PSD

July 7, 2020

Contents

Chapter 1

Introduction

1.1 Introduction

PSD is a finite elements-based solid mechanics solver with capabilities of performing High Performance Computing (HPC) simulations with billions of unknowns. The kernel of PSD is wrapped around FreeFEM for finite element discretization, and PETSc for linear algebra/Preconditioning. PSD solver contains straightforward supports for static or dynamic simulations with linear and nonlinear solid mechanics problems. Besides these hybrid-phase field fracture mechanics models have also been incorporated within PSD. For dynamics the generalized-α model for time discretization is used, this models enable straightforward use of Newmark-β, central difference, or HHT as time discretization. PSD uses sate-of-the art domain-decomposition paradigm via vectorial finite elements for parallel computing and all solvers are proven to scale quasi-optimally. PSD has proven scalabilty uptill 13,000 cores with largest problem solved containing over 5 Billion unknowns.

1.2 PSD evolution

PSD has been maturing and evolving with time, following subsections present the highlights of some key changes made to each PSD version.

1.2.1 Version 2.0

Added

- New preprocessing via C++, PSD_PreProcess binary.
- Scripting is now handled in `.hpp` files.
- New time discretization option `-timediscretization [string]` for dynamic simulation, with `[string]` choose between the following options `generalized-alpha`, `newmark-beta`, and `hht`.
- New Dirichlet point boundary conditions by `-dirichletpointconditions [int]` flag, with `[int]` number of Dirichlet point conditions.
- Paraxial element support for solidyanmics extended to 3D.
- New point boundary conditions.
- New dummy city mesh and analysis 2D for soil dynamics.
- Automatic identification of FreeFEM and Gmsh during `./configure`.
- New flags for `-with-FreeFEM=` and `-with-Gmsh=` during `./configure`.
- New flag `-problem linear-elasticity|damage|elastodynamics|soildynamics` to define physics.

- New flag `-model` to set approximation for damage mechanics `hybrid-phase-field|Mazar`.
- Better energy splitting included Hybrid phase-field compressibility vs tensile energy condition.
- Introduce boundary conditions via `-dirichletconditions [int]` flag.
- Introduce point boundary conditions via `-dirichletpointcondition [int]` flag.
- Introduce traction boundary conditions via `-tractionconditions [int]` flag.

Changed

- Moved to FreeFEM 4.6.
- Moved to PETSc 13.13.
- Moved to C++ for preprocessing.
- Dirichlet conditions handled now by `-dirichletconditions [int]` flag, with `[int]` number of Dirichlet conditions.
- Traction conditions handled now by `-tractionconditions [int]` flag, with `[int]` number of traction conditions.

1.2.2 Version 1.8 - 2020-01-21

Added

- New soil dynamic module `-soildynamics`
- New paraxial element support in 2D.
- New timeplotting support `timepvd`
- New `-postprocess` option for postprocessing `u` , `v`, `a` , or `uav`.

Changed

- Moved to FreeFEM 4.4.2.
- Moved to PETSc 13.12.
- New simpler way of plotting `savevtk` in parallel with `append` flag for iterative solutions.
- VTU files get stored with a date and time stamp.
- New way of maintaining a logfile for all simulations (date,time,case,..) in `simulation-log.csv`.

1.2.3 Version 1.7 - 2019-11-08

Added

- New mesh reordering via Reverse Cuthill-Mackee via `-useRCM`.
- New quasi-static parallel solver (Extension of B.Masseron & G.Rastiello sequential version).
- New GFP plugin for Mazar's damage update for 2D/3D problems `GFPMazarsDamageUpdate(...)`.
- New MPI plotting routine `plotJustMeshMPI()`.
- New option `-fastmethod` to switch back to default variational formulation.
- New make flag for compiling on supercomputer.

Changed

- Changed variational formulation now using $\epsilon(u) : A : \epsilon(v)$.
- Using GFP becomes optional `-useGFP`.

- Better documentation via `.md` and `.html` files.
- Better plotting support for `PlotMPI()`.
- Moved to FreeFEM 4.4.

1.2.4 Version 1.6 - 2019-06-11

Added

- Dynamic linear solver in 2D and 3D parallel/sequential.
- New finite element variable for partition of unity for fixing integrals.

Changed

- Better documentation via `.md` and `.html` files.
- Correct quadrature order for faster computations.
- Major changes/splitting of `.script` files.

Removed

- Removed the `BoundaryAndSourceConditions.script` merged with `ControlParameters.script`.

Bugs

- Bug in integrals fixed.

1.2.5 Version 1.5 - 2019-05-29

Added

- Dynamic linear solver in 2D and 3D sequential.
- New meshes for dynamics tests `bar-dynamic.msh`.
- Checking modules `make check`.
- Faster sparsity pattern calculations.

Changed

- Better documentation via `.md` and `.html` files.
- Major restructuring of the codes.
- Moved to `automake` for solver installation.
- Mesh building via `make`.

Removed

- Removed the manufactured solution codes.

1.2.6 Version 1.4 - 2019-05-14

Added

- Fully vectorial finite element solver for phase-filed `-vectorial`.
- New `-supercomp` for avoiding xterm issues on super computers.
- New `MatViz()` function for matrix sparsity visualization.
- Introduced `GFP` plugin support (Go Fast Plugins).

Changed

- Elastic energy decomposition is now optional `-energydecomp`.
- Force calculation using integrals (Thanks to G.Rastiello).

1.2.7 Version 1.3 - 2019-04-08

Added

- New meshes in 2D/3D `Notched-plate` , `square-crack`, etc.
- New fracture mechanics module.
- New `-nonlinear` flag to activate phase-field model for brittle fracture.
- New `-timelog` for time logging the solver.
- New `-pipegnu` for GNUplot piping.

Changed

- Scripting now performed using `.script` files:
  - `BoundaryAndSourceConditions.script`
  - `LinearFormBuilderAndSolver.script`
  - `Macros.script`
  - `Main.script`
  - `VariationalFormulation.script`
  - ....
- Move to FreeFEM version 4.0.
- Move to PETSc version 3.11.

1.2.8 Version 1.2 - 2019-03-18

Added

- Support for Gmsh's `.msh` or Medit's `.mesh` meshes in folder `Meshes`.
- Advance to 3D physics.
- New MPI based parallel solver linear elasticity.
- New approach for solver generation via scripting (PhD thesis MA Badri) with `scriptGenerator.edp`.
- Integrated Domain decomposition macro (PhD thesis MA Badri).
- Customized `.vtk` support for ParaView post-processing.
- New point boundary condition macro `pointbc(Real[int], fespace, matrix)`.
- New flags for communicating with the solver: `-dimension`, `-plot`, `-bodyforce`, `-lagrange`, etc.

Changed

- More advance README.MD.
- Sequential solver now merged within scripting via flag `-sequential`.
- Move to FreeFEM version 3.62.
- Moved manufactured solutions to `validation-test` folder.

1.2.9 Version 1.1 - 2019-03-04

Added

- Initial FreeFEM files for sequential linear elasticity in 2D (case of constrained bar).
- More cases of manufactured solution for linear elasticity in 2D.
- Added `README.MD` for explaining the solver.
- ParaView plotting activated.

Changed

- Moved to Tuleap git hosting from CEA.
- Separate folder of manufactured solutions and the linear elastic solver.
- Move to FreeFEM version 3.61.

1.2.10 Version 1.0 - 2019-02-15

Added

- Initial FreeFEM files Method of manufactured solution for linear elasticity in 2D.

1.2.11 Version git tags

- | [1.0] |8a8ecb2746b7da792073358c60df33bae647f788 |
- | [1.1] |a667e6085ba1f92f8dd619bd40e18f85c593bc0a |
- | [1.2] |e48b7b3a30c05ad4c343efa6a17fee386031f437 |
- | [1.3] |39f43245503658498 52c5264b8d4535aae05e30d |
- | [1.4] |f51f678630eb9b2fed355e5cedf976ce8b5fa341 |
- | [1.5] |07293ba09a69d3d6a16278220a0b4a7a9f318f96 |
- | [1.6] |f359dd049fb1ddde376e8ad8e5177c663e430418 |
- | [1.7] |aee9bfec868a70b3d9974d7692bc19f9739ab7dc |
- | [1.8] |2f26292636c7248133e31ae912ee58113de2ef71 |

Chapter 2

Installation

PSD is cross-platform solver built to work with Linux, MacOs, and Windows platforms.

2.1 Dependencies

To install and work with PSD first check that you have installed all the dependencies. PSD depends on the following:

- `C++` (g++ version 4.8 or greater) (or Intel compiler)
- `automake`
- `FreeFEM`
- `PETSc` (optional)
- `Gmsh`
- `gnuplot` (optional)
- `git`

2.2 PSD installation steps

Now that I have all the dependencies what next ?

- Go ahead and grab the latest copy of PSD. The code is hosted on CEA's internal git repository.

```
git clone https://codev-tuleap.intra.cea.fr/plugins/git/hpcseism/freefem.git  PSD-Sources
```

- Now goto the `PSD-Sources` folder and autoconf PSD within the cloned folder

```
autoreconf -i
```

- Configure PSD within the cloned folder

```
./configure
```

  Note: `./configure will` install PSD in `$HOME/PSD` to change this directory use `-prefix=Your/Own/Path` with `./configure`.

- Make PSD directives

```
make FFINSTALLDIR=/usr/local/bin/ GMSH=/usr/bin/gmsh
```

Note: variable `FFINSTALLDIR` can be changed to the `FreeFEM` local path if installed locally. Similarly variable `GMSH` can be changed for `Gmsh`. Note: Please use the new version of `Gmsh` (greater than version 4.3.0) from their official website.

- Install PSD

```
make install
```

- Check if installation is correct

```
make check
```

Now you should have the solver at `$HOME/PSD`. To use the solver please go to `$HOME/PSD`.

2.3 Update PSD to the latest version

If you are a PSD user and would like to update your old PSD source to a new one. Go to your `PSD-Sources` folder and

```
git pull origin master
```

After this step simply

```
./reconfigure; make; make install; make check
```

2.4 Using PSD for brittle fracture

If you plan to use PSD for brittle fracture simulations, you must tweak your FreeFEM installation.

2.5 PSD installation for supercomputer

To install a copy of PSD that will be used on a supercomputer/cluster. During the `make` command of PSD from section 2.2 instead do

```
make supercomp
```

Note that the copy of PSD installed is not complete. You will have to manually build the static libraries on the supercomputer. Once your PSD copy is on supercomputer, go to `PSD/plugin` folder, and

```
make
```

Chapter 3

Theoretical background

3.1 Elastostatics

Let us consider d-dimensional domain $\Omega \in \mathbb{R}^d$ in a Euclidean referential $R(O, \mathbf{e}_i)$ (with $i = 1, \ldots, d$) submitted to a system of body forces $\mathbf{b}$. We denote $\partial\Omega$ the boundary of Ω and indicate with $\mathbf{n} = \mathbf{n}(\mathbf{x}) = n_i(\mathbf{x})\mathbf{e}_i$ its outer normal in any point $\mathbf{x} = x_i\mathbf{e}_i \in \partial\Omega$.

The problem to solve in order to characterize the dynamics equilibrium thus consists in finding a vector-valued displacement field $\mathbf{u} = \mathbf{u}(\mathbf{x}, t) : \Omega \times [0, T] \to \mathbb{R}^d$ regular "enough" and such that:

$$\begin{cases} \mathrm{div}\boldsymbol{\sigma} + \mathbf{b} = 0 & (\mathbf{x}, t) \in \Omega \times [0, T] \\ \boldsymbol{\sigma} = \boldsymbol{\sigma}(\mathbf{u}) & (\mathbf{x}, t) \in \Omega \times [0, T] \\ \mathbf{u} = \mathbf{u}^* & (\mathbf{x}, t) \in \partial_u\Omega \times [0, T] \\ \boldsymbol{\sigma} \cdot \mathbf{n} = \mathbf{t} & (\mathbf{x}, t) \in \partial_t\Omega \times [0, T] \end{cases} \tag{3.1}$$

where "div" denotes the divergence operator, symbol "$\cdot$" denotes the single contraction operation between tensors, $\rho = \rho(\mathbf{x}) : \Omega \to \mathbb{R}$ is the material density and $\boldsymbol{\sigma} = \boldsymbol{\sigma}(\mathbf{u})$ denotes a constitutive equation expressing the relationship between the second order Cauchy's stress tensor $\boldsymbol{\sigma} : \Omega \times [0, T] \to \mathbb{R}^{d\times d}$ and the displacement. Moreover, $\mathbf{u}^* = \mathbf{u}^*(\mathbf{x}, t) : \partial_u\Omega \times [0, T] \to \mathbb{R}^d$ is the imposed displacement field on $\partial_u\Omega$ (Dirichlet boundary condition) and $\mathbf{t} = \mathbf{t}(\mathbf{x}, t) : \partial_u\Omega \times [0, T] \to \mathbb{R}^d$ is the imposed traction vector on $\partial_t\Omega$ (Neumann boundary condition). The split of $\partial\Omega$ is such that $\partial\Omega = \overline{\partial_u\Omega \cup \partial_t\Omega}$ and $\partial_u\Omega \cap \partial_t\Omega = \emptyset$, with overline $\overline{\bullet}$ denoting the closure of set $\bullet$.

Let us now introduce the spaces of the admissible displacements fields ($\mathcal{U}$) and test functions ($\mathcal{V}$):

$$\begin{aligned} \mathcal{U} &= \left\{ \mathbf{u} = \mathbf{u}(\mathbf{x}, t) : \partial_u\Omega \times [0, T] \to \mathbb{R}^d \,|\, \mathbf{u} \in H^1(\Omega),\, \mathbf{u} = \mathbf{u}^*\, \partial_u\Omega \times [0, T] \right\} \\ \mathcal{V} &= \left\{ \mathbf{v} = \mathbf{v}(\mathbf{x}, t) : \partial_u\Omega \times [0, T] \to \mathbb{R}^d \,|\, \mathbf{v} \in H^1(\Omega),\, \mathbf{v} = 0\, \partial_u\Omega \times [0, T], \right\} \end{aligned} \tag{3.2}$$

The weak form of previous boundary value problem can be easily obtained by integrating by part the linear momentum balance equation using a test function $\mathbf{v} \in \mathcal{V}$, and imposing the Neumann boundary condition:

$$\underbrace{\int_\Omega \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\epsilon}(\mathbf{v})\, \mathrm{d}V}_{:=K(\mathbf{u},\mathbf{v})} = \underbrace{\int_\Omega \mathbf{b} \cdot \mathbf{v}\, \mathrm{d}V + \int_{\partial_t\Omega} \mathbf{t} \cdot \mathbf{v}\, \mathrm{d}S}_{:=b(\mathbf{v};\mathbf{b})+b(\mathbf{v};\mathbf{t})_{\partial_t\Omega}} \quad \forall\, \mathbf{v} \in \mathcal{V} \tag{3.3}$$

where symbol ":" is the double contraction operation between tensors, $K(\mathbf{u}, \mathbf{v})$ is the bi-linear symmetric form associated with the stiffness matrix and $b(\mathbf{v}; \mathbf{b}) + b(\mathbf{v}; \mathbf{t})_{\partial_t\Omega}$ are the linear forms associated with the external loading.[1]

[1] In the following of this document, given a known field $\mathbf{a}$, symbol $b(\mathbf{v}; \mathbf{a})$ will be used to denote the linear form $\int_\Omega \mathbf{a} \cdot \mathbf{v}\, \mathrm{d}V$, whereas $b(\mathbf{v}; \mathbf{a})_{Surf}$ will denote the linear form obtained form the surface integral $\int_{Surf} \mathbf{a} \cdot \mathbf{v}\, \mathrm{d}S$. Any linear form without down-script has to be interpreted as an integral over Ω. Only surface integrals will be defined explicitly.

