MuPhiSim: a platform for multiphysics simulation

Reference Manual for Users and Developers

December 15, 2022

Contents

CHAPTER 1

USER'S MANUAL

User's Reference Manual to understand the main lines of MuPhiSim

1.1 Introduction

Meshless methods have attracted the computational mechanics community as a consequence of their adaptability and versatility. These methods are based on a set of nodes without any, a priori, relation between them discretising the problem domain. This is contrary to the finite element method (FEM) in which nodes are related to each other to build elements. Meshless methods (MM) appear ideal for modelling problems in which the domain suffers large deformations, moving boundaries or material growth. However, these approaches are computationally more expensive than FEM, and the imposition of Dirichlet boundary conditions is more difficult, thus highlighting the coupling between FEM and mesh-free approaches as the best solution.

For installation, compilation, and execution of the code, please refer to README.txt in the root folder for more information.

1.2 Run in Terminal

It is noted that MuPhiSim can be executed in an arbitrary folder. One can add the path to directory MuPhiSim-dir/bin in $PATH variable of the system in order to make MuPhiSim executable (*MuPhiSim*) available everywhere by adding the following line in *.bashrc*

```
export PATH=path-to-MuPhiSim-dir/bin:$PATH
```

To run MuPhiSim

- in the working folder when TYPE=SEQUENTIAL:

```
MuPhiSim-executable -input example.inp
```

- in the working folder folder when TYPE=PARALLEL, e.g. run with 3 processors:

```
mpiexec -np 3 MuPhiSim-executable -input example.inp
```

It is noted that by default, *MuPhiSim* uses two sub-folders respectively named as *input* and *output* to store the input file and the all output files. Other locations can be considered by using the following options

```
-inputDir indir -outputDir outDir
```

where *inDir* and *outDir* are user-defined.

There are others options can be provided from commandline:

- *-MMScalingFactor value* to set the meshless scaling factor by a double specified by *value* [by default: 1]
- *-MMIntegOrder n* to set the integration order of the meshless background mesh by a positive integer *n* (0,1,2,...), [by default: 0; the order decided by element type].
- *-MMAdaptiveSupportRadius n* to activate (1 is used)/deactivate (0 is used) the adaptive support radius in meshless simulation [by default: 1].
- *-noImplicitIterative n* to activate (1 is used)/ or deactivate (0 is used) the iterative process in the implicit solver [by default: 0]. This allows to do only one iteration per each time step in the implicit scheme, *e.g* in the linear cases.
- *-petscOptionsFile fileName* to specify the options of petsc [by default options.petsc in the current folder].

1.3 Debug with *ddd*

To debug the code with the test *example.inp*:

- In the working folder when TYPE=SEQUENTIAL:

```
ddd --args MuPhiSim-executable -input example.inp
```

- In the working folder folder when TYPE=PARALLEL, e.g. debug with 3 processors:

```
mpiexec -np 3 ddd --args MuPhiSim-executable -input example.inp
```

A number of ddd windows will appear corresponding to the number of processors. To start the simulation, simply type *r* in each window or click on Run in Command tool. Breakpoints can be added wherever in the code, and source files can be opened with "File->Open Source..." and click on "Cont" in Command tool.

1.4 Automatically build and test with Bitbucket pipeline

Bitbucket pipeline can be used to automatically build, test, and even deploy code based on a configuration file in the repository. In the root directory of MuPhiSim in Bitbucket website, go to Pipelines → Build C++ Application, copy-paste the following file:

```
image: ubuntu

pipelines:
  branches:
    master:
          - step:
            name: Build and Run
                script:
                  - export MuPhiSim_DIR=$PWD
                  - export DEBIAN_FRONTEND=noninteractive
                  - apt-get update && apt-get install -y git libpython3-dev python-
                    ↪ is-python3 libopenmpi-dev make gcc g++ gfortran liblapack-
                    ↪ dev libblas-dev libmetis-dev
                  - git clone -b release https://gitlab.com/petsc/petsc.git petsc
                    ↪ && cd petsc && git checkout v3.16.3
                  - export PETSC_DIR=$PWD && PETSC_ARCH=linux-g-c-opt
                  - ls $PETSC_DIR
                  - ./configure --with-debugging=0 --with-mpi=1 --download-
                    ↪ fblaslapack=1 --download-scalapack=1 --download-mumps=1 --
                    ↪ with-shared-libraries=1
                  - make
                  - cd $MuPhiSim_DIR
                  - export OMPI_ALLOW_RUN_AS_ROOT=1
                  - export OMPI_ALLOW_RUN_AS_ROOT_CONFIRM=1
                  - export OMPI_MCA_btl_vader_single_copy_mechanism=none
                  #compile the code
                  - make clean&& make TYPE=PARALLEL
                  - make test TYPE=PARALLEL
                  - make clean && make TYPE=SEQUENTIAL PETSCMPI=YES
                  - make test TYPE=SEQUENTIAL
```

First a *ubuntu* docker image is loaded. Then *MuPhiSim* is installed in this new system by the subsequent lines. By default, each time a new commit is pushed, the code is automatically built and tested.

1.5 *MuPhiSim* folders

MuPhiSim is a modular program in which each module is completely independent of the others. The natural split led to the following folders structure (all of them from the seed folder):

1. **doc**: all documentation related files will be placed in this folder. There are two different documentation types related to the program. The first is related to the user manual and corresponding latex files. The second is a Doxyfile used to generate the code documentation with Doxygen (latex and html files). Typing *make documentation* will create the doxygen files.

2. **lib**: all external libraries used by *MuPhiSim* will be placed in this folder. Until now, only Sukumar's library exists to compute the MM shape functions. However, it was modified,

adapted and improved for *MuPhiSim*.

3. **src**: the source code of the program is in this folder. It is also split into its natural parts:
   (a) **constitutiveModels**: All new constitutive models should be placed in this folder (and the makefile modified accordingly)
   (b) **IO**: All new input/output related files should be placed in this folder (and the makefile modified accordingly)
   (c) **mathematics**: All new mathematical files should be placed in this folder (and the makefile modified accordingly)
   (d) **solvers**: All new solvers should be placed in this folder (and the makefile modified accordingly)
   (e) **spatialDiscretisation**: All new element or geometrical entity files should be placed in this folder (and the makefile modified accordingly)
4. **applications**: this folder contains examples and all tools related to *MuPhiSim*

1.5.1 Naming variables, functions and files

As a general rule, all variables, functions and files with two words or more should be written continuously, without any spaces (or underscores). The first letter of a word will be lower case, while the first letter of the rest of the words will be in capital letters, e.g., viscoElasticGrowth.cpp, setIntVars().

1.6 The structure of MuPhiSim

Figure 1.2 shows the general structure of *MuPhiSim* which is similar to a conventional FEM software. The main difference lies in the calculation of the neighbourhood and shape functions of MM gauss points and nodes.

Figures 1.3 to 1.5 shows the structure for the explicit and implicit solvers (the implicit static solver is a minor transformation of the implicit solver). Again, they are similar to a conventional FEM software except for the use of nodes, GPs and shape functions due to the inclusion of MM part. In addition, another main difference lies in the visualisation of the displacement due to the fictitious displacement output from MM nodes, therefore, an extra step to calculation the real displacement is needed for visualisation purpose.

1.7 Important programming concept needed for developers

For beginners (or simply people without experience in programming), the most important programming concept needed to play (continue developing) with *MuPhiSim* is the abstract classes. An abstract class cannot be instantiated and is used here as a wrapper. An abstract class contains pure virtual functions that are implemented by the subclasses that inherit from the abstract class, see Figure 1.1.

In order to illustrate how this technique is used here see Figure 1.1. The class A contains two pure virtual functions (that makes the class A abstract). The class A does not implement any of its virtual functions. Instead, all virtual functions must be implemented by the subclasses B and C. For example, let's imagine that the class A corresponds to a classElements (where we do not care about

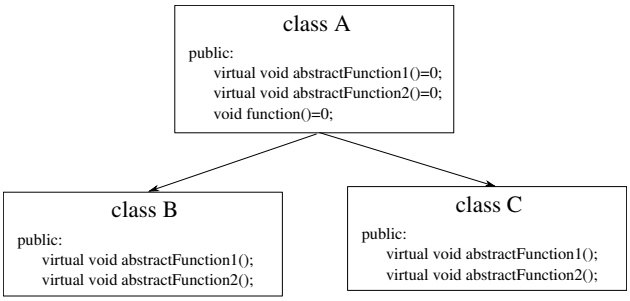


Figure 1.1: Abstract class. Used through the whole program

the type of elements). Then the subclass B becomes classTriangles, in which the virtual functions are implemented, for example, the function to get the nodes that build the element (which depends on the type of elements). For the subclass C, that can be classTetrahedra, the implementation of the virtual functions are different, for example, the function to get the nodes, but the declaration of the functions is common for both subclasses. This so-called virtual inheritance is exploited through the whole program. We can plug another subclass D, that can be other type of element, as long as ALL virtual functions of the abstract class are implemented. The documentation generated by Doxygen (make documentation) can be very useful to follow the class trees.

