016/j.jcp.2022.111685.

[20] W. Futterman, *Dispersive body waves*, Journal of Geophysical Research, 67 (1962), pp. 5279–5291, https://doi.org/10.1029/JZ067i013p05279.

[21] D. Givoli and J. B. Keller, *Non-reflecting boundary conditions for elastic waves*, Wave motion, 12 (1990), pp. 261–279, https://doi.org/10.1016/0165-2125(90)90043-4.

[22] F. Johansson, *Arb: efficient arbitrary-precision midpoint-radius interval arithmetic*, IEEE Transactions on Computers, 66 (2017), pp. 1281–1292.

[23] F. Johansson, *Computing hypergeometric functions rigorously*, ACM Transactions on Mathematical Software, to appear (2019).

[24] J. B. Keller and M. J. Grote, *Exact nonreflecting boundary condition for elastic waves*, SIAM Journal on Applied Mathematics, 60 (2000), pp. 803–819.

[25] H. Kolsky, *The propagation of stress pulses in viscoelastic solids*, Philosophical magazine, 1 (1956), pp. 693–710, https://doi.org/10.1080/14786435608238144.

[26] P. A. Martin, *Time-Domain Scattering*, vol. 180, Cambridge University Press, 2021, https://doi.org/10.1017/9781108891066.

[27] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research, 2 ed., 2006.

[28] H. Pham, F. Faucher, and H. Barucq, *On the implementation of Hybridizable Discontinuous Galerkin discretization for linear anisotropic elastic wave equation: Voigt-notation and stabilization*, Research Report RR-9533, Inria Bordeaux Sud-Ouest; Project-Team Makutu, December 2023, https://hal.archives-ouvertes.fr/hal-04356602.

[29] J. R. Shewchuk, *Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*, in Applied Computational Geometry: Towards Geometric Engineering, M. C. Lin and D. Manocha, eds., vol. 1148 of Lecture Notes in Computer Science, Springer-Verlag, May 1996, pp. 203–222, https://doi.org/10.1007/BFb0014497. From the First ACM Workshop on Applied Computational Geometry.

[30] J. R. Shewchuk, *Delaunay refinement algorithms for triangular mesh generation*, Computational geometry, 22 (2002), pp. 21–74, https://doi.org/10.1016/S0925-7721(01)00047-5.

[31] H. Si, *TetGen, a Delaunay-based quality tetrahedral mesh generator*, ACM Transactions on Mathematical Software (TOMS), 41 (2015), pp. 1–36, https://doi.org/10.1145/2629697.

[32] B. Ursin and T. Toverud, *Comparison of seismic dispersion and attenuation models*, Studia Geophysica et Geodaetica, 46 (2002), pp. 293–320, https://doi.org/10.1023/A:1019810305074.