The problem to solve can be finally written as:

$$\boxed{\begin{array}{l}\text{Find } \mathbf{u}\in\mathcal{U} \text{ such that :}\\ K(\mathbf{u},\mathbf{v}) = b(\mathbf{v};\mathbf{b}) + b(\mathbf{v};\mathbf{t})_{\partial_t\Omega} \quad \forall\,\mathbf{v}\in\mathcal{V}\end{array}} \tag{3.4}$$

3.2 Elastodynamics

The problem to solve in order to characterize the dynamics equilibrium thus consists in finding a vector-valued displacement field $\mathbf{u} = \mathbf{u}(\mathbf{x},t) : \Omega\times[0,T]\to\mathbb{R}^d$ regular "enough" and such that:

$$\begin{cases}\text{div}\boldsymbol{\sigma} + \mathbf{b} = \rho\ddot{\mathbf{u}} & (\mathbf{x},t)\in\Omega\times[0,T]\\ \boldsymbol{\sigma} = \boldsymbol{\sigma}(\mathbf{u}) & (\mathbf{x},t)\in\Omega\times[0,T]\\ \mathbf{u} = \mathbf{u}^* & (\mathbf{x},t)\in\partial_u\Omega\times[0,T]\\ \boldsymbol{\sigma}\cdot\mathbf{n} = \mathbf{t} & (\mathbf{x},t)\in\partial_t\Omega\times[0,T]\\ \mathbf{u} = \mathbf{u}_0 & \mathbf{x}\in\Omega,\ t=0\\ \dot{\mathbf{u}} = \dot{\mathbf{u}}_0 & \mathbf{x}\in\Omega,\ t=0\end{cases} \tag{3.5}$$

where "div" denotes the divergence operator, symbol "$\cdot$" denotes the single contraction operation between tensors, $\rho = \rho(\mathbf{x}) : \Omega\to\mathbb{R}$ is the material density, $\ddot{\mathbf{u}} = \ddot{\mathbf{u}}(\mathbf{x},t) = \mathbf{u}_{tt} : \Omega\times[0,T]\to\mathbb{R}^d$ is the acceleration field (i.e., the second time derivative of the field $\mathbf{u}$) and $\boldsymbol{\sigma} = \boldsymbol{\sigma}(\mathbf{u})$ denotes a constitutive equation expressing the relationship between the second order Cauchy's stress tensor $\boldsymbol{\sigma} : \Omega\times[0,T]\to\mathbb{R}^{d\times d}$ and the displacement. Moreover, $\mathbf{u}^* = \mathbf{u}^*(\mathbf{x},t) : \partial_u\Omega\times[0,T]\to\mathbb{R}^d$ is the imposed displacement field on $\partial_u\Omega$ (Dirichlet boundary condition) and $\mathbf{t} = \mathbf{t}(\mathbf{x},t) : \partial_u\Omega\times[0,T]\to\mathbb{R}^d$ is the imposed traction vector on $\partial_t\Omega$ (Neumann boundary condition). The split of $\partial\Omega$ is such that $\partial\Omega = \overline{\partial_u\Omega\cup\partial_t\Omega}$ and $\partial_u\Omega\cap\partial_t\Omega = \emptyset$, with overline $\overline{\bullet}$ denoting the closure of set $\bullet$. Finally, $\mathbf{u}_0 = \mathbf{u}_0(\mathbf{x},0) : \Omega\to\mathbb{R}^d$ and $\dot{\mathbf{u}}_0 = \dot{\mathbf{u}}_0(\mathbf{x},0) : \Omega\to\mathbb{R}^d$ are the displacement and velocity fields at time $t=0$ (initial conditions).

Let us now introduce the spaces of the admissible displacements fields ($\mathcal{U}$) and test functions ($\mathcal{V}$):

$$\begin{aligned}\mathcal{U} &= \left\{\mathbf{u} = \mathbf{u}(\mathbf{x},t) : \partial_u\Omega\times[0,T]\to\mathbb{R}^d \,|\, \mathbf{u}\in H^1(\Omega),\ \mathbf{u} = \mathbf{u}^*\,\partial_u\Omega\times[0,T],\ \mathbf{u}(\mathbf{x},0) = 0,\ \mathbf{u}(\dot{\mathbf{x}},0) = \dot{\mathbf{u}}_0\right\}\\ \mathcal{V} &= \left\{\mathbf{v} = \mathbf{v}(\mathbf{x},t) : \partial_u\Omega\times[0,T]\to\mathbb{R}^d \,|\, \mathbf{v}\in H^1(\Omega),\ \mathbf{v} = 0\,\partial_u\Omega\times[0,T],\right\}\end{aligned} \tag{3.6}$$

The weak form of previous boundary value problem can be easily obtained by integrating by part the linear momentum balance equation using a test function $\mathbf{v}\in\mathcal{V}$, and imposing the Neumann boundary condition:

$$\underbrace{\int_\Omega \rho\ddot{\mathbf{u}}\cdot\mathbf{v}\,\mathrm{d}V}_{:=M(\ddot{\mathbf{u}},\mathbf{v})} + \underbrace{\int_\Omega \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\epsilon}(\mathbf{v})\,\mathrm{d}V}_{:=K(\mathbf{u},\mathbf{v})} = \underbrace{\int_\Omega \mathbf{b}\cdot\mathbf{v}\,\mathrm{d}V + \int_{\partial_t\Omega}\mathbf{t}\cdot\mathbf{v}\,\mathrm{d}S}_{:=b(\mathbf{v};\mathbf{b})+b(\mathbf{v};\mathbf{t})_{\partial_t\Omega}} \quad \forall\,\mathbf{v}\in\mathcal{V} \tag{3.7}$$

where symbol ":" is the double contraction operation between tensors, $M(\ddot{\mathbf{u}},\mathbf{v})$ is the bi-linear symmetric form associated with inertial terms (i.e., dependent on the mass matrix), $K(\mathbf{u},\mathbf{v})$ is the bi-linear symmetric form associated with the stiffness matrix and $b(\mathbf{v};\mathbf{b}) + b(\mathbf{v};\mathbf{t})_{\partial_t\Omega}$ are the linear forms associated with the external loading.[2]

The problem to solve can be finally written as:

$$\boxed{\begin{array}{l}\text{Find } \mathbf{u}\in\mathcal{U} \text{ such that :}\\ M(\ddot{\mathbf{u}},\mathbf{v}) + K(\mathbf{u},\mathbf{v}) = b(\mathbf{v};\mathbf{b}) + b(\mathbf{v};\mathbf{t})_{\partial_t\Omega} \quad \forall\,\mathbf{v}\in\mathcal{V}\end{array}} \tag{3.8}$$

[2] In the following of this document, given a known field $\mathbf{a}$, symbol $b(\mathbf{v};\mathbf{a})$ will be used to denote the linear form $\int_\Omega \mathbf{a}\cdot\mathbf{v}\,\mathrm{d}V$, whereas $b(\mathbf{v};\mathbf{a})_{Surf}$ will denote the linear form obtained form the surface integral $\int_{Surf}\mathbf{a}\cdot\mathbf{v}\,\mathrm{d}S$. Any linear form without down-script has to be interpreted as an integral over Ω. Only surface integrals will be defined explicitly.

The only way for accounting form dumping effects in this formulation is through a proper definition of a constitutive law $\boldsymbol{\sigma} = \boldsymbol{\sigma}(\mathbf{u})$ modeling dissipative processes occurring at the material level. In some cases, however, it can be useful to account for damping effects in a more global way. This can be done by modifying the variational problem as follows:

$$\boxed{\begin{array}{l}\text{Find } \mathbf{u}\in\mathcal{U} \text{ such that :}\\ M(\ddot{\mathbf{u}},\mathbf{v}) + C(\dot{\mathbf{u}},\mathbf{v}) + K(\mathbf{u},\mathbf{v}) = b(\mathbf{v};\mathbf{b}) + b(\mathbf{v};\mathbf{t})_{\partial_t\Omega} \quad \forall\,\mathbf{v}\in\mathcal{V}\end{array}} \tag{3.9}$$

where $C(\dot{\mathbf{u}},\mathbf{v})$ is an additional bi-linear symmetric form associated with damping/viscous effects.

3.3 Time discretization

Time discretized variational formulations are illustrate in this subsection, considering several implicit time integration schemes. Representative members of these algorithms are, among others, the N$-\beta$ method [12], the HHT$-\alpha$ method [7], the WBZ$-\alpha$ method [13], the HP$-\theta_1$ method [8] and the CH$-\alpha$ method [3]. These methods exhibit second order accuracy in linear dynamics and permit efficient variable step size techniques, being one-step methods. The CH$-\alpha$, the HHT$-\alpha$ and the WBZ$-\alpha$ methods, the so called $\alpha-$methods, are one-parameter schemes which can be considered as particular cases of a more general class of methods named generalized $-\ \alpha$ (G $-\ \alpha$). This class of methods corresponds to the CH$-\alpha$ scheme [3], where the algorithmic parameters α_m, α_f, β and γ are assumed to be independent of each other.

3.3.1 Generalized-α method

The Generalized $-\ \alpha$ (G $-\ \alpha$) is an implicit method that allows for high frequency energy dissipation, reduced unwanted low-frequency dissipation, and second order accuracy (i.e., Δt^2), both in linear and nonlinear regimes. Depending on choices of input parameters, unconditionally stability can be achieved for linear problems (as for all implicit schemes). Stability properties for nonlinear problem were studied in [6]. In the latter work, the second-order accuracy of this class of algorithms was proved also in the non-linear regime, independently of the quadrature rule for non-linear internal forces. Conversely, the G-stability notion which is suitable for linear multi-step schemes devoted to non-linear dynamic problems cannot be applied, as the non-linear structural dynamics equations are not contractive. Nonetheless, [6] proved that the G $-\ \alpha$ methods are stable in an energy sense, and guarantee energy decay for high-frequencies and asymptotic cancellation. However, overshoot and heavy energy oscillations in the intermediate-frequency range are exhibited.

Problem setting

Let introduce a time discretization of the time interval $[0,T]$ in an ordered sequence of $N+1$ time increments $(0,\dots,t_i,t_{i+1},\dots,T)$ such that $t_{i+1} = t_i + \Delta t$, with $\Delta t = T/N$ denoting the time step (here supposed constant). According to the (G $-\ \alpha$) method, the dynamic evolution equation is solved at intermediate time $t_{n+1-\alpha} \in [t_n, t_{n+1}]$. The following notation is used to denote the value of a generic variable z at time $t_{n+1-\alpha}$:

$$z_{n+1-\alpha} = (1-\alpha)z_{n+1} + \alpha z_n \quad \text{with} \quad \alpha\in[0,1] \tag{3.10}$$

Furthermore, the following approximations (standard for Newmark schemes) for the displacement and velocity fields at time t_{n+1} are used [12]:

$$\begin{aligned}\mathbf{u}_{n+1} &= \overline{\mathbf{u}}_{n+1} + \beta\,\Delta t^2\,\ddot{\mathbf{u}}_{n+1}\\ \dot{\mathbf{u}}_{n+1} &= \dot{\overline{\mathbf{u}}}_{n+1} + \gamma\,\Delta t\,\ddot{\mathbf{u}}_{n+1}\end{aligned} \tag{3.11}$$

where $\overline{\mathbf{u}}_{n+1}$ and $\dot{\overline{\mathbf{u}}}_{n+1}$ are the following known contributions (predictions in predictor-corrector schemes):

$$\begin{aligned}\overline{\mathbf{u}}_{n+1} &= \mathbf{u}_n + \Delta t\,\dot{\mathbf{u}}_n + \Delta t^2\left(\frac{1}{2}-\beta\right)\ddot{\mathbf{u}}_n\\ \dot{\overline{\mathbf{u}}}_{n+1} &= \dot{\mathbf{u}}_n + \Delta t\,(1-\gamma)\ddot{\mathbf{u}}_n\end{aligned} \tag{3.12}$$

and (β, γ) are algorithmic parameters. By inverting the first equation of (3.11), one can express $\ddot{\mathbf{u}}_{n+1}$ as a function of $\mathbf{u}_{n+1}$ as:

$$\ddot{\mathbf{u}}_{n+1} = \frac{1}{\beta\,\Delta t^2}\left(\mathbf{u}_{n+1} - \overline{\mathbf{u}}_{n+1}\right) \tag{3.13}$$

3.3.2 Time discretized variational problem (no damping)

Neglecting damping effects, the problem to solve is written as:

$$\boxed{\begin{array}{l}\text{Find } \mathbf{u}_{n+1} \in \mathcal{U} \text{ such that :} \\ M(\ddot{\mathbf{u}}_{n+1-\alpha_m}, \mathbf{v}) + K(\mathbf{u}_{n+1-\alpha_f}, \mathbf{v}) = b(\mathbf{v}; \mathbf{b}) + b(\mathbf{v}; \mathbf{t}_{n+1-\alpha_f})_{\partial_t\Omega} \quad \forall\, \mathbf{v} \in \mathcal{V}\end{array}} \tag{3.14}$$

where $\ddot{\mathbf{u}}_{n+1-\alpha_m}$ and $\mathbf{u}_{n+1-\alpha_f}$ can be written according to (3.10):

$$\begin{aligned}\ddot{\mathbf{u}}_{n+1-\alpha_m} &= \frac{1-\alpha_m}{\beta\,\Delta t^2}\left(\mathbf{u}_{n+1} - \overline{\mathbf{u}}_{n+1}\right) + \alpha_m \ddot{\mathbf{u}}_n \\ \mathbf{u}_{n+1-\alpha_f} &= (1-\alpha_f)\mathbf{u}_{n+1} + \alpha_f \mathbf{u}_n\end{aligned} \tag{3.15}$$

Furthermore, parameters β and γ read:

$$\gamma = \frac{1}{2} + \alpha_f - \alpha_m \qquad \beta = \frac{1}{4}\left(\gamma + \frac{1}{2}\right)^2 \tag{3.16}$$

Bilinear and linear operators. Using equation (3.13), one can easily write the bi-linear part associated with the mass matrix in terms of the unknown displacement $\mathbf{u}_{n+1}$ as follows:

$$M(\ddot{\mathbf{u}}_{n+1-\alpha_m}, \mathbf{v}) = \frac{1-\alpha_m}{\beta\,\Delta t^2} M(\mathbf{u}_{n+1}, \mathbf{v}) - \frac{1-\alpha_m}{\beta\,\Delta t^2} m(\mathbf{v}; \overline{\mathbf{u}}_{n+1}) + \alpha_m m(\mathbf{v}; \ddot{\mathbf{u}}_n) \tag{3.17}$$

where linear forms $m(\mathbf{v}; \overline{\mathbf{u}}_{n+1})$ and $m(\mathbf{v}; \ddot{\mathbf{u}}_n)$ read:[3]

$$m(\mathbf{v}; \overline{\mathbf{u}}_{n+1}) = \int_\Omega \rho \overline{\mathbf{u}}_{n+1} \cdot \mathbf{v}\, \mathrm{d}V \qquad m(\mathbf{v}; \ddot{\mathbf{u}}_n) = \int_\Omega \rho \ddot{\mathbf{u}}_n \cdot \mathbf{v}\, \mathrm{d}V \tag{3.19}$$