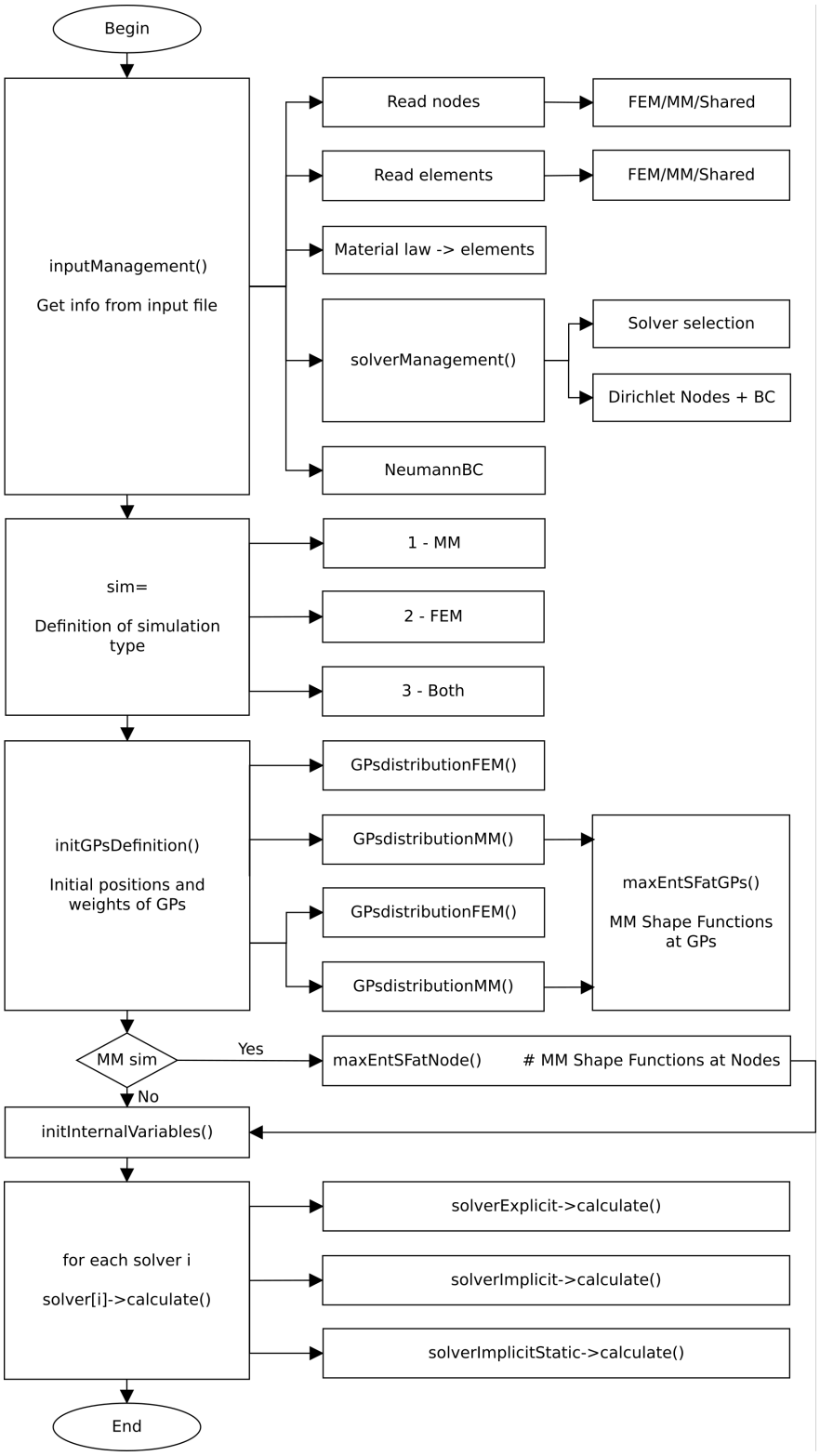


Figure 1.2: General structure of MuPhiSim

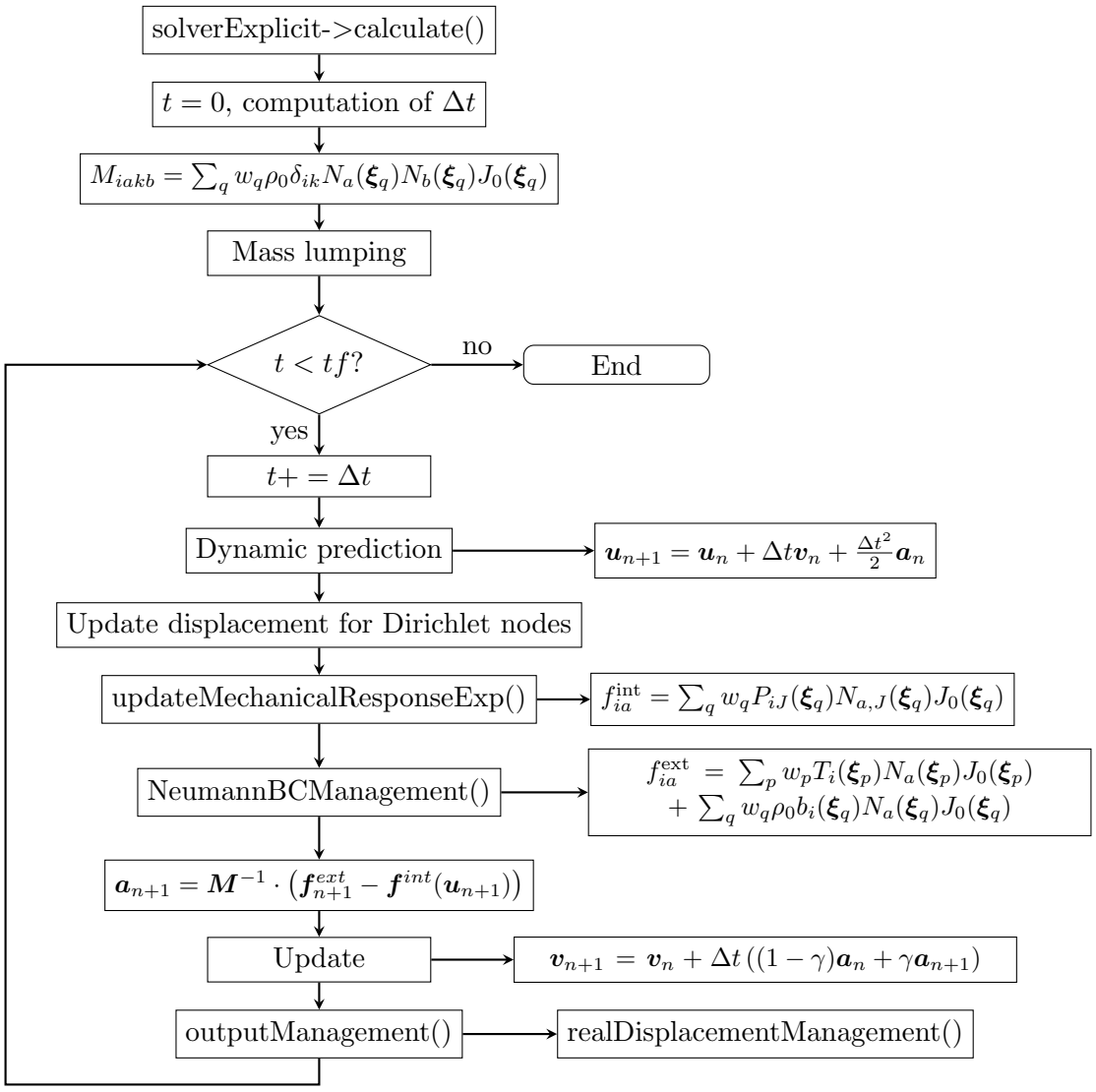


Figure 1.3: Structure of MuPhiSim Explicit Solver

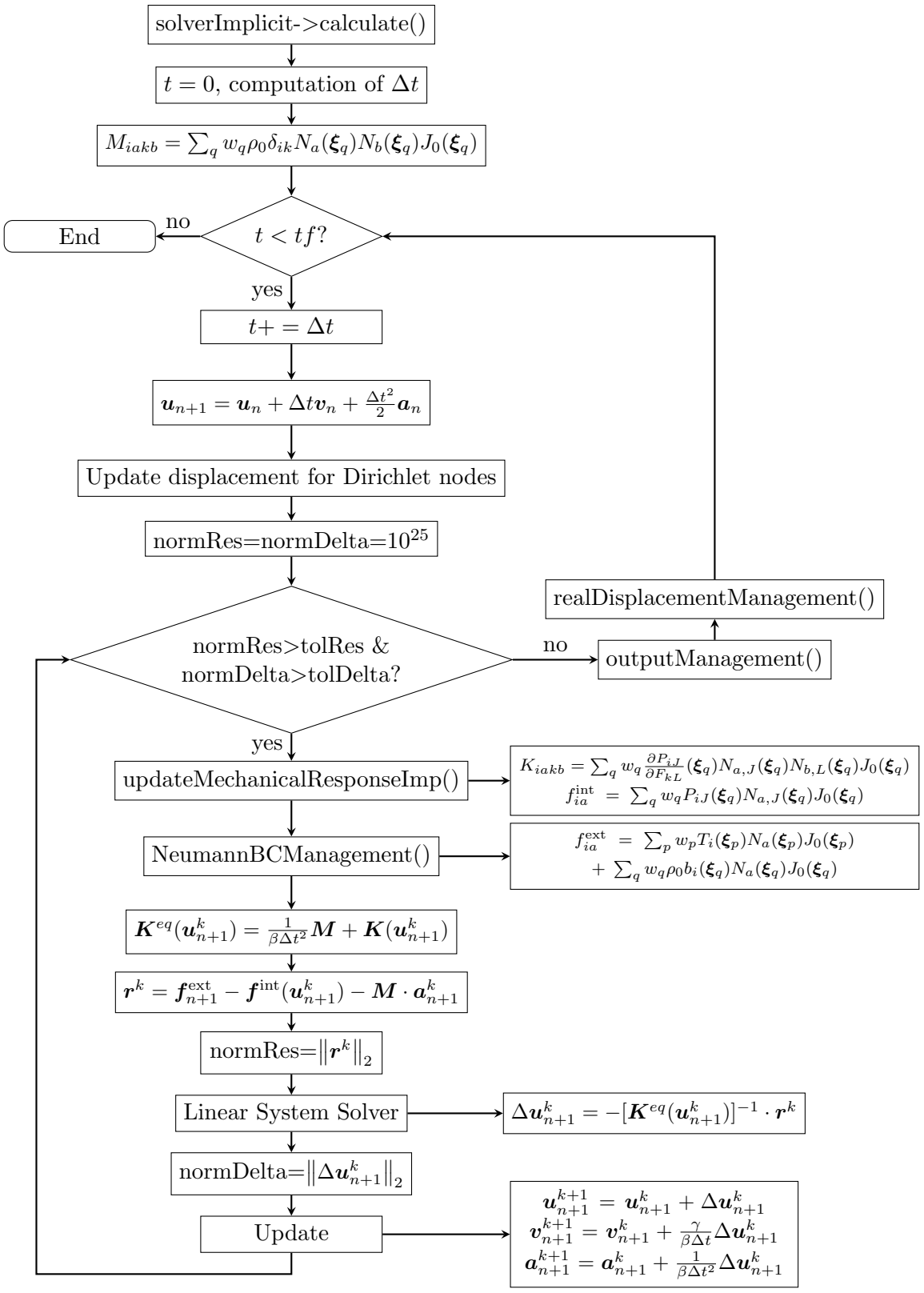


Figure 1.4: Structure of MuPhiSim Implicit Dynamic Solver

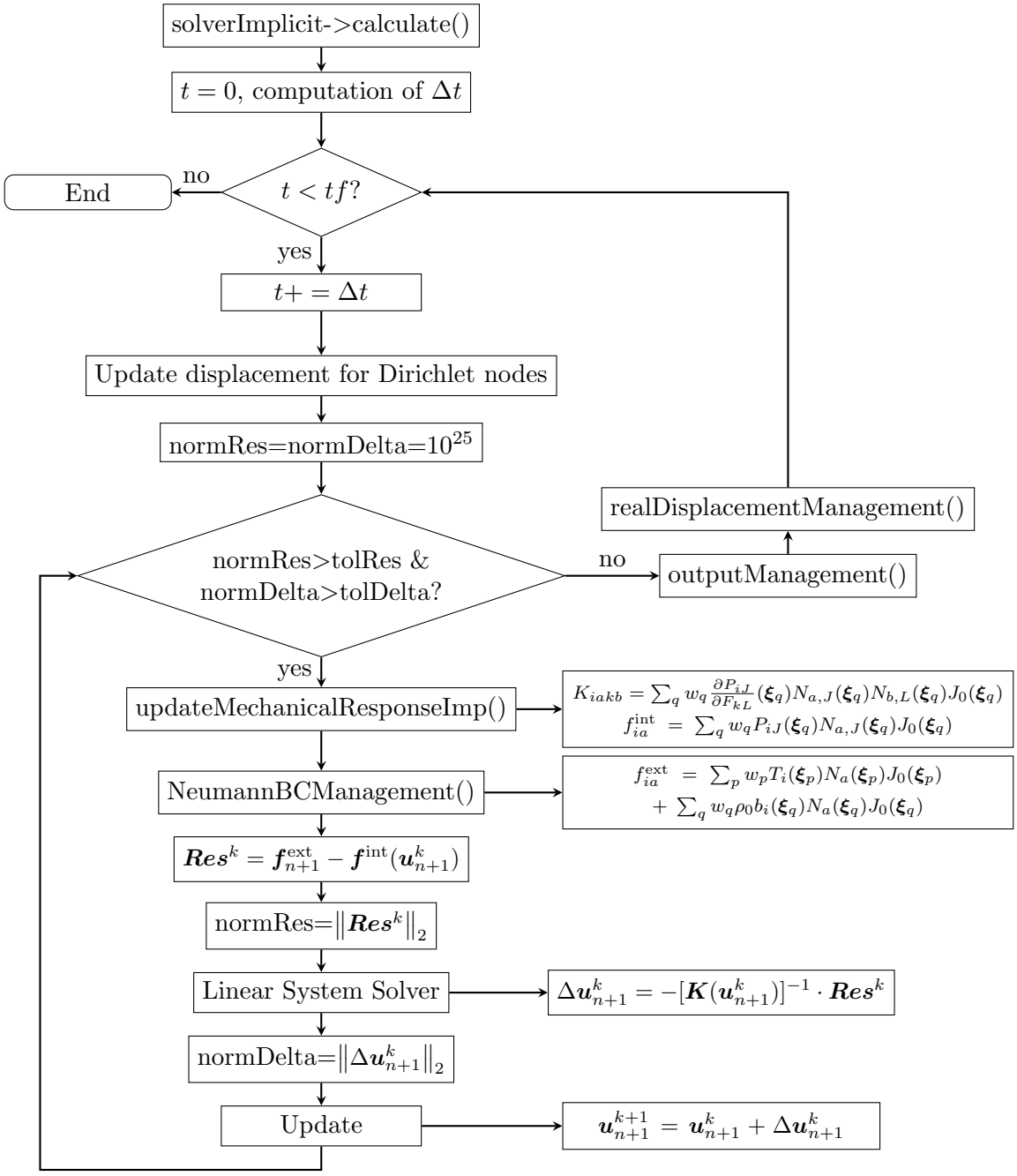


Figure 1.5: Structure of MuPhiSim Implicit Static Solver

CHAPTER 2

INPUT FILES

2.1 Introduction

MuPhiSim follows the Abaqus .inp format with some differences that are clearly specified in the following sections. The input file must be inside the input folder and should be named with the .inp extension e.g., `example.inp`. The program is case sensitive and every single space or comma can make a difference. Follow the next sections thoroughly in order to avoid problems. As a general rule, it does not matter in what order the keywords appear in the input file. However, several keywords must be grouped (they will be highlighted in the following sections). All keywords will be preceded by a star symbol *****. Although *MuPhiSim* is programmed in C++ and thus follows zero notation in the arrays, that is fully independent of the input file, and the numbers that appear therein. Therefore, the enumerations within the input file must start from 1. All columns are separated by commas in the input file. There is also an option to generate a *MuPhiSim* input file using a GUI tool, although the users are strongly advised to read through the manual input section and familiarize themselves with the specifics of *MuPhiSim* input.

2.2 Python code: python-gmsh

The input file can be easily created using *python-gmsh* python project located in *tools/python-gmsh*. The mesh is created using Gmsh (http://gmsh.info/).

Please refer to *tools/python-gmsh/ReadMe.md* for more information.

2.3 GUI

2.3.1 Description

This tool reads Abaqus input files, takes input from the user and produces a corresponding *MuPhiSim* file. To define a certain domain in *MuPhiSim*, the input file has to contain a fully written out list of nodes/elements. This tool was initially meant to simply read all such sets from Abaqus and print them out, but has since grown into an application that allows the users to create full *MuPhiSim* input files with aid of a graphical user interface. It should be noted that the tool