Term $m(\mathbf{v}; \overline{\mathbf{u}}_{n+1})$ figuring in equation (3.17) can also be expanded as:

$$m(\mathbf{v}; \overline{\mathbf{u}}_{n+1}) = m(\mathbf{v}; \mathbf{u}_n) + \Delta t\, m(\mathbf{v}; \dot{\mathbf{u}}_n) + \Delta t^2 \left(\frac{1}{2} - \beta\right) m(\mathbf{v}; \ddot{\mathbf{u}}_n) \tag{3.20}$$

As a consequence (3.17) can be rewritten as:[4]

$$M(\ddot{\mathbf{u}}_{n+1-\alpha_m}, \mathbf{v}) = \frac{1-\alpha_m}{\beta\,\Delta t^2} M(\mathbf{u}_{n+1}, \mathbf{v}) - \frac{1-\alpha_m}{\beta\,\Delta t^2} m(\mathbf{v}; \mathbf{u}_n) - \frac{1-\alpha_m}{\beta\,\Delta t} m(\mathbf{v}; \dot{\mathbf{u}}_n) + \left(1 - \frac{1-\alpha_m}{2\beta}\right) m(\mathbf{v}; \ddot{\mathbf{u}}_n) \tag{3.22}$$

In a similar way, we can rewrite the bi-linear form associated with the stiffness matrix as:

$$K(\mathbf{u}_{n+1-\alpha_f}, \mathbf{v}) = (1-\alpha_f) K(\mathbf{u}_{n+1}, \mathbf{v}) + \alpha_f k(\mathbf{v}; \mathbf{u}_n) \tag{3.23}$$

[3] More in general, given a field $\mathbf{a} = \mathbf{a}(\mathbf{x}) : \Omega \to \mathbb{R}^d$, $m(\mathbf{v}; \mathbf{a})$ denotes the linear form:

$$m(\mathbf{v}; \mathbf{a}) = \int_\Omega \rho \mathbf{a} \cdot \mathbf{v}\, \mathrm{d}V \tag{3.18}$$

[4] When summing up the terms depending on $\ddot{\mathbf{u}}_n$, coming from the definition of $\overline{\mathbf{u}}_{n+1}$ and from equation (3.17), we have:

$$-\left[(1-\alpha_m)\left(\frac{1-2\beta}{2\beta}\right) - \alpha_m\right] = -\frac{(1-\alpha_m)(1-2\beta) - 2\beta\alpha_m}{2\beta} = -\frac{1-2\beta-\alpha_m+2\beta\alpha_m-2\beta\alpha_m}{2\beta} = 1 - \frac{1-\alpha_m}{2\beta} \tag{3.21}$$

where $k(\mathbf{v};\mathbf{u}_n)$ is the linear form:[5]

$$k(\mathbf{v};\mathbf{u}_n) = \int_\Omega \boldsymbol{\sigma}\left(\mathbf{u}_n\right) : \boldsymbol{\epsilon}(\mathbf{v})\,\mathrm{d}V \tag{3.25}$$

Finally, the linear form $b(\mathbf{v};\mathbf{t}_{n+1-\alpha_f})_{\partial_t\Omega}$ becomes:

$$b(\mathbf{v};\mathbf{t}_{n+1-\alpha_f})_{\partial_t\Omega} = (1-\alpha_f)b(\mathbf{v};\mathbf{t}_{n+1})_{\partial_t\Omega} + \alpha_f b(\mathbf{v};\mathbf{t}_n)_{\partial_t\Omega} \tag{3.26}$$

Final variational problem. The time discretized variational formulation to solve becomes:

$$\boxed{\begin{array}{l}\text{Find } \mathbf{u}_{n+1}\in\mathcal{U} \text{ such that :}\\ \widetilde{K}(\mathbf{u}_{n+1},\mathbf{v}) = \widetilde{l}(\mathbf{v})\end{array}} \tag{3.27}$$

where $\widetilde{K}(\mathbf{u}_{n+1},\mathbf{v})$ is the bi-linear form associated with the effective/equivalent stiffness matrix:

$$\widetilde{K}(\mathbf{u}_{n+1},\mathbf{v}) = \frac{1-\alpha_m}{\beta\,\Delta t^2}M(\mathbf{u}_{n+1},\mathbf{v}) + (1-\alpha_f)K(\mathbf{u}_{n+1},\mathbf{v}) \tag{3.28}$$

and $\widetilde{l}(\mathbf{v}) = \widetilde{l}(\mathbf{v};\{\mathbf{b},\mathbf{t}_n,\mathbf{t}_{n+1},\mathbf{u}_n,\dot{\mathbf{u}}_n,\ddot{\mathbf{u}}_n\})$ is the following linear form:

$$\begin{aligned}\widetilde{l}(\mathbf{v}) = b(\mathbf{v};\mathbf{b}) + b(\mathbf{v};\mathbf{t}_{n+1-\alpha_f})_{\partial_t\Omega} + \frac{1-\alpha_m}{\beta\,\Delta t^2}m(\mathbf{v},\mathbf{u}_n) + \frac{1-\alpha_m}{\beta\,\Delta t}m(\mathbf{v};\dot{\mathbf{u}}_n)\\ \cdots + \left(1-\frac{1-\alpha_m}{2\beta}\right)m(\mathbf{v};\ddot{\mathbf{u}}_n) - \alpha_f k(\mathbf{v};\mathbf{u}_n)\end{aligned} \tag{3.29}$$

3.3.3 Time discretized variational problem (Rayleigh damping)

The problem to solve is now:

$$\boxed{\begin{array}{l}\text{Find } \mathbf{u}_{n+1}\in\mathcal{U} \text{ such that :}\\ M(\ddot{\mathbf{u}}_{n+1-\alpha_m},\mathbf{v}) + C(\dot{\mathbf{u}}_{n+1-\alpha_f},\mathbf{v}) + K(\mathbf{u}_{n+1-\alpha_f},\mathbf{v}) = b(\mathbf{v};\mathbf{b}) + b(\mathbf{v};\mathbf{t}_{n+1-\alpha_f})_{\partial\Omega} \quad \forall\,\mathbf{v}\in\mathcal{V}\end{array}} \tag{3.30}$$

where, following a simple Rayleigh formulation, the bi-linear form associated with the damping matrix can be written as:

$$C(\dot{\mathbf{u}}_{n+1-\alpha_f},\mathbf{v}) = \eta_M M(\dot{\mathbf{u}}_{n+1-\alpha_f},\mathbf{v}) + \eta_K K(\dot{\mathbf{u}}_{n+1-\alpha_f},\mathbf{v}) \tag{3.31}$$

with (η_M,η_K) denoting two positive model parameters.

Now, using definitions (3.10), (3.11) and (3.12), $\dot{\mathbf{u}}_{n+1-\alpha_f}$ can be written as:[6]

$$\dot{\mathbf{u}}_{n+1-\alpha_f} = \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}\mathbf{u}_{n+1} + (1-\alpha_f)\dot{\overline{\mathbf{u}}}_{n+1} - \frac{\gamma(1-\alpha_f)}{\beta\,\Delta t}\overline{\mathbf{u}}_{n+1} + \alpha_f\dot{\mathbf{u}}_n \tag{3.33}$$

[5] More in general, given a field $\mathbf{a} = \mathbf{a}(\mathbf{x}) : \Omega \to \mathbb{R}^d$, $k(\mathbf{v};\mathbf{a})$ denotes the linear form:

$$k(\mathbf{v};\mathbf{a}) = \int_\Omega \boldsymbol{\sigma}\left(\mathbf{a}\right) : \boldsymbol{\epsilon}(\mathbf{v})\,\mathrm{d}V \tag{3.24}$$

[6] Using definitions (3.10), (3.11) and (3.12), the velocity field at time $t_{n+1-\alpha_f}$ reads:

$$\begin{aligned}\dot{\mathbf{u}}_{n+1-\alpha_f} &= (1-\alpha_f)\dot{\mathbf{u}}_{n+1} + \alpha_f\dot{\mathbf{u}}_n\\ &= (1-\alpha_f)\dot{\overline{\mathbf{u}}}_{n+1} + \alpha_f\dot{\mathbf{u}}_n + \gamma\,\Delta t\,(1-\alpha_f)\ddot{\mathbf{u}}_{n+1}\\ &= \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}\mathbf{u}_{n+1} + (1-\alpha_f)\dot{\overline{\mathbf{u}}}_{n+1} - \frac{\gamma(1-\alpha_f)}{\beta\,\Delta t}\overline{\mathbf{u}}_{n+1} + \alpha_f\dot{\mathbf{u}}_n\end{aligned} \tag{3.32}$$

or, using definitions (3.12), as:[7]

$$\dot{\mathbf{u}}_{n+1-\alpha_f} = \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}\mathbf{u}_{n+1} - \frac{\gamma(1-\alpha_f)}{\beta\,\Delta t}\mathbf{u}_n - \left[\frac{\gamma(1-\alpha_f)}{\beta} - 1\right]\dot{\mathbf{u}}_n - \Delta t(1-\alpha_f)\left(\frac{\gamma}{2\beta} - 1\right)\ddot{\mathbf{u}}_n \tag{3.35}$$

Bilinear and linear operators. Operator $M(\dot{\mathbf{u}}_{n+1-\alpha_f}, \mathbf{v})$ reads:

$$\begin{aligned} M(\dot{\mathbf{u}}_{n+1-\alpha_f}, \mathbf{v}) = &\frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}M(\mathbf{u}_{n+1}, \mathbf{v}) - \frac{\gamma(1-\alpha_f)}{\beta\,\Delta t}m(\mathbf{v}; \mathbf{u}_n) \\ &\cdots - \left[\frac{\gamma(1-\alpha_f)}{\beta} - 1\right]m(\mathbf{v}; \dot{\mathbf{u}}_n) - \Delta t(1-\alpha_f)\left(\frac{\gamma}{2\beta} - 1\right)m(\mathbf{v}; \ddot{\mathbf{u}}_n) \end{aligned} \tag{3.36}$$

Similarly, the stiffness contribution becomes:

$$\begin{aligned} K(\dot{\mathbf{u}}_{n+1-\alpha_f}, \mathbf{v}) = &\frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}K(\mathbf{u}_{n+1}, \mathbf{v}) - \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}k(\mathbf{v}; \mathbf{u}_n) \\ &\cdots - \left[\frac{\gamma(1-\alpha_f)}{\beta} - 1\right]k(\mathbf{v}; \dot{\mathbf{u}}_n) - \Delta t(1-\alpha_f)\left(\frac{\gamma}{2\beta} - 1\right)k(\mathbf{v}; \ddot{\mathbf{u}}_n) \end{aligned} \tag{3.37}$$

Final variational problem. Finally, the variational problem to solve reads:

$$\boxed{\begin{array}{l} \text{Find } \mathbf{u}_{n+1} \in \mathcal{U} \text{ such that :} \\ \widetilde{\widetilde{K}}(\mathbf{u}_{n+1}, \mathbf{v}) = \widetilde{\widetilde{l}}(\mathbf{v}) \quad \forall\, \mathbf{v} \in \mathcal{V} \end{array}} \tag{3.38}$$

where $\widetilde{\widetilde{K}}(\mathbf{u}_{n+1}, \mathbf{v})$ is the bi-linear form associated with the effective stiffness matrix:

$$\begin{aligned} \widetilde{\widetilde{K}}(\mathbf{u}_{n+1}, \mathbf{v}) &= \widetilde{K}(\mathbf{u}_{n+1}, \mathbf{v}) + \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}C(\mathbf{u}_{n+1}, \mathbf{v}) \\ &= \frac{1-\alpha_m}{\beta\Delta t^2}M(\mathbf{u}_{n+1}, \mathbf{v}) + \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}C(\mathbf{u}_{n+1}, \mathbf{v}) + (1-\alpha_f)\,K(\mathbf{u}_{n+1}, \mathbf{v}) \end{aligned} \tag{3.39}$$

with $C(\mathbf{u}_{n+1}, \mathbf{v})$ denoting the Rayleigh damping operator:

$$C(\mathbf{u}_{n+1}, \mathbf{v}) = \eta_M M(\mathbf{u}_{n+1}, \mathbf{v}) + \eta_K K(\mathbf{u}_{n+1}, \mathbf{v}) \tag{3.40}$$

and $\widetilde{\widetilde{l}}(\mathbf{v}) = \widetilde{\widetilde{l}}(\mathbf{v}; \{\mathbf{b}, \mathbf{t}_n, \mathbf{t}_{n+1}, \mathbf{u}_n, \dot{\mathbf{u}}_n, \ddot{\mathbf{u}}_n\})$ being the following linear form:

$$\begin{aligned} \widetilde{\widetilde{l}}(\mathbf{v}) = &\widetilde{l}(\mathbf{v}) + \Delta t(1-\alpha_f)\left(\frac{\gamma}{2\beta} - 1\right)c(\mathbf{v}; \ddot{\mathbf{u}}_n) \\ &\cdots + \left[\frac{\gamma(1-\alpha_f)}{\beta} - 1\right]c(\mathbf{v}; \dot{\mathbf{u}}_n) + \frac{\gamma(1-\alpha_f)}{\beta\,\Delta t}c(\mathbf{v}; \mathbf{u}_n) \end{aligned} \tag{3.41}$$

In previous equation we introduced the following notation:

$$c(\mathbf{v}; \mathbf{a}) = \eta_M m(\mathbf{v}; \mathbf{a}) + \eta_K k(\mathbf{v}; \mathbf{a}) \tag{3.42}$$

[7] Using definitions (3.12) one obtains:

$$\begin{aligned} \dot{\mathbf{u}}_{n+1-\alpha_f} &= \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}\mathbf{u}_{n+1} + (1-\alpha_f)\,[\dot{\mathbf{u}}_n + \Delta t\,(1-\gamma)\ddot{\mathbf{u}}_n] - \frac{\gamma(1-\alpha_f)}{\beta\,\Delta t}\left[\mathbf{u}_n + \Delta t\,\dot{\mathbf{u}}_n + \Delta t^2\left(\frac{1}{2} - \beta\right)\ddot{\mathbf{u}}_n\right] + \alpha_f\dot{\mathbf{u}}_n \\ &= \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}\mathbf{u}_{n+1} + \left[1 - \frac{\gamma(1-\alpha_f)}{\beta}\right]\dot{\mathbf{u}}_n + (1-\alpha_f)\Delta t\left\{1 - \gamma\left[1 + \left(\frac{1-2\beta}{2\beta}\right)\right]\right\}\ddot{\mathbf{u}}_n - \frac{\gamma(1-\alpha_f)}{\beta\,\Delta t}\mathbf{u}_n \\ &= \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}\mathbf{u}_{n+1} - \frac{\gamma(1-\alpha_f)}{\beta\,\Delta t}\mathbf{u}_n - \left[\frac{\gamma(1-\alpha_f)}{\beta} - 1\right]\dot{\mathbf{u}}_n - \Delta t(1-\alpha_f)\left(\frac{\gamma}{2\beta} - 1\right)\ddot{\mathbf{u}}_n \end{aligned} \tag{3.34}$$

3.3.4 Implicit N$-\beta$ and HHT$-\alpha$ method as special cases

Newmark. One can easily show that, the Newmark scheme is obtained by choosing $\alpha_m = \alpha_f = 0$.

Without damping, the stiffness matrix becomes:

$$\widetilde{K}(\mathbf{u}_{n+1}, \mathbf{v}) = \frac{1}{\beta \Delta t^2} M(\mathbf{u}_{n+1}, \mathbf{v}) + K(\mathbf{u}_{n+1}, \mathbf{v}) \tag{3.43}$$

whereas the linear form simplifies as follows:

$$\begin{aligned} \widetilde{l}(\mathbf{v}) &= b(\mathbf{v}; \mathbf{b}) + b(\mathbf{v}; \mathbf{t}_{n+1})_{\partial_t \Omega} + \frac{1}{\beta\, \Delta t^2} m(\mathbf{v}; \overline{\mathbf{u}}_{n+1}) \\ &= b(\mathbf{v}; \mathbf{b}) + b(\mathbf{v}; \mathbf{t}_{n+1})_{\partial_t \Omega} + \frac{1}{\beta\, \Delta t^2} \left[m(\mathbf{v}; \mathbf{u}_n) + \Delta t\, m(\mathbf{v}; \dot{\mathbf{u}}_n) + \Delta t^2 \left(\frac{1}{2} - \beta \right) m(\mathbf{v}; \ddot{\mathbf{u}}_n) \right] \end{aligned} \tag{3.44}$$

When Rayleigh damping is considered, the bi-linear operator $\widetilde{\widetilde{K}}(\mathbf{u}_{n+1}, \mathbf{v})$ becomes:

$$\widetilde{\widetilde{K}}(\mathbf{u}_{n+1}, \mathbf{v}) = \frac{1}{\beta \Delta t^2} M(\mathbf{u}_{n+1}, \mathbf{v}) + \frac{\gamma}{\beta\, \Delta t} C(\mathbf{u}_{n+1}, \mathbf{v}) + K(\mathbf{u}_{n+1}, \mathbf{v}) \tag{3.45}$$

whereas the linear form simplifies as follows:

$$\widetilde{\widetilde{l}}(\mathbf{v}) = \widetilde{l}(\mathbf{v}) + \Delta t \left(\frac{\gamma}{2\beta} - 1 \right) c(\mathbf{v}; \ddot{\mathbf{u}}_n) + \left(\frac{\gamma}{\beta} - 1 \right) c(\mathbf{v}; \dot{\mathbf{u}}_n) + \frac{\gamma}{\beta\, \Delta t} c(\mathbf{v}; \mathbf{u}_n) \tag{3.46}$$

HHT. One can also show that HHT$-\alpha$ [7] method is recovered for $\alpha_m = 0$. Such formulation is not detailed in the following of this document, since it is less used than the classic Newmark approach.

3.3.5 Considerations on methods based upon operator splitting

In order to introduce predictor-correction, implicit-explicit and more in general schemes based upon operator splitting, one can rewrite displacement and velocity in a predictor-correction fashion as in (3.11) and (3.12), where (3.12) now defines predictors and (3.11) correctors (for more general information, the interested reader can refer to [9, 14]). For instance, a simple explicit predictor-corrector method can be obtained through solving problem (3.30) with $K(\bar{\mathbf{u}}_{n+1}, \mathbf{v})$ and $C((1-\alpha_f)\dot{\bar{\mathbf{u}}}_{n+1} + \alpha_f \mathbf{u}_n)$. Mixed implicit-explicit predictor-corrector methods can also be obtained through splitting Ω into two subdomains and using different time-integration schemes for solving the dynamic equilibrium problem on each of them.

3.4 Space discretization

Space discretization is performed according to the standard finite element method. The computational domain Ω is thus discretized into a mesh Ω^h comprising a finite number (n_{el}) of subdomains, the finite element Ω_e^h, such that $\Omega \approx \Omega^h = \cup_{e=1}^{n_{el}} \Omega_e^h$. Inside each element, the displacement field is interpolated based on nodal displacements ($\mathbf{d}$) through the shape functions matrix ($\mathbf{N}$), i.e., $\mathbf{u} \approx \mathbf{u}^h = \mathbf{N}(\mathbf{x})\mathbf{d}$. As usual, gradient terms are interpolated using the derivatives of the shape functions, i.e., $\boldsymbol{\epsilon} \approx \boldsymbol{\epsilon}^h = \mathbf{B}(\mathbf{x})\mathbf{d}$.

In a standard matrix format, after spatial discretization of the displacement field, the problem to solve can be written in the standard form as:

$$\boxed{\begin{aligned} &\text{Find } \mathbf{d}_{n+1} \text{ such that :} \\ &\left[\frac{1-\alpha_m}{\beta \Delta t^2} \mathbf{M} + (1-\alpha_f)\, \mathbf{K} \right] \mathbf{d}_{n+1} \\ &= \mathbf{f}_{n+1-\alpha_f} + \mathbf{M} \left[\left(\frac{1-\alpha_m}{2\beta} - 1 \right) \ddot{\mathbf{d}}_n + \frac{1-\alpha_m}{\beta\, \Delta t} \dot{\mathbf{d}}_n + \frac{1-\alpha_m}{\beta\, \Delta t^2} \mathbf{d}_n \right] - \alpha_f \mathbf{K} \mathbf{d}_n \end{aligned}} \tag{3.47}$$

where $\mathbf{M}$ and $\mathbf{K}$ are now the mass and stiffness matrices. They are obtained through assembling (operator $\mathbf{A}$) the corresponding elemental operators over the finite element mesh as:

$$\begin{aligned}\mathbf{M} &= \mathop{\mathbf{A}}_{e=1}^{n_{el}} \int_{\Omega_e} \rho \mathbf{N}^\top \mathbf{N} \mathrm{d}V \\ \mathbf{K} &= \mathop{\mathbf{A}}_{e=1}^{n_{el}} \int_{\Omega_e} \mathbf{B}^\top \mathbf{D} \mathbf{B} \mathrm{d}V\end{aligned} \tag{3.48}$$

where $\mathbf{D}$ is the material stiffness matrix defining the link between the stress and strain tensors (or between theirs rates of variation – more details are given in the next section).

When Rayleigh damping is considered the problem to solve is written as:

$$\begin{aligned}&\text{Find } \mathbf{d}_{n+1} \text{ such that :} \\ &\left[\frac{1-\alpha_m}{\beta \Delta t^2}\mathbf{M} + \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}\mathbf{C} + (1-\alpha_f)\,\mathbf{K}\right]\mathbf{d}_{n+1} \\ &= \mathbf{f}_{n+1-\alpha_f} + \mathbf{M}\left[\left(\frac{1-\alpha_m}{2\beta}-1\right)\ddot{\mathbf{d}}_n + \frac{1-\alpha_m}{\beta\,\Delta t}\dot{\mathbf{d}}_n + \frac{1-\alpha_m}{\beta\,\Delta t^2}\mathbf{d}_n\right] - \alpha_f \mathbf{K}\mathbf{d}_n \\ &+ \mathbf{C}\left\{\Delta t(1-\alpha_f)\left(\frac{\gamma}{2\beta}-1\right)\ddot{\mathbf{d}}_n + \left[\frac{\gamma(1-\alpha_f)}{\beta}-1\right]\dot{\mathbf{d}}_n + \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t}\mathbf{d}_n\right\}\end{aligned} \tag{3.49}$$

where matrix $\mathbf{C}$ is now defined as:

$$\mathbf{C} = \eta_M \mathbf{M} + \eta_K \mathbf{K} \tag{3.50}$$

3.5 Linear and nonlinear dynamic solvers

Elastodynamics is the simplest case one can encounter in structural mechanics. In that case, the space-time discretized linear system of equations is linear and finding the solution at any time t_{n+1} is straightforward. In most applications, however, material behavior is nonlinear since structural materials often dissipate energy and exhibit damage, permanent strains, etc. In that case, the resulting discretized problem to solve is nonlinear, and Newton–Raphson procedures can be used. In that case, the solution is found iteratively through solving a series of linearized problems.

3.5.1 Linear case - linear elastic material behavior

Let us start from the linear case first. Under small strains conditions, if the material is assumed isotropic linear elastic, the Cauchy's stress tensor reads $\boldsymbol{\sigma} = \lambda \mathrm{tr}(\boldsymbol{\epsilon})\mathbf{I} + 2\mu\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} = (\nabla\mathbf{u} + \nabla^\top\mathbf{u})/2$ is the small strain tensor (i.e., the symmetric part of the displacement gradient $\nabla\mathbf{u}$), $\lambda = \lambda(\mathbf{x})$ and $\mu = \mu(\mathbf{x})$ are the Lame's parameters and $\mathbf{I} = \delta_{ij}\mathbf{e}_i \otimes \mathbf{e}_j$ denotes the second order identity tensor. As a consequence, the bi-linear form $K(\mathbf{u}_{n+1}, \mathbf{v})$ and the corresponding linear form $k(\mathbf{v}; \mathbf{u}_n)$ can be rewritten in a more explicit form as:

$$\begin{aligned}K(\mathbf{u}_{n+1}, \mathbf{v}) &= \int_\Omega \boldsymbol{\sigma}\left(\mathbf{u}_{n+1}\right) : \boldsymbol{\epsilon}(\mathbf{v})\,\mathrm{d}V = \int_\Omega \left[\lambda\,\mathrm{tr}\,\boldsymbol{\epsilon}(\mathbf{u}_{n+1})\mathbf{I} + 2\mu\boldsymbol{\epsilon}(\mathbf{u}_{n+1})\right] : \boldsymbol{\epsilon}(\mathbf{v})\,\mathrm{d}V = \int_\Omega \boldsymbol{\epsilon}(\mathbf{u}_{n+1}) : \mathbb{E} : \boldsymbol{\epsilon}(\mathbf{v})\,\mathrm{d}V \\ k(\mathbf{v}; \mathbf{u}_n) &= \int_\Omega \boldsymbol{\sigma}\left(\mathbf{u}_n\right) : \boldsymbol{\epsilon}(\mathbf{v})\,\mathrm{d}V = \int_\Omega \left[\lambda\,\mathrm{tr}\,\boldsymbol{\epsilon}(\mathbf{u}_n)\mathbf{I} + 2\mu\boldsymbol{\epsilon}(\mathbf{u}_n)\right] : \boldsymbol{\epsilon}(\mathbf{v})\,\mathrm{d}V = \int_\Omega \boldsymbol{\epsilon}(\mathbf{u}_n) : \mathbb{E} : \boldsymbol{\epsilon}(\mathbf{v})\,\mathrm{d}V\end{aligned} \tag{3.51}$$

where $\mathbb{E}$ is the fourth-order elastic stiffness tensor.

3.5.2 Nonlinear case - inelastic material behaviors (under implementation)

An iterative Newton–Raphson procedure is used to solve the nonlinear problem. The unknown displacement $\mathbf{u}_{n+1}$ at global iteration $k+1$ is written as $\mathbf{u}_{n+1}^{k+1} = \mathbf{u}_{n+1}^{k} + \delta\mathbf{u}_{n+1}^{k+1}$, where $\mathbf{u}_{n+1}^{k}$ is the solution at iteration

k, and $\delta\mathbf{u}_{n+1}^{k+1}$ is the solution variation at iteration $k+1$. The latter is computed from the resolution of a linearized system of equations.

For this purpose, the variational formulation (3.39) is first written in residual form as:

$$\boxed{\begin{array}{l}\text{Find } \mathbf{u}_{n+1}^{k+1} \in \mathcal{U} \text{ such that :} \\ R(\mathbf{u}_{n+1}^{k+1}, \mathbf{v}) = \widetilde{\widetilde{K}}(\mathbf{u}_{n+1}^{k+1}, \mathbf{v}) - \widetilde{\widetilde{l}}(\mathbf{v}) = 0 \quad \forall\, \mathbf{v} \in \mathcal{V}\end{array}} \tag{3.52}$$

The residual(i.e., the out-of-balance force) is then linearized around solution $\mathbf{u}_{n+1}^{k}$ as follows:

$$R(\mathbf{u}_{n+1}^{k+1}, \mathbf{v}) = r(\mathbf{v}; \mathbf{u}_{n+1}^{k}) + R'(\delta\mathbf{u}_{n+1}^{k+1}, \mathbf{v}; \mathbf{u}_{n+1}^{k}) \tag{3.53}$$

where $r(\mathbf{v}; \mathbf{u}_{n+1}^{k}) = r(\mathbf{v}; \{\mathbf{b}, \mathbf{t}_n, \mathbf{t}_{n+1}, \mathbf{u}_n, \dot{\mathbf{u}}_n, \ddot{\mathbf{u}}_n\}, \mathbf{u}_{n+1}^{k})$ is the linear form corresponding to the out-of-balance forces at iteration k:

$$r(\mathbf{v}; \mathbf{u}_{n+1}^{k}) = \frac{1-\alpha_m}{\beta \Delta t^2} m(\mathbf{v}; \mathbf{u}_{n+1}^{k}) + \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t} c(\mathbf{v}; \mathbf{u}_{n+1}^{k}) + (1-\alpha_f)\, k(\mathbf{v}; \mathbf{u}_{n+1}^{k}) - \widetilde{\widetilde{l}}(\mathbf{v}) \tag{3.54}$$

and:

$$R'(\delta\mathbf{u}_{n+1}^{k+1}, \mathbf{v}; \mathbf{u}_{n+1}^{k}) = (1-\alpha_f)\left(1 + \frac{\gamma}{\beta\,\Delta t}\eta_K\right) K_t(\delta\mathbf{u}_{n+1}^{k+1}, \mathbf{v}; \mathbf{u}_{n+1}^{k}) \tag{3.55}$$

with:

$$K_t(\delta\mathbf{u}_{n+1}^{k+1}, \mathbf{v}; \mathbf{u}_{n+1}^{k}) = \int_\Omega \boldsymbol{\epsilon}(\delta\mathbf{u}_{n+1}^{k+1}) : \mathbb{D}^k : \boldsymbol{\epsilon}(\mathbf{v})\, \mathrm{d}V \tag{3.56}$$

The fourth order stiffness tensor $\mathbb{D}^k = D_{ijkl}(\mathbf{e}_i \otimes \mathbf{e}_j \otimes \mathbf{e}_k \otimes \mathbf{e}_l)$ can be defined differently according to the chosen algorithm. For instance, if a standard Newton–Raphson formulation is chosen, $\mathbb{D}^k = \partial_{\boldsymbol{\epsilon}}\boldsymbol{\sigma}(\mathbf{u}_{n+1}^{k})$ is the tangent stiffness operator at iteration k.

Finally, the discretized variational formulation to solve becomes:

$$\boxed{\begin{array}{l}\text{Find } \delta\mathbf{u}_{n+1}^{k+1} \in \mathcal{U}_\delta \text{ such that :} \\ (1-\alpha_f)\left(\dfrac{\gamma}{\beta\,\Delta t}\eta_K + 1\right) K_d(\delta\mathbf{u}_{n+1}^{k+1}, \mathbf{v}; \mathbf{u}_{n+1}^{k}) = -r(\mathbf{v}; \mathbf{u}_{n+1}^{k}) \quad \forall\, \mathbf{v} \in \mathcal{V}\end{array}} \tag{3.57}$$

where $\mathcal{U}_\delta$ is the admissibility space of the displacement variations, and $-r(\mathbf{v}; \mathbf{u}_{n+1}^{k})$ can now be interpreted as the difference between the external forces (represented by the linear form $\widetilde{\widetilde{l}}(\mathbf{v})$) and the internal forces (first three terms of equation (3.54)).

3.6 Paraxial formulation for absorbing layers

When spatially unbounded (infinite) domains are represented through bounded computational domains, spurious wave reflections can be observed boundaries. Several techniques have been proposed in the literature to introduce proper treatments of the boundary conditions allowing to reproduce propagation processes in infinite one-phase and two-phase media artificially. Among the available formulations, one can cite the ones based upon using lumped dumpers [10], Perfectly Matched Layers (PML) [2] and paraxial boundaries [5, 4, 1, 11].