does not necessarily encapsulate all features of *MuPhiSim*, and does not automatically include all updates done to the main code, such as for example addition of new constitutive models. Use this tool with a pinch of salt, and refer to the manual input section if in doubt.

2.3.2 Note to developers

The tool consists of two Python scripts, MuPhiSiminput.py and parser.py. Parser.py handles the logic, while MuPhiSiminput.py builds the GUI and reads the user input. Most minor changes, for instance addition of a material model, will be done in the MuPhiSiminput.py script, in this case, a simple modification of the material box class. However, the developer should look at the relevant section of the parser script, in case the modification to the input is incompatible with its processing.

2.3.3 Installation

The parser tool can be found in tools/GUI folder, and requires Python 3.0 or higher to run. It also requires the Tkinter module, which can be installed by running the following console line:
$sudo apt-get install python3-tk
To launch the GUI app, run:
$python3 MuPhiSiminput.py

2.3.4 Usage

1. **File I/O** - After running the script, a window will appear prompting the user to open an Abaqus file. Select the relevant file, as the app will get all the nodes, elements and sets. Afterwards, the main GUI window with an *MuPhiSim* logo should appear, such as shown in Figure 2.1. Choose where the output should be written by using the "Save as" button, and if no file with the specified name exists, such file will be created.

2. **Region Set** - Define the FEM and/or MM regions using the drop-down menu, which will contain all sets created in Abaqus. One may select both MM and FEM sets, or if all nodes/elements are to be used, simply select "All". The user can also create custom computation partitions for the mesh. To do so, select the number of processors that will be used, and assign node and element sets to each processor. Also surface order allows the user to control the integration order for surface elements. Keep in mind that surface order must be less than or equal to overall bulk order.

3. **Solver Options** - Type in the desired number of solvers, and click "Confirm". Fill in all the boxes using appropriate format, if unclear, refer to the manual input section. The time is specified for each solver, and the parser will produce the correct ranges in the *MuPhiSim* file, in the case of Figure 2.1, [0,1] for the first solver and [1,3] for the second. The BC displacement sets should be input manually, as there can be several. These sets should be separated by commas, keeping in mind these are sets of nodes, make sure the name is exactly as it appears in the Abaqus file. One can get a complete list of all sets by simply scrolling through one of the selection boxes for sets. "BC DOFs" refers to the constraints that should be applied to the named BC sets. These should be input as ordered tuples separated by commas. If one wishes to constrain only some of the DOFs in that set, the other DOFs should be substituted by a question mark (?). The number of DOFs is determined by the nature of the simulation,

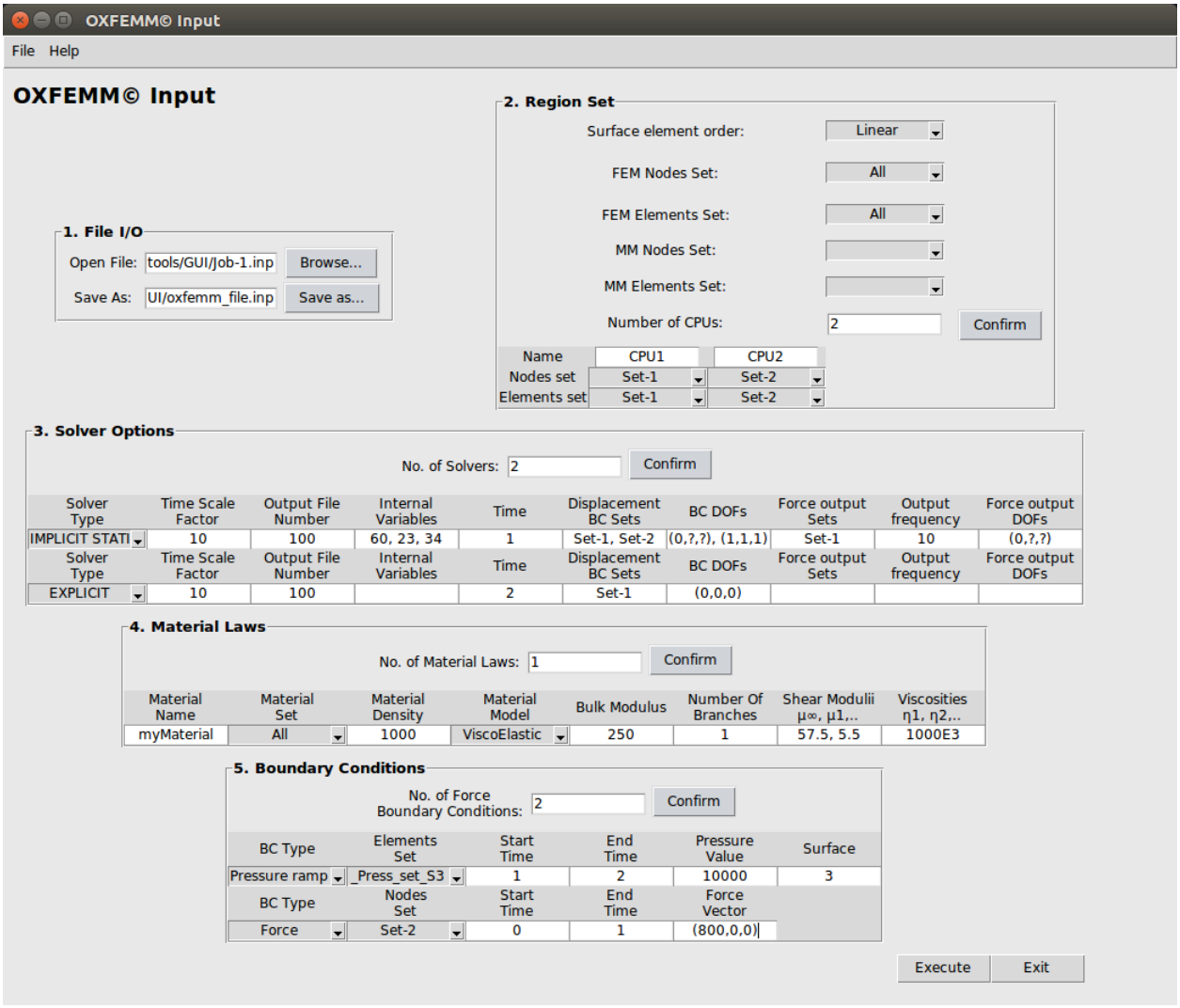


Figure 2.1: GUI window

so for a 3D simulation with an extra DOF, the user can input up to 4 DOFs per tuple. For Figure 2.1, the first solver will have only constrained all first DOFs in Set1 to 0, and all second DOFs of all nodes in Set2 to 1 unit. The second solver fixes first, second and third DOF in Set1 to 0. Forces output allows the user to print the equivalent forces for Dirichlet nodes. The user specifies the frequency of the output (frequency of 2 means that forces will be written twice per unit time), the set of Dirichlet nodes as well as the DOF of interest. Question mark again omits the DOF, while any number will enable that particular DOF. So in order to write only z-direction, the input can be (?,?,1). The forces are defined inside the solver environment, and each solver will produce a .csv file in the output folder.

* In order to avoid duplicating BC on the shared nodes, for the cases with FEM and MM elements / nodes, the first set to be entered must be the one in which we are going to define the desired BC for the shared nodes.

4. **Material Laws** - Select number of materials to be present in the simulation, and press "Confirm". Fill in the boxes, keeping in mind that every element present in the simulation

should have exactly one material assigned to it. However, extra DOFs are defined in the material section, so that does not apply to previous statement. After choosing a material model, fill in all parameters, separating them with commas if needed.

5. **Boundary Conditions** - Select the number of Neumann boundary conditions desired, keeping in mind these are not dependent on the number of solvers, as they are defined outside the solver environment and rather depend on time. Pressures are applied to surfaces of elements, and these surfaces should be entered into the "Surface" box. If the elements to which pressure should be applied is defined in Abaqus using "Pressure" loading condition, then once the file is loaded into the GUI, the surface to which pressure is applied should appear next to the name of the set. In the example used, the set was named "pre", but appears as "_pre_S2", which means that the surface of interest is 2. Nodal force is quite trivial to use, simply select the set of nodes, time interval, and define a force vector. Moreover, it permits to define the different Neumann BC for extradof. Current Inst, define the Neumann BC for electrophysiology. The user has to select the set of elements, time interval, and define a amount of current. Moreover, heat inst, define a surface heat flux for heat equation. As well as pressure, surface heat flux is applied on a surface. The user has to select the surface to which heat flux is applied (this should appear next to the name of the set) , time interval, heat flux vector (the last term of the vector will be zero for 2D) and the surface of interest (see Figure 2.2).

When the input is complete, click "Execute", and the *MuPhiSim* file should be written to the location specified by the user.

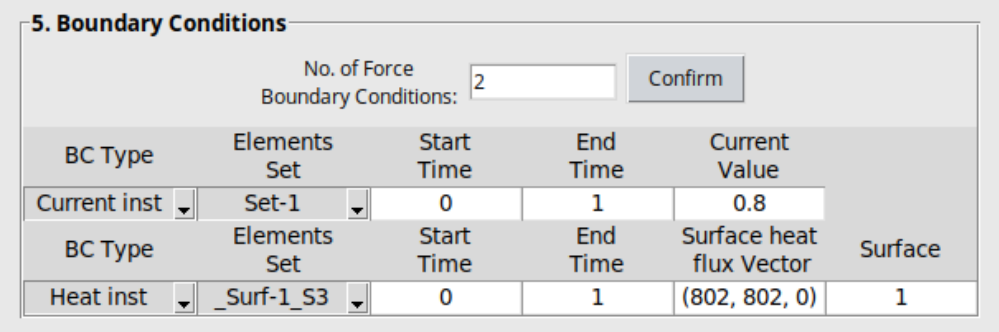


Figure 2.2: GUI window-extradof boundary conditions

2.4 Manual input

2.4.1 Nodes

The list of nodes should begin with the keyword ***Node** followed by the list of nodes enumerated in the first column and then each coordinate 2D/3D in the following columns (exactly the same format as Abaqus).

Example:

```
*Node
1, 0.4, 0.9, 4
...
```

2.4.2 Elements

The list of elements should begin with the keyword ***Element, type=typeOfElement** followed by the list of elements enumerated in the first column and then each node that builds the element should be specified in 2D/3D (exactly the same format as Abaqus). The names of the elements follow the Abaqus nomenclature. The elements available so far in *MuPhiSim* are the following:

1. Linear segments **typeOfElement=Segments** (1D) It is used for the boundaries (not to carry out simulations in 1D)
2. Linear triangles **typeOfElement=CPS3** (2D)
3. Linear squares **typeOfElement=CPE4** (2D)
4. Linear tetrahedra **typeOfElement=C3D4** (3D)
5. Linear hexahedra **typeOfElement=C3D8** (3D)
6. Quadratic triangles **typeOfElement=CPS6** (2D)
7. Quadratic tetrahedra **typeOfElement=C3D10** (3D)
8. Quadratic triangles with linear surfaces **typeOfElement=CPS6_lin_surf** (2D)
9. Quadratic tetrahedra with linear surfaces **typeOfElement=C3D10_lin_surf** (3D)

Example:

```
*Element, type=C3D4
1, 227, 137, 219
...
```

Implementation of new type of elements

Following the concepts shown in Section 1.7, the inheritance diagram for the classElements is shown in Figure 2.3. All new elements will be new classes that inherit from classElements. It is recommended to copy/paste an existing type of element and modify it accordingly. See the Doxygen documentation to have the list of virtual functions and what they do. This is easy to do by navigating the html files generated by Doxygen.