3.6.1 Standard formulation

Paraxial approximation constitutes a local boundary condition which permits diffracting waves to be evacuated from the computational domain. To introduce the formulation, let us consider a split of the total

domain Ω^∞ into two subdomains Ω and Ω^E separated by a surface $\Sigma \in \mathbb{R}^d$ of outer normal $\mathbf{m}$ (pointing from Ω to Ω^E). On surface Σ, the continuity condition of the displacement field read:

$$[|\mathbf{u}|] = \mathbf{u} - \mathbf{u}^E = 0 \qquad \Sigma \times [0,T] \tag{3.58}$$

whereas the continuity of the traction vector reads:

$$[|\boldsymbol{\sigma}|] \cdot \mathbf{m} = \left(\boldsymbol{\sigma} - \boldsymbol{\sigma}^E\right) \cdot \mathbf{m} = \boldsymbol{\sigma} \cdot \mathbf{m} + \boldsymbol{\sigma}^E \cdot (-\mathbf{m}) = \mathbf{t} + \mathbf{t}^E = 0 \qquad \Sigma \times [0,T] \tag{3.59}$$

In previous equations, symbol $[|\bullet|]$ is used to denote the jump of function $\bullet$ across surface Σ, $\mathbf{u}^E = \mathbf{u}^E(\mathbf{x},t) : \Omega^E \times [0,T] \to \mathbb{R}^d$ is the vector-valued displacement field on Ω^E, and $\boldsymbol{\sigma}^E = \boldsymbol{\sigma}^E(\mathbf{u}^E) : \Omega^E \times [0,T] \to \mathbb{R}^{d\times d}$ is the corresponding stress tensor.

Variational formulation

Given the traction continuity condition (3.59), the variational problem to solve on Ω reads:

$$\boxed{\begin{array}{l}\text{Find } \mathbf{u} \in \mathcal{U} \text{ such that:} \\ M(\ddot{\mathbf{u}}, \mathbf{v}) + C(\dot{\mathbf{u}}, \mathbf{v}) + K(\mathbf{u}, \mathbf{v}) = b(\mathbf{v}; \mathbf{b})_\Omega + b(\mathbf{v}; \mathbf{t})_{\partial_t\Omega} - b(\mathbf{v}; \mathbf{t}^E)_\Sigma \quad \forall\, \mathbf{v} \in \mathcal{V}\end{array}} \tag{3.60}$$

Using a zeroth-order paraxial approximation, the traction vector $\mathbf{t}^E$ can be written as:

$$\mathbf{t}^E = A_0(\dot{\mathbf{u}}) = \rho c_p \dot{u}_m \mathbf{m} + \rho c_s \dot{\mathbf{u}}_s \tag{3.61}$$

where (c_p, c_s) are the the propagation velocities of compressional and shear waves, $\dot{u}_m = \dot{\mathbf{u}} \cdot \mathbf{m}$ is the velocity normal to Σ and $\dot{\mathbf{u}}_s = \dot{\mathbf{u}} - \dot{u}_m \mathbf{m}$ its tangent counterpart.

More in general, given a vector-valued field $\mathbf{a}$ (e.g., the displacement, velocity and acceleration fields), we write:[8]

$$A_0(\mathbf{a}) = \rho c_p \left(\mathbf{m} \otimes \mathbf{m}\right) \cdot \mathbf{a} + \rho c_s (\mathbf{I} - \mathbf{m} \otimes \mathbf{m}) \cdot \mathbf{a} = \rho \left[(c_p - c_s)\, m_i m_j + c_s \delta_{ij}\right] a_j \mathbf{e}_i \tag{3.62}$$

where symbol $\otimes$ denotes the dyadic product between first order tensors (vectors).[9]

Time-discretization

After time discretization, equation (3.61) becomes:

$$\begin{aligned} b(\mathbf{v}; \mathbf{t}^E)_\Sigma = {} & \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t} b(\mathbf{v}; A_0(\mathbf{u}_{n+1}))_\Sigma - \left[\frac{\gamma(1-\alpha_f)}{\beta} - 1\right] b(\mathbf{v}; A_0(\dot{\mathbf{u}}_n))_\Sigma \\ & \cdots - \Delta t(1-\alpha_f)\left(\frac{\gamma}{2\beta} - 1\right) b(\mathbf{v}; A_0(\ddot{\mathbf{u}}_n))_\Sigma - \frac{\gamma(1-\alpha_f)}{\beta\,\Delta t} b(\mathbf{v}; A_0(\mathbf{u}_n))_\Sigma \end{aligned} \tag{3.66}$$

The variational problem to solve thus reads:

$$\boxed{\begin{array}{l}\text{Find } \mathbf{u}_{n+1} \text{ such that :} \\ \tilde{\tilde{K}}(\mathbf{u}_{n+1}, \mathbf{v})(\mathbf{u}_{n+1}, \mathbf{v}) = \tilde{\tilde{l}}(\mathbf{v}) \quad \forall\, \mathbf{v} \in \mathcal{V}\end{array}} \tag{3.67}$$

[8] We recall that given three Euclidean vectors $\mathbf{v} \in \mathbb{R}^d$, $\mathbf{w} \in \mathbb{R}^d$ and $\mathbf{z} \in \mathbb{R}^d$, the dyadic product $\mathbf{v} \otimes \mathbf{w} \in \mathbb{R}^{d\times d}$ is the second order tensor defined by: $(\mathbf{v} \otimes \mathbf{w}) \cdot \mathbf{z} = (\mathbf{w} \cdot \mathbf{z})\mathbf{v}$. In components: $(\mathbf{v} \otimes \mathbf{w})_{ij} = v_i\, w_j$.

[9] Denoting (m_x, m_y, m_z) the components of vector $\mathbf{m}$ in the reference system $R(O, \mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$, the components of $A_0(\mathbf{a})$ read:

$$(A_0(\mathbf{a}))_x = \rho\left[(c_p - c_s) m_x (m_x a_x + m_y a_y + m_z a_z) + c_s\, a_x\right] \tag{3.63}$$

$$(A_0(\mathbf{a}))_y = \rho\left[(c_p - c_s) m_y (m_x a_x + m_y a_y + m_z a_z) + c_s\, a_y\right] \tag{3.64}$$

$$(A_0(\mathbf{a}))_z = \rho\left[(c_p - c_s) m_z (m_x a_x + m_y a_y + m_z a_z) + c_s\, a_z\right] \tag{3.65}$$

where $\tilde{\tilde{\tilde{K}}}(\mathbf{u}_{n+1}, \mathbf{v})$ is:

$$\begin{aligned}\tilde{\tilde{\tilde{K}}}(\mathbf{u}_{n+1}, \mathbf{v}) &= \tilde{\tilde{K}}(\mathbf{u}_{n+1}, \mathbf{v}) + \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t} b(\mathbf{v}; A_0(\mathbf{u}_{n+1}))_\Sigma \\ &= \frac{1-\alpha_m}{\beta \Delta t^2} M(\mathbf{u}_{n+1}, \mathbf{v}) + \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t} C(\mathbf{u}_{n+1}, \mathbf{v}) + (1-\alpha_f)\, K(\mathbf{u}_{n+1}, \mathbf{v}) \\ &\quad \cdots + \frac{\gamma\,(1-\alpha_f)}{\beta\,\Delta t} b(\mathbf{v}; A_0(\mathbf{u}_{n+1}))_\Sigma\end{aligned} \tag{3.68}$$

and $\tilde{\tilde{\tilde{l}}}(\mathbf{v})$ is:

$$\begin{aligned}\tilde{\tilde{\tilde{l}}}(\mathbf{v}) &= \tilde{\tilde{l}}(\mathbf{v}) + \left[\frac{\gamma(1-\alpha_f)}{\beta} - 1\right] b(\mathbf{v}; A_0(\dot{\mathbf{u}}_n))_\Sigma \\ &\quad \cdots + \Delta t(1-\alpha_f)\left(\frac{\gamma}{2\beta} - 1\right) b(\mathbf{v}; A_0(\ddot{\mathbf{u}}_n))_\Sigma + \frac{\gamma(1-\alpha_f)}{\beta\,\Delta t} b(\mathbf{v}; A_0(\mathbf{u}_n))_\Sigma\end{aligned} \tag{3.69}$$

3.6.2 Accounting for incident waves

Let us now split the total displacement vector at Σ into its incident $\mathbf{u}_{in}$ and radiant $\mathbf{u}_r$ components:

$$\mathbf{u} = \mathbf{u}^E = \mathbf{u}_{in} + \mathbf{u}_r \tag{3.70}$$

and use the zeroth-order paraxial approximation for expressing the traction contribution to the traction vector $\mathbf{t}^E$ due to the radiant field. The traction continuity condition (3.59), together with the linearity hypotheses at the vicinity of Σ, enables us to write:

$$\mathbf{t} = -\mathbf{t}^E(\mathbf{u}^E) = -\mathbf{t}^E(\mathbf{u}_{in}) - \mathbf{t}^E(\mathbf{u}_r) = -\mathbf{t}^E(\mathbf{u}_{in}) - A_0(\dot{\mathbf{u}}_r) = -\mathbf{t}^E(\dot{\mathbf{u}}_{in}) - A_0(\dot{\mathbf{u}}) + A_0(\dot{\mathbf{u}}_{in}) \tag{3.71}$$

here $\dot{\mathbf{u}}_{in}$ is known, whereas $\dot{\mathbf{u}}$ is the unknown velocity field.

The variational equation to solve now reads:

$$\boxed{\begin{array}{l}\text{Find } \mathbf{u}\in\mathcal{U} \text{ such that :} \\ M(\ddot{\mathbf{u}}, \mathbf{v}) + C(\dot{\mathbf{u}}, \mathbf{v}) + K(\mathbf{u}, \mathbf{v}) = b(\mathbf{v}; \mathbf{b})_\Omega + b(\mathbf{v}; \mathbf{t})_{\partial_t\Omega} - b(\mathbf{v}; A_0(\dot{\mathbf{u}}))_\Sigma \\ \quad \cdots - b(\mathbf{v}; \mathbf{t}_E(\dot{\mathbf{u}}_{in}))_\Sigma + b(\mathbf{v}; A_0(\dot{\mathbf{u}}_{in}))_\Sigma \quad \forall\, \mathbf{v} \in \mathcal{V}\end{array}} \tag{3.72}$$

where the last two terms are the only novelty with respect to equation (3.60).

Chapter 4

Tutorials

Preliminaries
Before diving into the tutorials, here are some preliminaries that will help you guide easily through them.

- A PSD simulation is performed in three steps: preprocessing, solving, and postprocessing.
- Domain: denoted by Ω is a a n-dimensional solid body such that $\Omega \subset \mathbb{R}^n$ with $n = 2$ for 2D problems or $n = 3$ for 3D problems.
- Finite element mesh: denoted by Ω^h with mesh size h. Mesh can be triangular in 2D and tetrahedral in 3D.
- MPI processes for simulation: denoted by N_p these are the total MPI ranks that will work in parallel to solve the problem.
- Partitioned mesh: denoted by $\{\Omega_i^h\}_{i=1}^{N_\mathrm{p}}$ these are set of subdomains which are held by each MPI rank during a parallel simulation.

4.1 Linear Elasticity

This is the basic model for solving simple solid mechanics. PSD allows for solving Linear Elasticity both in sequential and in parallel. We shall discuss how to do so in details within this section.

PSD is a FEM based solver, to solve a given physics it heavily relies on the variational formulations of the underlying physics. Let us begin with writing the variational formulation of system of Elasticity in which the primary unknown is the displacements vector $\mathbf{u} = \{u_j\}_{j=1}^n$. In the Lagrangian FE framework for searching the unknown nodal displacements vector $\mathbf{u}^h = \{u_j^h\}_{j=1}^n$ the variational formulation of system of Elasticity reads,

$$\forall i \in [\![1; N_\mathrm{p}]\!], \int_{\Omega_i^h} \boldsymbol{\sigma}(\boldsymbol{u}^h) : \boldsymbol{\varepsilon}(\boldsymbol{v}^h) = \int_{\partial\Omega_{i,\mathrm{N}}^h} \mathbf{f} \cdot \boldsymbol{v}^h \quad \forall\, \boldsymbol{v}^h \in \mathbb{V}^h, \boldsymbol{u}^h \in \mathbb{V}^h, \tag{4.1}$$

here, $\boldsymbol{u}^h$ is in fact the FE trial function and $\boldsymbol{v}^h = \{v_j^h\}_{j=1}^n$ is the FE test function.

$$\forall i \in [\![1; N_\mathrm{p}]\!], \int_{\Omega_i^h} \lambda \nabla \cdot \boldsymbol{u}^h \nabla \cdot \boldsymbol{v}^h + \int_{\Omega_i^h} 2\mu\boldsymbol{\varepsilon}(\boldsymbol{u}^h) : \boldsymbol{\varepsilon}(\boldsymbol{v}^h) - \int_{\Omega_i^h} \mathbf{f} \cdot \boldsymbol{v}^h = 0, \quad \forall \boldsymbol{v}^h \in [H_0^1(\Omega_i^h)]^n \tag{4.2}$$

In these formulations λ and μ are the Lame's parameters, $\mathbf{f}$ is the body force vector.

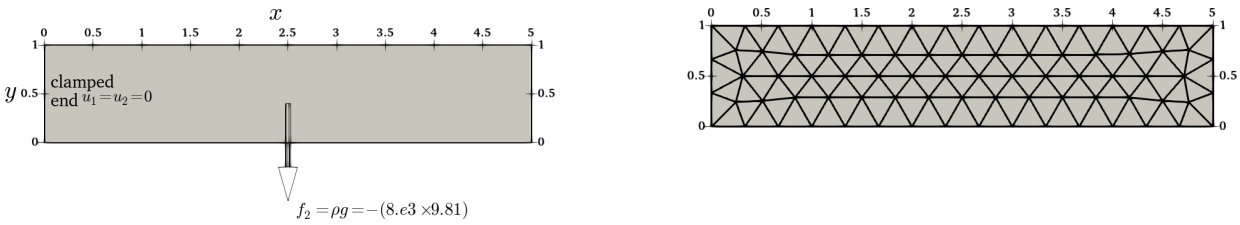


Figure 4.1: 2D bar clamped at left end and bending under own load. Geometry (left) and mesh (right).

4.1.1 PSD simulation of 2D bar problem bending under own body weight

To showcase the usage of Linear elasticity, we shall discuss here an example of a 2D bar, which bends under its own load. The bar —5×1 m^2 in area– is made up of material with $\rho = 8E3$, $E = 200E9$, and $\nu = 0.3$.

Step 1: Preprocessing

First step in a PSD simulation is "PSD setup", at this step you tell PSD what kind of physics, boundary conditions, approximations, mesh, etc are you expecting to solve. For "PSD setup" go to the `PSD/Solver` folder, launch `PSD_PreProcess` from the terminal, to do so run the following command.

```
PSD_PreProcess -dimension 2 -bodyforce  -dirichletconditions 1 -plot
```

After the "PSD setup" runs successfully you should see many `.edp` files in your `PSD/Solver` folder. *What do the arguments mean ?* `-dimension 2` means it is a 2D simulation, `-bodyforce` with body force; `-dirichletconditions 1` says we have one Dirichlet border; and `-plot` means we would like to have ParaView post processing files. The input properties "$E, \nu, \mathbf{f}$" can be changed in `ControlParameters.edp` by changing `E, nu, f2`. These are changed to `E = 200.e9`, `nu = 0.3;`, `f2=8.e3*(-9.81);`. Note, `f2` is the second component of **f**. Dirichlet boundary conditions are also provided in `ControlParameters.edp`. To provide Dirichlet label (clamped end), the vector `int[int] Dlabel = [2];` should be used, here we assume that the label number of the left end is 2. For this clamped end $u_1 = u_2 = 0$, this is provided by `real[int] Dvalue = [0.,0.];`.