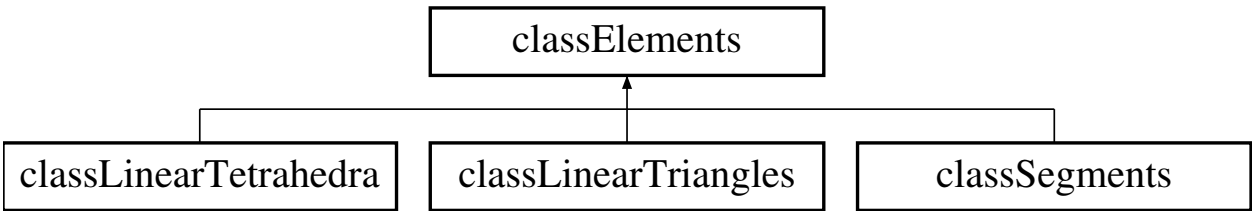


Figure 2.3: Class elements inheritance diagram

When a new type of element is added, the input file should be also modified accordingly. It is recommended to copy/paste the same structure of the input file of the existing type of elements.

2.4.3 Distribution in FEM and/or MM

The list of nodes and elements of each type should be provided (in the future, some algorithms will be implemented to define such distribution). The keywords are ***FEM/*MM Nodes** and ***FEM/*MM Elements** followed by the list (one label per line) of nodes/elements by means of the corresponding label of the full list of nodes/elements. If all nodes/elements in the simulation are FEM/MM the keyword **All** can be used instead of repeating the whole list. This part is new compared to the traditional input files for Abaqus.

2.4.4 Computation region assignment in parallel

In parallel mode, *MuPhiSim* automatically partitions the mesh and assigns the regions to the processors. However, there is an option for the user to assign what region the different processors will be responsible for. To do so, the user first assigns all the nodes to the processors, and then all the corresponding elements. The syntax for defining a region for i-th CPU is as following:

```
*NSET="Name of the region"
*CPU=i
```

where $i \in [1, n]$ and n is the total number of CPUs in the simulation. "Name of the region" should be replaced by a user-defined name, and all the subsequent lines should list the nodes in that region. Similarly, the elements that the i-th CPU is responsible for should be written using the following syntax:

```
*ELSET="Name of the region"
*CPU=i
```

and the following lines should list all elements in that region. Below is a sample input for 2 processors:

```
*NSET=region1
*CPU=1
1
2
5
6
*NSET=region2
*CPU=2
2
3
4
5
*ELSET=region1
*CPU=1
1
2
*ELSET=region2
*CPU=2
3
4
```

The user should keep in mind that in case of manual partitioning, the input file must contain the same number of CPUs as specified when running MuPhiSim in the command line.

2.4.5 Solver

The choice of the solver should be defined by the keyword ***SOLVER, TYPE**, where **TYPE** can be either **EXPLICIT**, **IMPLICIT** (dynamic) or **IMPLICIT STATIC**. The keyword ***END SOLVER** should be at the end of the solver definition. The number of solvers is not limited, and the solvers must be distributed continuously along the time of simulation.

If implicit and explicit (dynamic) solvers are used, the keyword ***FSI** can be used to run a coupled simulation with OpenFOAM. It must be followed by the list of nodes at the coupling interface by means of the corresponding label in the full list of nodes. See chapter 13 for more details.

A C++ file - input_file_generation.cpp - writes the input file for the simple example implemented in the **quickstart** folder. It generates the input file directly at the right place, i.e. the input folder of *MuPhiSim*.

The keyword ***Scale Factor** can be used to define the scale factor to be applied to the critical time step. Actually, **dt = <Scale factor> * <Critical timestep>**. The value should be in the next line after the keyword.

The keyword ***Number Of Steps** can be used to define the number of time steps in the implicit and implicit static solvers. Once ***Number Of Steps** is given, ***Scale Factor** is not used.

The keyword ***TIME STEPPING PLAN** can be used to define the time stepping plan in static solver. For example, a simulation from 0 to 1s following the plan: 0s-0.1s with 10 steps, 0.1s-0.5s with 3 steps, and 0.5s-1s with 50 steps can be performed with the following plan

```
*TIME STEPPING PLAN
0.1, 10
0.5, 3
1, 50
```

It is noted that ***Scale Factor** must be used in an explicit scheme to define the time step. In the implicit and implicit static schemes, the time step is defined first by the first line in ***TIME STEPPING PLAN**. If it is not available, ***Number Of Steps** is used. If the latter is still not available, ***Scale Factor**.

The maximum time step can be constrained with the keyword ***Max Time Step**. For example, we can limit the time step in the implicit and implicit static simulation by a value of 0.01 using

```
*Max Time Step
0.01
```

The automatic adaptive time stepping can be considered by using the keyword ***ADAPTIVE TIME STEP**. The idea is to adapt the time step for next step based on the convergence of the current step by a general relation

$$\Delta t_{new} = \mathrm{f}(\Delta t_{prev}, ...)$$

In the current version, only the power law is implemented as

$$\Delta t_{new} = \Delta t_{prev} \left(\frac{N_{opt}}{N_{iter}^{prev}} \right)^n$$

where N_{iter}^{prev} is the number of iterations to converge the solution in the current step, N_{opt} is its expected value, n is a positive real number. Using this function, the time step increases if the number of iterations required to converge the solution is smaller than the expected value and vice-versa. In the input file, the following line is added to activate this option

```
*ADAPTIVE TIME STEP
POWER,6,0.5
```

where $N_{opt} = 6$ and $n = 0.5$.

The keyword ***Absolute Tolerance** (10^{-12} by default) can be used to define the absolute tolerance in the implicit and implicit static solvers.

The keyword ***Relative Tolerance** (10^{-6} by default) can be used to define the relative tolerance in the implicit and implicit static solvers.

The keyword ***Max Iterations** (50 by default) can be used to define the maximal number of iteration in each time step in the implicit and implicit static solvers.

The keyword ***STIFFNESS MATRIX** is used to define the strategy of modifying the stiffness during an implicit (static) scheme. By default, the stiffness is updated every iteration at every time step. In some case, the modification of the stiffness can be planned as follows

- The stiffness matrix is updated all time steps and iterations (this is also the default option in the solver)

```
*STIFFNESS MATRIX, TYPE=CURRENT
```

- One single stiffness matrix for all subsequent time steps and iterations

```
*STIFFNESS MATRIX, TYPE=BEGINNING
```

- The stiffness matrix is updated only at the beginning of each time step

```
*STIFFNESS MATRIX, TYPE=INITIAL
```

- The stiffness matrix is updated after kth iteration of each time step

```
*STIFFNESS MATRIX, TYPE=ITERATION
k
```

- The stiffness matrix is updated after each n iteration of each time step

```
*STIFFNESS MATRIX, TYPE=INTERVAL
n
```

It is noted that the **TYPE=BEGINNING** or **TYPE=INITIAL** should be used, as it reduces the computation cost (LU factorisation) since the matrix is factorised once but re-used through iterations without the need of re-factorising.

The stiffness matrix can be computed by perturbation without the need of the computation of tangents at the material law. This option can be activated by

```
*TANGENT BY PERTURBATION
1e-8
```

where 1e-8 is a tolerance which can be introduced arbitrarily. This option is a very efficient way to obtain a good convergence rate.

If explicit solver is used, the keyword ***Bulk viscosity** or ***Dev viscosity** can be used to add an artificial bulk or deviatoric viscosity when first Piola-Kirchhoff tensor is computed. The default parameters for linear and quadratic coefficients are {0.06,1.2} for bulk viscosity and {1,0.5} for deviatoric viscosity. The user can change these values adding a second line.

Example :

```
*Bulk viscosity
0.08,1.2
 ...
```

The keyword ***Outputs** can be used to define the number of output files if the number of time steps allows it. The value should be in the next line after the keyword.

In addition, there can be a second line after this keyword that relates to the internal variables. All the internal variables can be stored using **All, max** in that line, where **max** is the maximum number of internal variables. Some specific internal variables can be stored using their positions in the intVar array, e.g., **60, 2, 5** will store internal variables 2 and 5, while 60 is the maximum size of the internal variables. If the constitutive model does not have internal variables, zeros will be printed out in the output files. The default outputs are VMS, displacements and Cauchy Stress.

Example :

```
 *Outputs
All, max
60, 2, 5
 ...
```

In a possible third line (without which, any internal variable will be given in the outputs), the first value is the maximum number of internal variables in the whole domain (regardless if there is more than one constitutive model). For example, if there is one constitutive model with 8 internal variables and another with 2, 8 must be provided to the program. It is the responsibility of the user to specify how the internal variables are stored in the intVars array. One 3x3 tensor corresponds to 9 spaces in the vector, etc. In each output file, the requested variables will be in a vector intVars, where each component corresponds to the specified position. After a comma, all the components which are desired in the output files must be provided in the intVars array. If position 6 is requested in the previous example, in the output, the elements with the constitutive model 1 (which had two variables) will have zeros assigned to these positions. If the constitutive models have some common internal variables that it would be useful to compare, make sure that they are placed in the same positions of the internal variable arrays, in order to be able to compare them in Paraview. Finally, if all internal variables are required, simply use All in the third line, and after a comma, provide the maximum number of internal variables in the domain.