Step 2: Solving

As PSD is a parallel solver, let us use 4 cores to solve the 2D bar case. To do so enter the following command:

```
ff-mpirun -np 4 Main.edp
```

Here `-np 4` denote the argument used to enter the number of cores. Note that if your problem is large use more cores. PSD has been tested upto 13,000 cores, surely you will now need that many for the 2D bar problem.

Step 3: Postprocessing

PSD allows postprocessing of results in ParaView. After the step 2 mentioned above finishes. Launch ParaView and have a look at the `.pvd` file in the `PSD/Solver/VTUs_DATE_TIME` folder.

Figure 4.2: 2D clamped bar results. Partitioned mesh (left) and 1000X warped displacement field (right).

4.1.2 PSD simulation of 2D bar problem clamped at both ends

For this test the properties of the material are the same as used in section 4.1.1.

Step 1: Preprocessing

For "PSD setup" go to the `PSD/Solver` folder, launch the terminal there and run the following command.

```
PSD_PreProcess -dimension 2 -plot -dirichletconditions 2 -bodyforce
```

Since basic nature of both the problems is same this is the exact same command used in preprocessing of section 4.1.1. The only difference of this problem compared to the one from section 4.1.1 is that an additional Dirichlet condition needs to be supplied, notified to PSD by `-dirichletconditions 2`. To provide Dirichlet label of the two clamped end in `ControlParameters.edp` the vector `int[int] Dlabel = [2,4];` should be used, here we assume that the label number of the left end is 2 and right end is 4. For these clamped end $u_1 = u_2 = 0$, this is provided by `real[int] Dvalue = [0.,0.,0.,0.];`. In the `real[int] Dvalue = [0.,0.,0.,0.];` the fist two digits correspond to u_1, u_2 of label 2 and the last two digits correspond to u_1, u_2 of label 4.

Step 2: Solving

Let us now use 3 cores to solve this problem. To do so enter the following command:

```
ff-mpirun -np 3 Main.edp
```

Notice, that this is the exact same command used in solving of section 4.1.1, with only difference that we now use `-np 3` vs. `-np 4`.

Step 3: Postprocessing

Launch ParaView and have a look at the `.pvd` file in the `PSD/Solver/VTUs_DATE_TIME` folder.

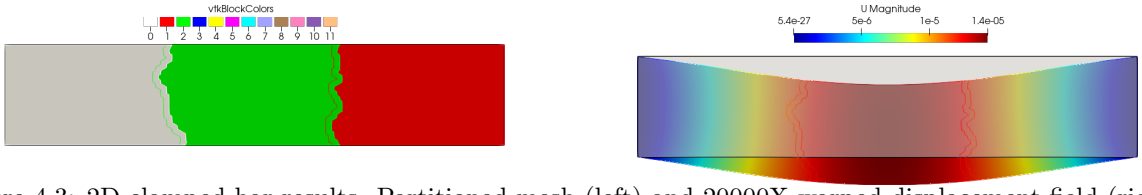


Figure 4.3: 2D clamped bar results. Partitioned mesh (left) and 20000X warped displacement field (right).

In fig. 4.3 there are only three subdomais in the partitioned mesh since only three cores were used.

Redoing the test on Jupiter and moon

Imagine, you wish to know how the test would compare if performed on Moon and Jupiter. The only thing that will change now is the gravitational pull, for Moon $g = 1.32$ and for Jupiter $g = 24.79$. To perform the moon test simply change `f2=8.e3*(-1.32);` in `ControlParameters.edp` and redo step 2 and step 3. Similarly for the Jupiter test `f2=8.e3*(-24.79);` in `ControlParameters.edp` and redo step 2 and step 3.

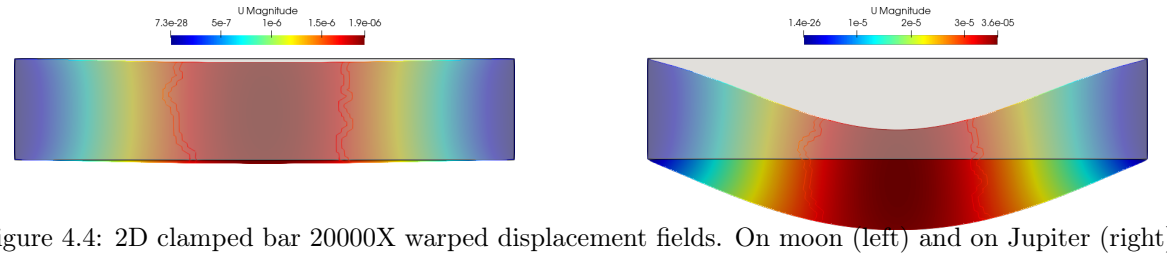


Figure 4.4: 2D clamped bar 20000X warped displacement fields. On moon (left) and on Jupiter (right).

4.1.3 PSD simulation of 2D bar problem clamped at one end wile being pulled at the other end (Dirichlet-Dirichlet case)

In this section we showcase the 2D bar problem simulation with one end clamped wile being pulled at the other end. Body force is neglected and the non clamped ends pull is approximated with Dirichlet displacement

$u_1 = 1$. If this simulation is compared to the previous one from section 4.1.1, the only difference now is that no body force is applied and an additional Dirichlet condition is applied at the free end of the bar. Here is how PSD simulation of this case can be performed.

Step 1: Preprocessing

For "PSD setup" go to the `PSD/Solver` folder, launch the terminal there and run the following command.

```
PSD_PreProcess -dimension 2 -dirichletconditions 2 -plot
```

In comparison to preprocessing from sections 4.1.1 and 4.1.2, notice that `-bodyforce` is missing. This is due to the fact that for this problem we assume zero body force. Just like in section 4.1.2 `-dirichletconditions 2`, which notifies to PSD that there are two Dirichlet borders —the clamped and the pulled ends of the bar— in this simulation. To add the values and label numbers of the Dirichlet borders edit the `ControlParameters.edp`, the vector `int[int] Dlabel = [2,4];` should be used, here we assume that the label number of the left end is 2 and right end is 4. For the left end $u_1 = 0, u_2 = 0$ and for the right end $u_1 = 1, u_2 = 0$, this is provided by `real[int] Dvalue = [0.,0.,1.,0.];`. In the `real[int] Dvalue = [0.,0.,0.,0.];` the fist two digits correspond to u_1, u_2 of label 2 and the last two digits correspond to u_1, u_2 of label 4. Note that here at border 4 we have explicitly set $u_2 = 0$ this means the bar is not allowed to shrink in y direction (compress), however you might wish to allow the bar to compress. Such a simulation requires a little bit of more editing in preprocessing step, we shall deal with this in the later tutorials, for now we focus on the current case. So now PSD should be ready to solve.

Step 2: Solving

Let us now use 2 cores to solve this problem. To do so enter the following command:

```
ff-mpirun -np 2 Main.edp
```

Notice, that this is the exact same command used in solving of sections 4.1.1 and 4.1.2, with only difference that we now use `-np 2` vs. `-np 4` in section 4.1.1 and `-np 3` in section 4.1.2.

Step 3: Postprocessing

Launch ParaView and have a look at the `.pvd` file in the `PSD/Solver/VTUs_DATE_TIME` folder.

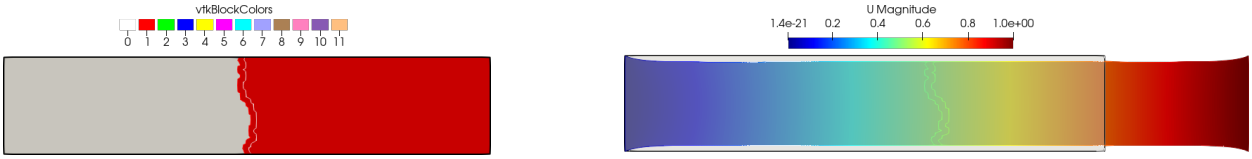


Figure 4.5: 2D bar results. Partitioned mesh (left) and 1.5X warped displacement field (right).

Note now in fig. 4.5 there are only two subdomais in the partitioned mesh since only three cores were used. As expected we see that the right end of the bar which is being pulled does not contract in y direction.

4.1.4 PSD simulation of 2D bar problem clamped at one end wile being pulled at the other end (Dirichlet–Neumann case)

Similar simulation, as in section 4.1.3 is presented in this section. We showcase the 2D bar problem simulation with one end clamped wile being pulled at the other end. Just like simulation from section 4.1.3 the body force is neglected, However now the non clamped ends pull is approximated with Neumann force $\int_{\partial\Omega_N^h}(\mathbf{t}\cdot\boldsymbol{v}^h)$. To simulate the pull we assume traction vector $\mathbf{t} = [t_1, t_2] = [10^9., 0]$ acting on the non clamped right end of the bar, i.e., force in x direction is 10 units. Here is how PSD simulation of this case can be performed.

Step 1: Preprocessing

For "PSD setup" go to the `PSD/Solver` folder, launch the terminal there and run the following command.

```
PSD_PreProcess -dimension 2 -dirichletconditions 1  -tractionconditions 1 -plot
```

the comandline flag `-dirichletconditions 1`, notifies to PSD that there is one Dirichlet border —the clamped end of the bar— in this simulation. And the flag `-tractionconditions 1` notifies to PSD that there is one traction border —the right end of the bar— in this simulation. To add the values and label numbers of the Dirichlet borders edit the `ControlParameters.edp`, the vector `int[int] Dlabel = [2];` should be used, here we assume that the label number of the left end is 2. For the left end $u_1 = 0, u_2 = 0$ hence in `ControlParameters.edp`, use `real[int] Dvalue = [0.,0.];`. To add the values and label numbers of the traction borders edit the `ControlParameters.edp`, the vector `int[int] Tlabel = [4];` should be used, here we assume that the label number of the right end is 4. For this end $\mathbf{t} = [t_1, t_2] = [10^9., 0]$, hence in `ControlParameters.edp`, use `real tx0=1.e9, ty0=0.;`.

Step 2: Solving

Let us now use 5 cores to solve this problem. To do so enter the following command:

```
ff-mpirun -np 5 Main.edp
```

Notice, that this is the exact same command used in solving the previous bar problems from other sections, with only difference that we now use `-np 5`.

Step 3: Postprocessing

Launch ParaView and have a look at the `.pvd` file in the `PSD/Solver/VTUs_DATE_TIME` folder.

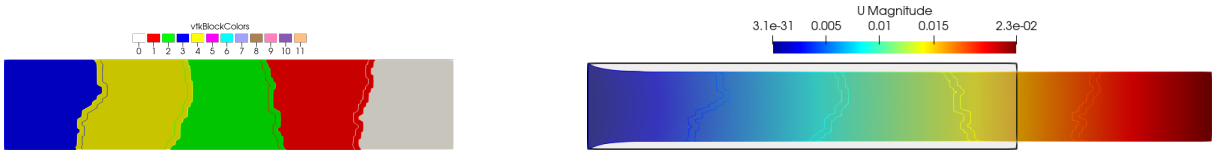


Figure 4.6: 2D bar results. Partitioned mesh (left) and 100X warped displacement field (right).

Note now in fig. 4.6 there are five subdomais in the partitioned mesh since five cores were used. Contrary to fig. 4.5, as expected, we see that the right end of the bar which is being pulled now contract in y direction. This is due to the fact that there is no Dirichlet condition at this end now.

4.1.5 PSD simulation of 2D bar problem clamped at one end wile being pulled at the other end (Dirichlet-Neumann-Point boundary conditions case)

Similar simulations, as in sections 4.1.3 and 4.1.4 is presented in this section. We showcase the 2D bar problem simulation with one end clamped wile being pulled at the other end. Contrary to simulation in sections 4.1.3 and 4.1.4, the clamped end just restricts x movement, i.e, $u_1 = 0$. Just like simulation from sections 4.1.3 and 4.1.4 the body force is neglected. Just like simulation in section 4.1.4, the non clamped ends pull is approximated with Neumann force $\int_{\partial\Omega_N^h}(\mathbf{t}\cdot\boldsymbol{v}^h)$. To simulate the pull we assume traction vector $\mathbf{t} = [t_1, t_2] = [10^9, 0]$ acting on the non clamped right end of the bar, i.e., force in x direction is 10^9 units. Here is how PSD simulation of this case can be performed.

Step 1: Preprocessing

For "PSD setup" go to the `PSD/Solver` folder, launch the terminal there and run the following command.

```
PSD_PreProcess -dimension 2 -dirichletconditions 1 -tractionconditions 1 -dirichletpointconditions 1 -plot
```

Additional flag `-dirichletpointconditions 1` now appears, this notifies to PSD that there is one Dirichlet point boundary condition. To add the values and label numbers of the Dirichlet borders which contains this point edit the `ControlParameters.edp`, the vector `int[int] Dpointlab = [2];` should be used, here we assume that the label number of the left end is 2 and that left end contains our Dirichlet point. For this point $x = y = 0$ and $u_1 = 0, u_2 = 0$ to specify this in `ControlParameters.edp`, use `PnV = [ 0., 0., 0., 0.];` which is infact a vector of $[x, y, u_1, u_2]$. Via the flags we specified that `-dirichletconditions 1`, i.e., there is one Dirichlet border. However now in this simulation the border only has one Dirichlet condition $u_1 = 0$, while u_2 is free. This condition needs editing of `VariationalFormulations.edp`, comment or remove the line `u1 = Dvalue[1]`. Commenting or removing this line lets free `u1` which is infact u_2. Note that this is due to the fact that in PSD $[u_1, u_2, u_3]$ maps to `[u,u1,u2]`.

Step 2: Solving

Let us now use 6 cores to solve this problem. To do so enter the following command:

```
ff-mpirun -np 5 Main.edp
```

Notice, that this is the exact same command used in solving the previous bar problems from other sections, with only difference that we now use `-np 6`.

Step 3: Postprocessing

Launch ParaView and have a look at the `.pvd` file in the `PSD/Solver/VTUs_DATE_TIME` folder.

Figure 4.7: 2D bar results. Partitioned mesh (left) and 100X warped displacement field (right).

Note now in fig. 4.7 there are six subdomais in the partitioned mesh. As expected, we see that the right and the left end of the bar which is being pulled now contract in y direction, and the bar elongates in x direction.

4.1.6 PSD simulation of 3D bar problem clamped at one end wile being pulled at the other end (Dirichlet-Neumann case)

In this section we present a 3D PSD simulation of a clamped bar which his being loaded in vertical direction at the non-clamped end. This simulation is like the one presented in section 4.1.4, however in 3D. The material properties are same as before, and at the non-clamped end traction $t_y = -10^9$ units. The 3D bar is $1 \times 1 \times 5$ m^3.

Here is how PSD simulation of this case can be performed.

Step 1: Preprocessing

For "PSD setup" go to the `PSD/Solver` folder, launch the terminal there and run the following command.

```
PSD_PreProcess -dimension 3 -dirichletconditions 1  -tractionconditions 1 -plot
```

the comandline flag `-dirichletconditions 1` notifies to PSD that there is one Dirichlet border —the clamped end of the bar— in this simulation; `-dimension 3` means the simulation is 3D. And the flag `-tractionconditions 1` notifies to PSD that there is one traction border —the right end of the bar— in this simulation. To add the values and label numbers of the Dirichlet borders edit the `ControlParameters.edp`, the vector `int[int] Dlabel = [1];` should be used, here we know that the label number of the left clamped end is 1. For the left end $u_1 = 0, u_2 = 0, u_3 = 0$ hence in `ControlParameters.edp`, use `real[int] Dvalue = [0.,0.,0.];`. To add the values and label numbers of the traction borders edit the `ControlParameters.edp`, the vector `int[int] Tlabel = [2];` should be used, here we know that the label number of the right end is 2. For this end $\mathbf{t} = [t_1, t_2, t_3] = [0., 10^9, 0.]$, hence in `ControlParameters.edp`, use `real tx0=0., ty0=1.e9, tz0=0.;`.

Step 2: Solving

Let us now use 4 cores to solve this problem. To do so enter the following command:

```
ff-mpirun -np 4 Main.edp
```

Notice, that this is the exact same command used in solving the previous bar problems from other sections.

Step 3: Postprocessing

Launch ParaView and have a look at the `.pvd` file in the `PSD/Solver/VTUs_DATE_TIME` folder.

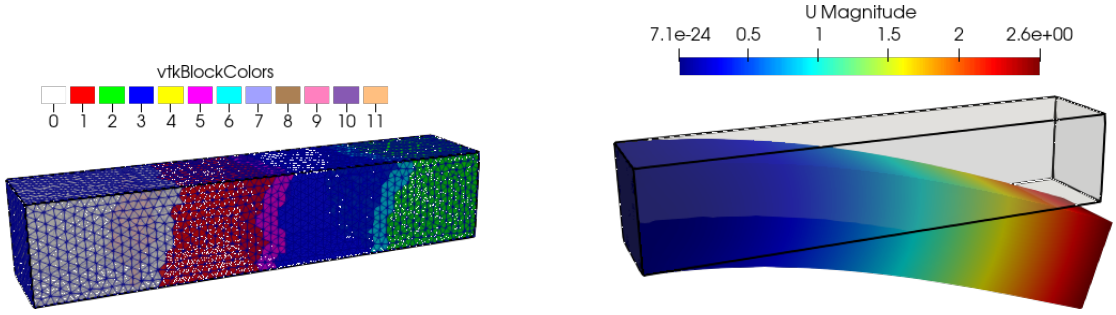


Figure 4.8: 3D bar results. Partitioned mesh (left) and 0.5X warped displacement field (right).

In fig. 4.8 there are four subdomais in the partitioned mesh since four cores were used.

4.1.7 PSD simulation of 3D mechanical piece (Dirichlet-Neumann case) with complex mesh

So far in the previous cases we only concentrated on bar simulations, which were more or less trivial cases. Moreover, the bar meshes are provided with the PSD solver. In this section we now turn towards 3D simulation of a mechanical piece, the geometry of which is shown in fig. 4.9. The left (small) hole is fixed: $u_1 = u_2 = u_3 = 0$, while as traction force $t_x = 10^9$ is applied on the large hole.

You can grab a copy of CAD geometry for the mechanical piece (the Gmsh `.geo`) your local Gmsh installation folder `gmsh/share/doc/gmsh/demos/simple_geo/piece.geo`. The listing of the file is also given in @. To generate the mesh `piece.msh` simply do

```
gmsh -3 piece.geo
```

Place the generated mesh `piece.msh` in `/PSD/Meshes/3D/piece.msh`. Now the PSD simulation can be performed.

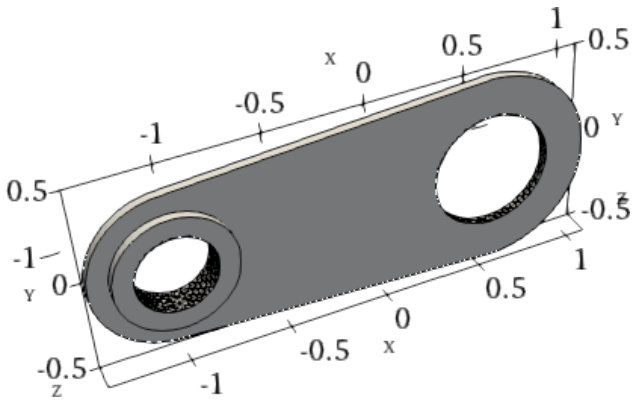


Figure 4.9: 3D mechanical piece.

Step 1: Preprocessing

For "PSD setup" go to the `PSD/Solver` folder, launch the terminal there and run the following command.

```
PSD_PreProcess -dimension 3 -dirichletconditions 1  -tractionconditions 1 -plot
```

Here, by using these parameters we have generated one Dirichlet condition and one traction condition, respectively to be applied to the small and the large holes in the mesh. Further, by using `-dimension 3` we have let PSD know that the problem is 3D .In the `/PSD/Meshes/3D/piece.msh` generated, the label 4 (resp. 3) corresponds to the Dirichlet (resp. traction) border. To add the values and label numbers of the Dirichlet borders edit the `ControlParameters.edp`, the vector `int[int] Dlabel = [4];` should be used. For this label $u_1 = 0, u_2 = 0, u_3 = 0$ hence in `ControlParameters.edp`, use `real[int] Dvalue = [0.,0.,0.];`. To add the values and label numbers of the traction borders edit the `ControlParameters.edp`, the vector `int[int] Tlabel = [3];` should be used. For this end $\mathbf{t} = [t_1, t_2, t_3] = [0., 10^9, 0.]$, hence in `ControlParameters.edp`, use `real tx0=0., ty0=1.e9, tz0=0.,;`. Finally we use steel properties for the material, so in `ControlParameters.edp` the parameters `real E = 200.e9;` and `real nu = 0.3;` should be used. These represent E and ν, respectively. With all the properties and boundary conditions set we now use `string ThName = "../Meshes/3D/piece";` in the `ControlParameters.edp` file, this notifies PSD about the name of the mesh used for this simulation.

Step 2: Solving

Let us now use 2 cores to solve this problem. To do so enter the following command:

```
ff-mpirun -np 2 Main.edp
```

Step 3: Postprocessing

Launch ParaView and have a look at the `.pvd` file in the `PSD/Solver/VTUs_DATE_TIME` folder.

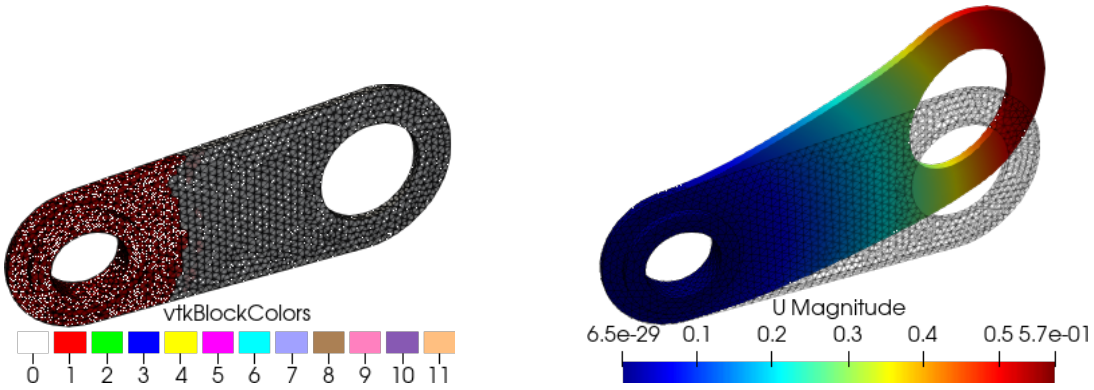
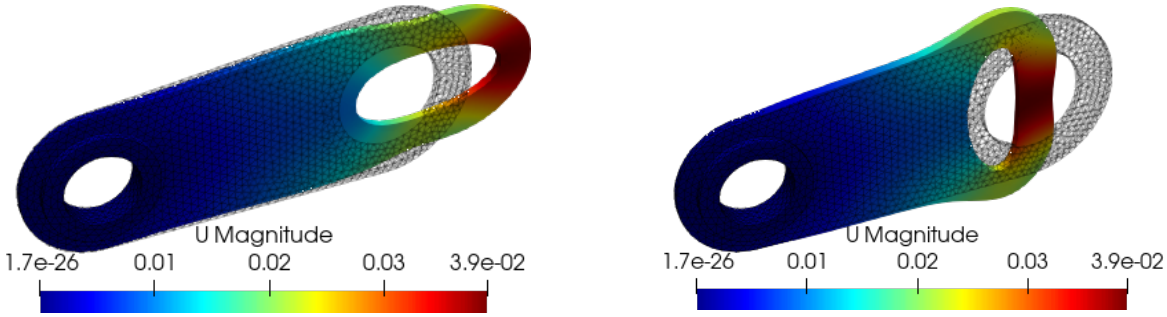


Figure 4.10: Mechanical piece test results. Partitioned mesh (left) and warped displacement field (right).

Redoing the test with different conditions



(a) `real tx0=1.e9, ty0=0, tz0=0.,;`

(b) `real tx0=1.e9, ty0=0, tz0=0.,;`

Figure 4.11: Mechanical piece test results.

4.2 Damage mechanics

4.2.1 Hybrid phase-field for damage

On a meshed domain $\Omega^h \in \Omega \subset \mathbb{R}^n$, for damage mechanics the mixed finite element variational formulation in the Lagrangian framework for searching the unknown nodal displacements vector $\mathbf{u}^h = [u_1, u_2, u_3]^\mathsf{T}$ reads,

$$\begin{aligned} &\text{search } \boldsymbol{u}^h \in \mathbb{V}^h \text{ that satisfies } \forall\, t \in [0,T]: \\ &\int_{\Omega^h} \left[(1-d^h)^2 + \kappa\right] \boldsymbol{\sigma}(\boldsymbol{u}^h) : \boldsymbol{\varepsilon}(\boldsymbol{v}^h)\, \mathrm{dv} = \int_{\partial\Omega^h_\mathrm{N}} \overline{\mathbf{t}} \cdot \boldsymbol{v}^h\, \mathrm{ds} \quad \forall\, \mathbf{v}^h \in \mathbb{V}^h, \end{aligned} \tag{4.3}$$

where $\kappa \ll 1$ is a model parameter to prevent numerical singularity when $d \to 1$. In this formulation, the notation ":" is used for the double contraction between tensors (i.e., component-wise tensor product) and $\mathbb{V}^h$ is a mixed third order vector valued finite element functional space to approximate vector test function $\boldsymbol{v}^h$ and vector trial function $\boldsymbol{u}^h$:

$$\mathbb{V}^h = \left\{ \mathbf{u}^h \in [H^1(\Omega^h)]^3 \;\; \forall t \in [0,T] \mid \forall \mathbf{x} \in \partial\Omega^h_\mathrm{D} \;\; \boldsymbol{u}^h = \overline{\mathbf{u}} \right\}, \tag{4.4}$$

with $H^1(\Omega^h)$ denoting a square integrable Sobolev functional space. Similarly, for the phase-field the standard finite element variational formulation for the unknown damage scalar d^h reads,

$$\begin{aligned} &\text{search } d^h \in V^h \text{ that satisfies } \forall\, t \in [0,T]: \\ &\int_{\Omega^h} \left[\frac{G_\mathrm{c}}{l_0} + 2\mathcal{H}^+(\boldsymbol{u}^h) \right] d^h \theta^h\, \mathrm{dv} + \int_{\Omega^h} G_\mathrm{c} l_0 \nabla d^h \cdot \nabla \theta^h\, \mathrm{dv} = \int_{\Omega^h} 2\mathcal{H}^+(\boldsymbol{u}^h)\theta^h\, \mathrm{dv} \quad \forall\, \theta^h \in V^h, \end{aligned} \tag{4.5}$$

where, V^h denotes the scalar finite element functional space to approximate scalar test function θ^h and scalar trial function d^h:

$$V^h = \left\{ d^h \in H^1(\Omega^h) \;\; \forall t \in [0,T] \;\middle|\; d^h \in [0,1] \right\}. \tag{4.6}$$

4.3 Elastodynamics

4.4 Soil dynamics

```
import math
import numpy as np
from lib.analytical import csa

sin2_theta  = np.sin(theta)**2  // THis is  a commen
+= -= *= /= + - * / ? < > & % == <=
# += -= *= /= + - * / ? < > & % == <=
def test(a=100, b=True):
    <= >= == 2 + 3j * 7e-3
```

Chapter 5

Validation

Chapter 6

Functions and descriptions

6.1 Flag descriptions

`INT TYPE`	
`-dirichletpointconditions`	Number of Dirchlet points. Default `1.`
`-dirichletconditions`	Number of Dirchlet boundaries. Default `1.`
`-tractionconditions`	Number of Neumann/traction boundaries. Default `1.`
`-parmetis_worker`	Active when mesh partitioner is parmetis.
`-dimension`	Dimension of problem. 2 for 2D 3 for 3D. Default 2.
`-lagrange`	Lagrange order used for building FE space. Default `1 for P1.`
`STRING TYPE`	
`-timediscretization`	Time discretization type. Use `generalized-alpha | newmark-beta | hht-alpha | central-difference.`
`-nonlinearmethod`	Nonlinear method type. Use `Picard | Newton-Raphsons.`
`-partitioner`	Mesh partitioner. Use `metis | scotch | parmetis.`
`-postprocess`	Indicate postprocessing quantity. Use `u | v | a | phi | uphi | uva.`
`-problem`	Interested problem. Use `linear-elasticity | damage | elastodynamics | soildynamics.`
`-model`	Interested model. Use `hybrid-phase-field | Mazar.`
`BOOL TYPE`	
`-help`	Helping message on the terminal.
`-plot`	Output solution to ParaView.
`-debug`	OpenGL plotting routine for displaying solution.
`-useGFP`	Activate use of GoFastPlugins. A suite of C++ plugins.
`-timelog`	To setup time logging for various phases of the solver.
`-useRCM`	Mesh level renumbering via Reverse Cuthill Mckee.
`-pipegnu`	Realtime pipe plotting using GnuPlot.
`-bodyforce`	Use volumetric source term (body force).
`-vectorial`	Generate vectorial space solver for non-linear.
`-energydecomp`	Hybrid phase-field energy decomposition.
`-fastmethod`	Produce a fast solver (more optimized).
`-energydecomp`	Hybrid phase-field energy decomposition.
`-sequential`	To generate a sequential PSD solver.

6.2 Functions in gofastplugins.cpp

6.2.1 GFPeigen

`GFPeigen(A,Eval,Evec);`	`A` is a matrix, `Eval` is vector returning eigenvalues, `Evec` is the matrix returning eigenvectors.

This is a call by reference pointer-based function of GFP library. It is used for computation of the eigenvalues and eigenvectors of a real symmetric matrix (upper triangular). This function inturn uses LAPACK libraries `dsyev_` for calculation of eigenvalues and eigenvectors.

The function `GFPeigen` which can be called from PSD is coded as `lapack_dsyevIn` function within the gofastplugins.cpp. this function argument 1: `A` is the supplied symmetric matrix, argument 2: `vp` are the output eigenvalues and argument 3: `vectp` are the output eigenvectors.

```
long lapack_dsyev (KNM<double> *const &A, KN<double> *const &vp, KNM<double> *
    const &vectp)
{
  intblas n = A->N();
  KNM<double> mat(*A);

    .
    .
    dsyev_(&JOBZ, &UPLO, &n, mat, &n, *vp, w, &lw, &info);
    .
    .
    *vectp = mat;
}
```

6.2.2 GFPeigenAlone

`GFPeigen(A,Eval,Evec);`	`A` is a matrix, `Eval` is vector returning eigenvalues.