The keyword ***Time** is used to define the time of the simulation for this particular solver. The initial and the final times should be in the next line, separated by comma.

Dirichlet boundary conditions can be defined as a progressive ramp between t_0 and t_f or instantaneously at t_0. For the latter, in implicit and explicit cases, the velocity is defined considering the time step instead of the total time. The keywords are ***BOUNDARY, TYPE=DISPLACEMENT RAMP** and ***BOUNDARY, TYPE=DISPLACEMENT INST**. The detection of instantaneous BC actives a boolean flag in inputFile.cpp

```
} else if (line2.compare(0, 33, "*BOUNDARY, TYPE=DISPLACEMENT INST") == 0) {
flagInstantaneous = true;
```

this flag defines the construction of classDirichlet as

```
if (flagInstantaneous) {
   //if flagInstantaneous = true Dirichlet BC are applied instantaneously
   tempDirichlet = new classDirichlet(ndim, tempme, tempxyz, UU, direction, numNodes);
} else {
    //if flagInstantaneous = false Dirichlet BC they are applied as a ramp
    tempDirichlet = new classDirichlet(ndim, tempme, tempxyz, UU, direction, numNodes,
    duration);
}
```

Depending on the definition of Dirichlet BC, an integer, called *flagDiri*, is be defined as 1 (ramp) or -1 (instantaneous) in classNodes, that will be used subsequently to apply the BC in each specific solver.
To specify the list of nodes with their corresponding displacements, each node, direction [0-x, 1-y, 2-z], and the corresponding value of the displacement should be provided, e.g., *node label*, $1, 0.1$ which is read as a displacement in the y direction of 0.1 meters.

In a multiphics simulation, in which the keyword ***ExtraDof** is specified later in the material field, the boundary conditions of the extra-dofs can be also specified. For example, in an electro-mechanical or a thermal-mechanical problem, the electrical Dirichlet boundary condition Φ or temperature θ may be specified at the end of the line [0-x, 1-y, 2-z, 3-Φ] in 3D or [0-x, 1-y, 2-Φ] in 2D. If there are more than one extradof, e.g. electro-thermal-mechanical problem, the second extradof (and successive) Dirichlet boundary condition may be specified at the end of the line [0-x, 1-y, 2-z, 4-variable] in 3D or [0-x, 1-y, 3-variable] in 2D.

The keyword ***Forces** can be used to create another output file (format .csv) where the user can print the nodal forces among x, y or z. On the first line, the user specifies the frequency of outputs (outputs per unit time). The user then specifies on the following lines the nodes that will be selected and for each node, the direction (0 for x, 1 for y and 2 for z).

Example:

```
...
*Forces
0.01
1, 0
2, 0
```

```
3, 0
4, 0
1, 1
2, 1
3, 1
4, 1
...
```

Finally, the user can choose which PETSc solver and preconditioner are employed to solve the system using the keywords ***PETSC_SOLVER** and ***PETSC_PC**. The complete list of PETSc solvers and preconditioners can be found in the following links:
`https://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/KSP/KSPType.html#KSPType`
`https://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/PC/PCType.html`

The following example shows the assignment of the iterative solver Conjugate Gradient (cg) with the Jacobi preconditioner (jacobi):

```
...
*PETSC_SOLVER, cg
*PETSC_PC, jacobi
*END SOLVER
...
```

Note that if the user does not introduce any PETSc solver and preconditioner through the keywords in the input file, the default iterative solver Conjugate Gradient (cg) is assigned with the default preconditioners LU and block Jacobi (bjacobi) for the sequential and parallel simulations, respectively. In general, the direct LU solver (by default in the code) is recommended for both sequential and parallel simulations.

Multisolvers

To have two or more successive solvers, simply add their definition chronologically. The only requirement is that the time interval of each solver should be in agreement with each other without any gap.

Example:

```
*SOLVER, EXPLICIT
*Scale Factor
0.01
*Outputs
100
All, 2
*Time
0, 820
*BOUNDARY, TYPE=DISPLACEMENT
   3, 0, 0
   4, 0, 0
   6, 0, 0
   7, 0, 0
  39, 0, 0
```

```
  101, 0, 0
  109, 1, 100
 110, 1, 100
 140, 1, 100
 347, 1, 100
 348, 1, 100
 349, 1, 100
   1, 0, 0
*END SOLVER
*SOLVER, EXPLICIT
*Scale Factor
0.01
*Outputs
100
All, 2
*Time
820, 10000
*BOUNDARY, TYPE=DISPLACEMENT
   3, 0, 0
 349, 1, 100
   1, 0, 0
*END SOLVER
```

Progressively activation/deactivation of elements

MuPhiSim includes the option to activate/deactivate elements in every step. To use this option, see *Appendix A*.

2.4.6 Neumann boundary conditions

The Neumann BCs are defined globally. They should be defined out of the Solver environment, e.g just after the last solver. Force at the nodes will be imposed instantaneously whereas the pressure can be imposed either instantaneously or progressively as shown on fig. 2.4.

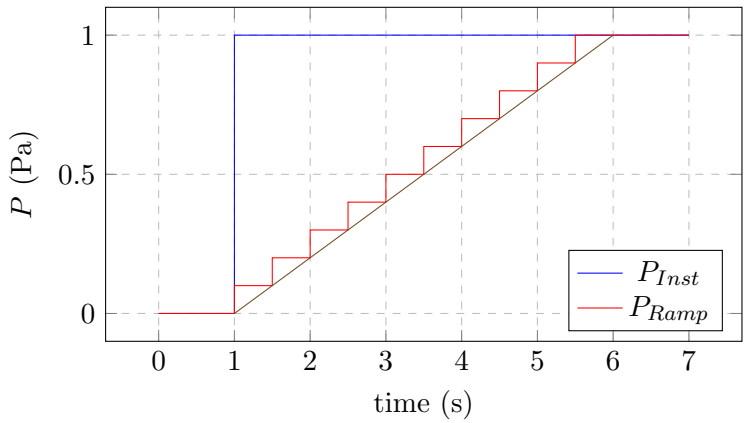


Figure 2.4: Neumann Boundary Condition for pressure, $t_0 = 1$s , $t_f = 6$s , $dt = 0.5$s

2.4.6.1 Nodal forces

The keyword ***BOUNDARY, TYPE=FORCE** will be followed by the t0, tf, and the amount of force F in the next line separated by commas. In the next lines, the list of nodes should be provided together with the specific factors of F in each following column that corresponds to the directions of the force (column 1-x, column 2-y, column 3-z).

Example:

```
*BOUNDARY, TYPE=FORCE
0, 0.1, 1000
15, 0, 1, 0
30, 0, 1, 0
45, 0, 1, 0
60, 0, 1, 0
75, 0, 1, 0
90, 0, 1, 0
105, 0, 1, 0
120, 0, 1, 0
*BOUNDARY, TYPE=FORCE
0.1, 0.2, -1000
15, 0.25, 1, 0
30, 0.25, 1, 0
45, 0.25, 1, 0
60, 0, 1, 0
75, 0, 1, 0
90, 0, 1, 0
105, 0, 1, 0
120, 0, 1, 0
```

For node 15, a force of 1000 will be applied in the y direction from 0 to 0.1, and from 0.1 to 0.2 a load of -1000*0.25 in the x direction and -1000 in the y direction.

2.4.6.2 Pressure

The application of pressure is not straightforward. It is available in 2D and 3D. The application is as follows (see [10]):

$$f_p = \int_{\partial\Omega_c} N_a p \mathbf{n} d\Omega \tag{2.1}$$

where p is the pressure (which is applied incrementally or progressively), N is the shape function and $\mathbf{n}$ is the normal vector at the current configuration. Using Nanson's formula, the equation in the reference configurations reads:

$$f_p = \int_{\partial\Omega_0} N_a J p \mathbf{F}^{-T} \mathbf{n}_0 d\Omega_0 \tag{2.2}$$

where this equation should be integrated numerically.

The keyword ***BOUNDARY, TYPE=PRESSURE INST** is needed for an instant and constant pressure and the keyword ***BOUNDARY, TYPE=PRESSURE RAMP** is needed to make a progressive ramp between t0 and tf.

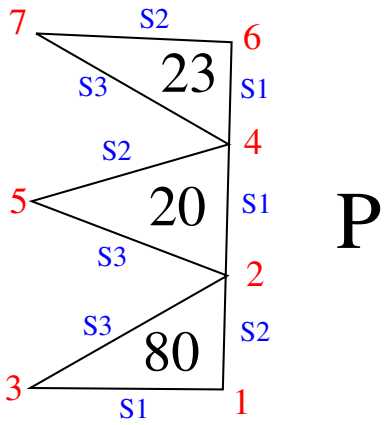


Figure 2.5: Pressure application in 2D

In 2D, pressure on edges

In order to apply the pressure, the edges on which the pressure will be applied should be identified, see Figure 2.5. The pressure P will be applied to that face of the domain. In that region we have the triangular elements 80 (with the nodes: 3, 1 and 2), 20 (with the nodes: 2, 4 and 5) and 23 (with the nodes: 4, 6 and 7), all of them anti-clock wise. The advantage of Abaqus is that the edges of the triangles are always defined going from the first node of the element to the last one, see Figure 2.5. So, for the elements 80, 20 and 23, the subelements S1, S2 and S3 are arranged as shown in the figure. Therefore, to apply the pressure P, it should be applied to the subElement S2 of element 80, and subelements S1 of 20 and 23. The application of the pressure P shown in Figure 2.5 reads:

```
*BOUNDARY, TYPE=PRESSURE INST
t0, tf, P, 1
20
23
*BOUNDARY, TYPE=PRESSURE INST
t0, tf, P, 2
80
```

where in the first line, t0 is the initial time to apply the pressure, tf the final time, P the amount of pressure and the last value corresponds to the surface of the elements listed below.

3D, pressure on surfaces

In order to apply the pressure, the surfaces on which the pressure will be applied should be identified, see Figure 2.6. The pressure P will be applied to that face of the domain. In that region we have the tetrahedron elements E1 (with the nodes 4, 5, 2 and 3) and E2 (with the nodes: 3, 5, 2 and 1), all of them anti-clock wise. The advantage of Abaqus is that the edges of the tetrahedrons are always defined in a specific order, see Figure 2.6. So, for the elements E1 and E2, the subelements S1, S2, S3 and S4 are arranged as shown in the figure. Therefore, to apply the pressure P on the grey area, it should be applied to the subElement S3 of element E1, and subelement S1 of E2. The application of the pressure P shown in Figure 2.6 reads:

```
*BOUNDARY, TYPE=PRESSURE INST
```

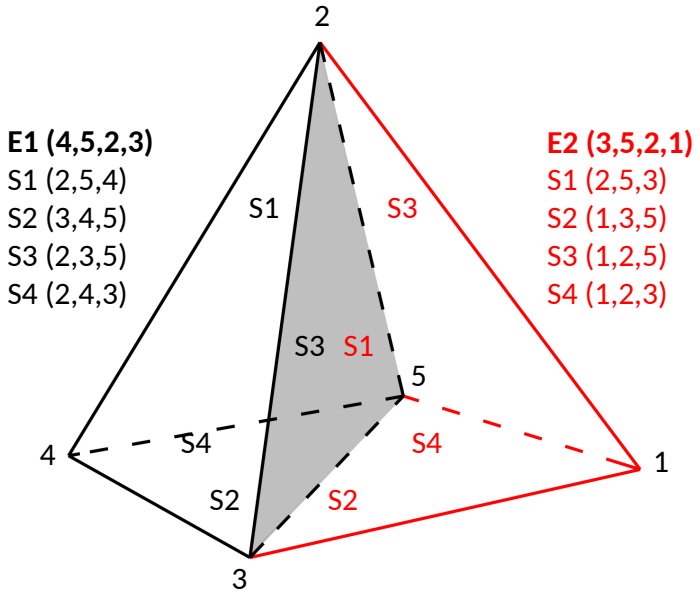


Figure 2.6: Pressure application in 3D

```
t0, tf, P, 1
2
*BOUNDARY, TYPE=PRESSURE INST
t0, tf, P, 3
1
```

where in the first line, t0 is the initial time to apply the pressure, tf the final time, P the amount of pressure and the last value corresponds to the edge of the elements listed below.

2.4.6.3 Traction boundary conditions

The keyword ***BOUNDARY, TYPE=TRACTION INST** is needed for an instant traction and the keyword ***BOUNDARY, TYPE=TRACTION RAMP** is needed to make a progressive ramp between t0 and tf.

The application of the traction boundary condition is available in 2D and 3D. The support is defined in a similar way to that considered in the case of the pressure boundary condition. The application of a traction (Tx, Ty, Tz) on the grey area in Figure 2.6 reads

```
*BOUNDARY, TYPE=TRACTION INST
t0, tf, Tx, Ty, Tz, 1
2
*BOUNDARY, TYPE=TRACTION INST
t0, tf, Tx, Ty, Tz, 3
1
```

2.4.6.4 Contact

The application of contact is not straightforward. The radius of the area of the contact, a, and the friction coefficient μ must be specified in the input file.

$$f_p = \int_{\partial\Omega_c} N_a p \mathbf{n} d\Omega \tag{2.3}$$

$$f_t = \int_{\partial\Omega_c} N_a t \mathbf{n_t} d\Omega, \tag{2.4}$$

where p and t are respectively the inhomogeneous normal and tangential pressure (which are applied incrementally or progressively), N is the shape function and $\mathbf{n}$ is the normal vector at the current configuration. Using Nanson's formula, the equation in the reference configurations reads:

$$f_p = \int_{\partial\Omega_0} N_a J p \mathbf{F}^{-T} \mathbf{n}_0 d\Omega_0 \tag{2.5}$$

$$f_p = \int_{\partial\Omega_0} N_a J t \mathbf{F}^{-T} \mathbf{n_{t0}} d\Omega_0. \tag{2.6}$$

2.4.6.5 Extra dof - Neumann flux

The keyword ***HERTZIAN CONTACT** will be followed by start and end times –t0 and tf– and the total normal and tangential forces applied in the area of contact, the friction coefficient, the radius of contact and the index of the subsurface in the next line separated by commas. In the next line, the coordinates of the center of the contact is specified. In the next lines, the elements should be listed line by line.

Example:

```
*HERTZIAN CONTACT
0,1,800000,200000,0.3,3,3
0,0,2
1
2
3
```

2.4.6.6 Extra dof - Surface heat flux

The keyword ***FLUX EXTRADOF, TYPE=HEAT INST** will be followed by start and end times –t0 and tf–, the values of the components of the diagonal of the matrix that define the surface heat flux $\boldsymbol{q_0}$ (the third components is equal to zero for a 2D case) in the next lines separated by commas and finally, the surface on which the heat flux will be applied. In the next lines, the elements should be listed line by line.

Example:

```
*FLUX EXTRADOF, TYPE=HEAT INST
0, 200, 802, 802, 0, 1
50
```

Notice that different sign nomenclature is used in 2D and 3D. A surface heat flux acting inwards the surface has to be defined as positive for 2D, and negative for 3D.

2.4.6.7 Extra dof - Volumetric heat flux

To define constant volumetric heat flux during the time interval $[t_0, t_f]$ in a group of elements, the keyword ***FLUX EXTRADOF, TYPE=VOLUMETRIC HEAT FLUX** will be followed by start and end times –t0 and tf–, and the prescribed volumetric heat flux per volume r. In the next lines, the elements should be listed line by line.

The heat flux can be also defined as a moving Gaussian source by the keyword ***FLUX EXTRADOF, TYPE=GAUSSIAN VOLUMETRIC HEAT FLUX**. For a heat source whose initial position (x_0, y_0, z_0) moving at a velocity (v_x, v_y, v_z), the current flux reads

$$r = q_{max} \exp(-R)\,, \tag{2.7}$$

$$\text{where } R = (\frac{x - x_c}{a})^2 + (\frac{y - y_c}{b})^2 + (\frac{z - z_c}{c})^2 \tag{2.8}$$

$$\text{with } x_c = x_0 + v_x(t - t_0), y_c = y_0 + v_y(t - t_0), \text{ and } z_c = z_0 + v_z(t - t_0), \tag{2.9}$$

In the input file, this boundary condition is defined as

```
*FLUX EXTRADOF, TYPE=GAUSSIAN VOLUMETRIC HEAT FLUX
t0, tf, qmax, x0, y0, z0, vx, vy, vz, a, b, c
```

2.4.6.8 Extra dof - Surface convection

The keyword ***FLUX EXTRADOF, TYPE=CONVECTION** will be followed by start and end times –t0 and tf–, the film coefficient h, the sink temperature θ^0 and finally, the surface on which the heat flux will be applied. In the next lines, the elements should be listed line by line.

2.4.6.9 Extra dof - Radiation

The keyword ***FLUX EXTRADOF, TYPE=RADIATION** will be followed by start and end times –t0 and tf–, the radiation constant A, which is calculated as emissivity times the Stefan-Boltzmann constant($5.67 \times 10^{-8} \quad W/m^2K^4$), the sink temperature θ^0 and finally, the surface on which the heat flux will be applied. In the next lines, the elements should be listed line by line. It should be noted that the implementation of radiation has been carried out considering kelvin as a temperature scale.

2.4.7 Initial conditions

Initial conditions is an option to initialise the value of the extradof variables, i.e. normalised voltage, temperature etc. If this is not defined in the input file, the variables are initialised to zero.