This is a call by reference pointer based function of GFP library. It is used for computation of the eigenvalues of a real symmetric matrix (upper triangular). This function inturn uses LAPACK library for calculation of eigenvalues. The function `GFPeigenAlone` which can be called from PSD is coded as `lapack_dsyevAlone` function within the gofastplugins.cpp. In this function argument 1: `A` is the supplied symmetric matrix and argument 2: `vp` is the output eigenvalues of matrix `A` .

```
long lapack_dsyevAlone (KNM<double> *const &A, KN<double> *const &vp)
```

6.2.3 GFPmaxintwoFEfields

This is a call by reference pointer based function of GFP library. It is used to find out max between two real numbers `f` and `f1` (two 1D arrays). The max is stored in array `f`.

```
double GFPmaxintwoP1(KN<double> *const & f, KN<double> *const & f1)
```

Chapter 7

Gallery

This chapter showcases some test cases that have been performed with PSD.

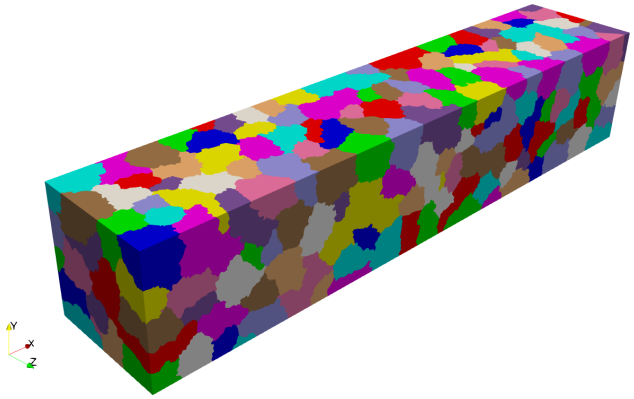


Figure 7.1: 90 M dof with 400 partitions.

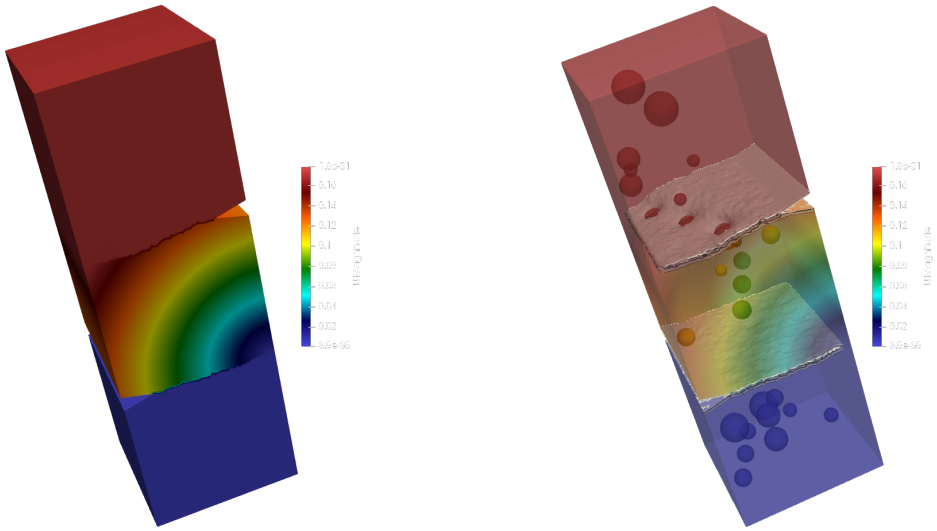


Figure 7.2: Perforated concrete bar cracking.

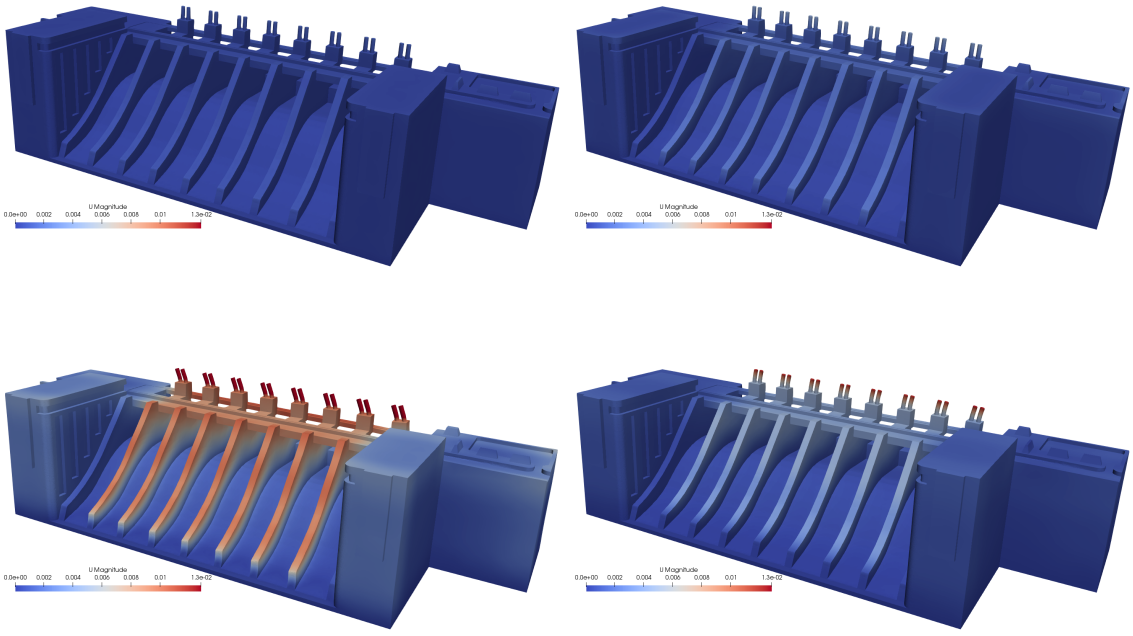


Figure 7.3: Full scale dam under seismic load.

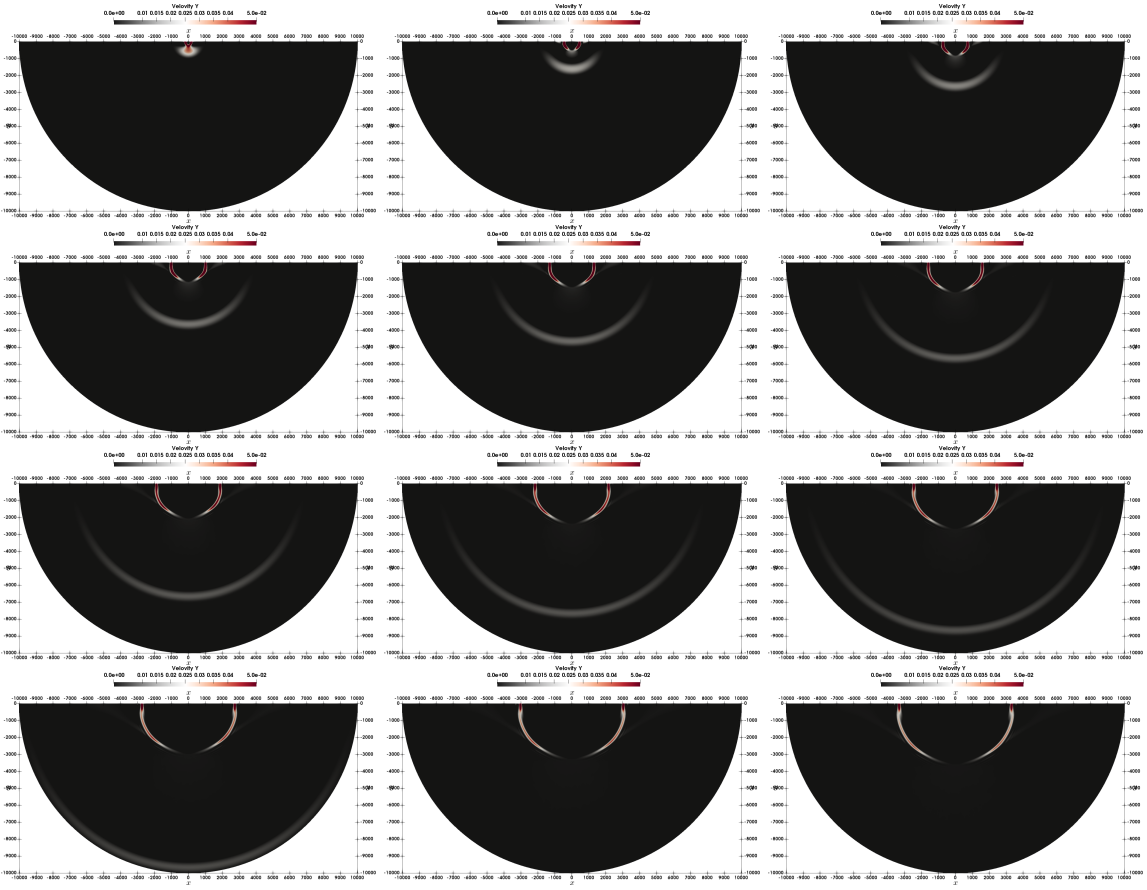


Figure 7.4: Seismic signal dispersion in soil.

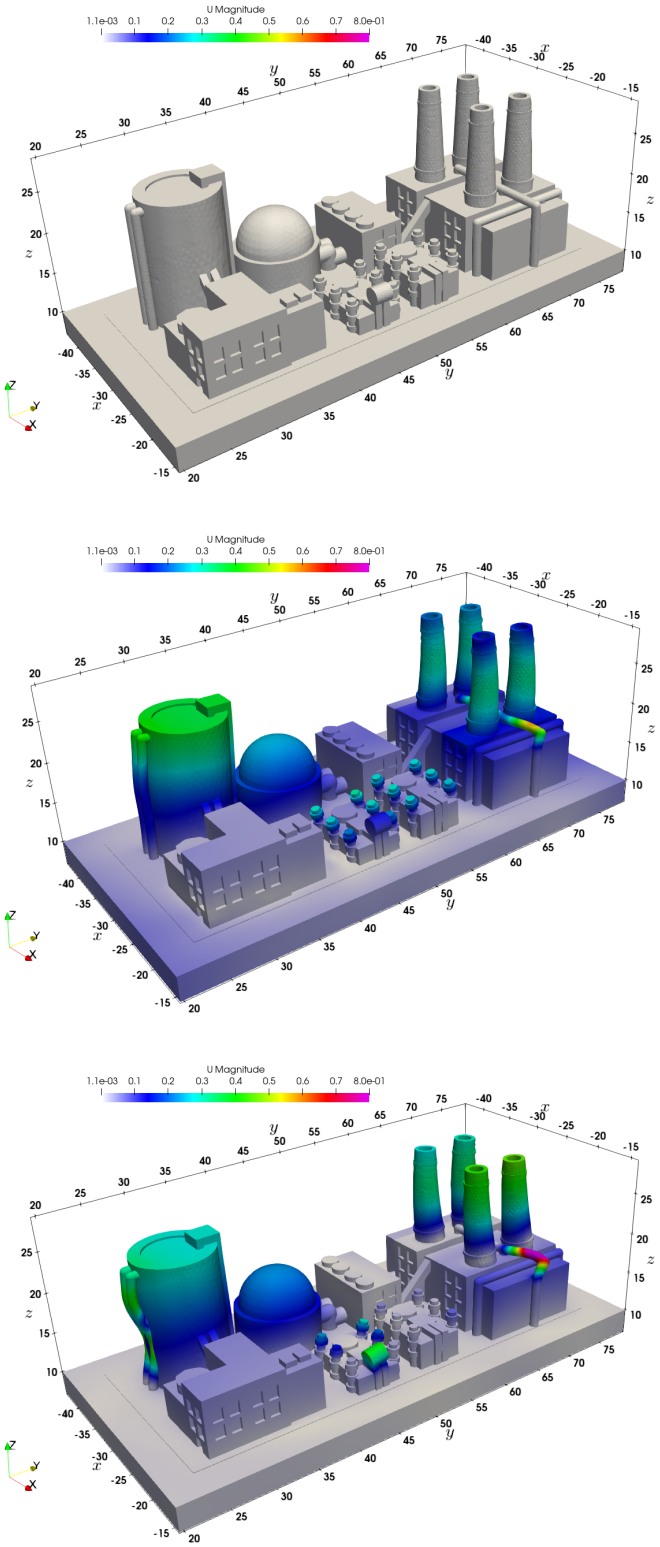


Figure 7.5: Seismic signal on nuclear site.

Bibliography

[1] D Aubry et al. "Local amplification of a seismic incident field through an elastoplastic sedimentary valley". In: *International conference on numerical methods in geomechanics*. 1985, pp. 1343–1350.

[2] Jean-Pierre Berenger. "A perfectly matched layer for the absorption of electromagnetic waves". In: *Journal of computational physics* 114.2 (1994), pp. 185–200.

[3] Jintai Chung and GM1223971 Hulbert. "A time integration algorithm for structural dynamics with improved numerical dissipation: the generalized-α method". In: (1993).

[4] Robert Clayton and Björn Engquist. "Absorbing boundary conditions for acoustic and elastic wave equations". In: *Bulletin of the seismological society of America* 67.6 (1977), pp. 1529–1540.

[5] Björn Engquist and Andrew Majda. "Absorbing boundary conditions for numerical simulation of waves". In: *Proceedings of the National Academy of Sciences* 74.5 (1977), pp. 1765–1766.

[6] Silvano Erlicher, Luca Bonaventura, and Oreste S Bursi. "The analysis of the generalized-α method for non-linear dynamic problems". In: *Computational Mechanics* 28.2 (2002), pp. 83–104.

[7] Hans M Hilber, Thomas JR Hughes, and Robert L Taylor. "Improved numerical dissipation for time integration algorithms in structural dynamics". In: *Earthquake Engineering & Structural Dynamics* 5.3 (1977), pp. 283–292.

[8] C Hoff and PJ Pahl. "Development of an implicit method with numerical dissipation from a generalized single-step algorithm for structural dynamics". In: *Computer Methods in Applied Mechanics and Engineering* 67.3 (1988), pp. 367–385.

[9] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

[10] John Lysmer and Roger L Kuhlemeyer. "Finite dynamic model for infinite media". In: *Journal of the Engineering Mechanics Division* 95.4 (1969), pp. 859–878.

[11] Hormoz Modaressi and Ikhlef Benzenati. "Paraxial approximation for poroelastic media". In: *Soil Dynamics and Earthquake Engineering* 13.2 (1994), pp. 117–129. ISSN: 0267-7261. DOI: https://doi.org/10.1016/0267-7261(94)90004-3. URL: http://www.sciencedirect.com/science/article/pii/0267726194900043.

[12] Nathan Mortimore Newmark et al. "A method of computation for structural dynamics". In: American Society of Civil Engineers. 1959.

[13] WL Wood, M Bossak, and OC Zienkiewicz. "An alpha modification of Newmark's method". In: *International Journal for Numerical Methods in Engineering* 15.10 (1980), pp. 1562–1566.

[14] Olgierd C Zienkiewicz and Robert L Taylor. *The finite element method, ; volume 1: basic formulation and linear problems*. 1994.