The keyword ***INITIAL CONDITIONS EXTRADOF** will be followed by the initial value of each extradof, each separated by a comma. Example

```
*INITIAL CONDITIONS EXTRADOF
273
```

Note that these must follow the same order that is used in the definition of materials. That means if we define:

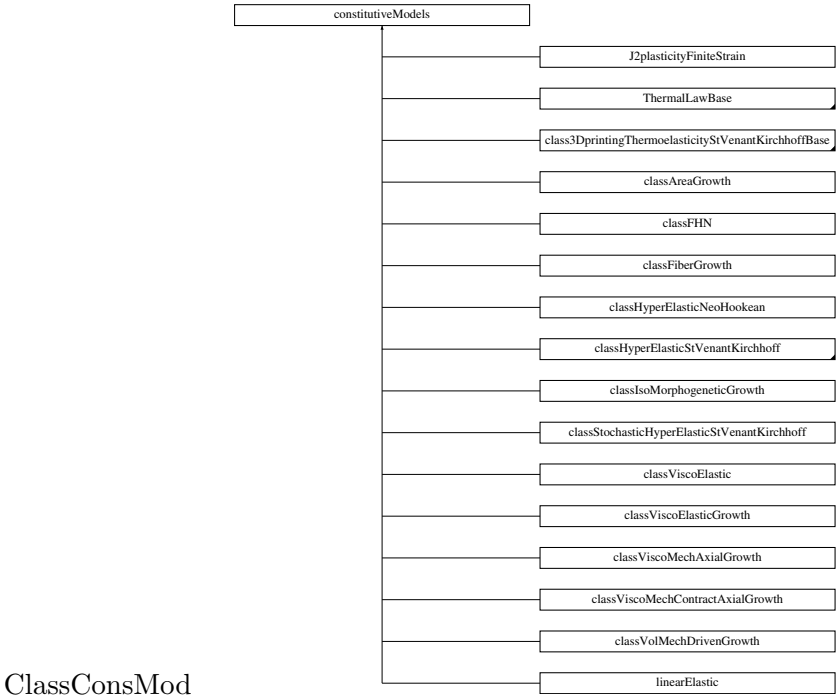
```
*ExtraDof
*FHNelec
1, 0, 0, 1, 0, 0.4, 0.4, 0.01, 0.085, 0
```

```
*ExtraDof
*Temperature
0, 0, 401, 0, 0, 0, 0
 ...
```

The first extradof variable will be normalised voltage and the second one temperature. In addition, this option must be defined before ***MATERIAL**.

2.4.8 Materials

All new constitutive models will inherit from the parent class constitutiveModels, see Figure **??**. The theory which the corresponding constitutive model is based on should be cited in the doxygen file. To implement a new law, some virtual functions specifying in *constitutiveModels.h* that MUST be implemented by the user.



ClassConsMod

Figure 2.7: Class constitutive models inheritance diagram

In terms of materials, each of them should be defined with the keyword ***MATERIAL, name=RegionX** where regionX is the name for the corresponding region. In the next line, the density should be specified with the keyword ***Density** with its value in the next line. In the next line the type of material should be specified. The constitutive models available in *MuPhiSim* are the following:

1. Area growth, keyword ***AreaGrowth**. Next line, the Young's modulus, the Poisson's ratio and the growth multiplier. The perpendicular direction to the growth plane is given in the constitutive model.

2. FitzHugh-Nagumo (FHN), electrophysiology, keyword ***ExtraDof**. This constitutive model is linked with *Electrophysiology.* analysis. Next line, ***FHNelec**. The mechanical coupling is obtained using the keyword ***FHNelecCoupling** in which the constitutive model depends

on the deformation gradient. Only one line is needed to define the model: $a_{o/x}$, $a_{o/y}$, $a_{o/z}$, D_{iso}, D_{ani}, a_{elec}, b, ε, Φ_{equi}, I where:
Axonal direction: $n_o = \{a_{o/x}, a_{o/y}, a_{o/z}\}$
Conductivity tensor, defined by its isotropic and anisotropic components: D_{iso} and D_{ani}
Activation parameters: a_{elec}, b
Time-scale difference: ε
Equilibrium potential: Φ_{equi}
stimulus current: I
Everything must be separated by commas.

3. Temperature, keyword ***ExtraDof**. This constitutive model can be linked to uncoupled heat transfer analysis (*Heat*) or with fully coupled thermal-stress analysis (*Thermomechanics.*). Therefore, in next line, the user has to specify ***Temperature** for uncoupled analysis or ***TemperatureCoupling** for coupling analysis. Only one line is needed to define the model: C_0, C_1, k_0, k_1, θ_{ini}, θ_{ref1}, θ_{ref2} where:
Specific heat capacity at constant volume, reference temperature: C_0
Parameters that described the temperature-dependence of C: C_1 and θ_{ref1}
Heat flux per volume: r
Material conductivity (current configuration) at reference temperature: k
Parameters that described the temperature-dependence of k: k_1 and θ_{ref2}
Reference temperature to describe the temperature-dependence of α (not implemented yet, just used for coupling analysis): θ_{ini}
Everything separated by commas.

4. Fiber growth, keyword ***FiberGrowth**. Next line, the Young's modulus, the Poisson's ratio and the growth multiplier. The direction of growth is given in the constitutive model.

5. Linear elastic, keyword ***Linear-Elastic**. Next line, the Young's modulus and the Poisson's ratio are defined. 2D plane stress/ plane strain can be defined by the keyword: *PlaneStress/ PlaneStrain*) after two material parameters.

6. Finite strain J_2 plasticity model, keyword ***J2plasticityFiniteStrain**. The model considered the bi-logarithmic elastic potential[1]. The elastic strain is defined by $\mathbf{E}^e = \log\sqrt{\mathbf{F}^{eT}\cdot\mathbf{F}^e}$ where $\mathbf{F}^e$ is the elastic deformation gradient. To approximate the logarithmic tensorial operator, the Taylor series is used instead of using the direct tensorial evaluation. Next line, the Young's modulus, the Poisson's ratio, the order of the Taylor series must be defined.The next line, a hardening law is provided. Three different laws are available:

   - The keyword LinISO for linear hardening, ie $\sigma_y = \sigma_y^0 + Hp$ where σ_y^0 is the initial yield stress, p is the equivalent plastic strain and H is a constant. To deine a finite strain J_2 plasticity model using this hardening law

     ```
     *J2plasticityFiniteStrain
     E,ν,order
     LinISO,σ_y^0,H
     ```

   - The keyword SwiftISO for Swift hardening, *i.e.* $\sigma_y = \sigma_y^0\,(1 + p/p_0)^n$ where σ_y^0 is the initial yield stress, p is the equivalent plastic strain and p_0, n are constants. To deine a finite strain J_2 plasticity model using this hardening law

[1]For the model details, refer to Nguyen, V. D., & Noels, L. (2014). Computational homogenization of cellular materials. International Journal of Solids and Structures, 51(11-12), 2183-2203 .

```
*J2plasticityFiniteStrain
E,ν,order
SwiftISO,σ_y^0,p_0,n
```

- The keyword VoceISO for Voce hardening, *i.e.* $\sigma_y = \sigma_y^0 + K\left[1 - \exp\left(Cp\right)\right]$ where σ_y^0 is the initial yield stress, p is the equivalent plastic strain and K, C are constants. To define a finite strain J_2 plasticity model using this hardening law

```
*J2plasticityFiniteStrain
E,ν,order
VoceISO,σ_y^0,K,C
```

It is noted that the *order* should be an odd number, *e.g.* a value equal to 11 should be a good choice.

7. Hyperelasticity, keyword ***HyperElastic, Type** where **Type** is directly replaced by the hyperelastic material:

   - **Neo-Hookean**, the Young's modulus and the Poisson's ratio should be defined on the next line.
   - **St-Venant-Kirchhoff**, the Young's modulus and the Poisson's ratio should be defined on the next line. Equivalent to linear elastic for small deformations. 2D plane stress/ plane strain hardcoded (must be defined in the input file with the keyword: *PlaneStress/ PlaneStrain*). 3D Isotropic material.
     *In addition, this constitutive model includes an activation criterion to activate elements which internal variable is greater than the simulation time at that moment. This option is only available using the keyword ***ACTIVATION** and defining the activation time as an internal variable using ***INITINTVARS**.
   - **St-Venant-Kirchhoff-ThermoMechanics**. This constitutive model is designed as a wrapper to couple Hyper Elastic St-Venant-Kirchhoff model with a thermomechanical couling term. The Young's modulus, the Poisson's ratio, thermal expansion and initial temperature (this model assumes that reference temperature for the thermal expansion is equal to the initial temperature) should be defined on the next line. Equivalent to linear elastic for small deformations. 2D plane stress/ plane strain hardcoded. 3D Isotropic material. This constitutive model is just used for *Thermomechanics* analysis.
     *In addition, this constitutive model includes an activation criterion to activate elements which internal variable is greater than the simulation time at that moment. This option is only available using the keyword ***ACTIVATION** and defining the activation time as an internal variable using ***INITINTVARS**.

8. Volumetric growth, keyword ***IsoMorphoGrowth**. Next line, the Young's modulus, the Poisson's ratio and the growth multiplier.

9. Viscoelastic, keyword ***ViscoElastic**. You will need 4 lines to define the model:
   Bulk modulus: K
   Number of branches in the generalised maxwell model
   Shear moduli for all the branches: μ_∞, μ_1 ...
   Viscosities for all the branches: η_1, η_2...

10. Viscoelastic Growth, keyword ***ViscoGrowth**. You will need 5 lines to define the model:
Bulk modulus: K
Number of branches in the generalised maxwell model
Shear moduli for all the branches: μ_∞, μ_1 ...
Viscosities for all the branches: η_1, η_2...
Growth multiplier of the isomorpho growth used in the coupling.

11. Viscoelastic fibre growth due to polymerisation, keyword ***ViscoMechAxialGrowth**. You will need 5 lines to define the model:
Bulk modulus: K
Number of branches in the generalised maxwell model
Shear moduli for all the branches: μ_∞, μ_1 ...
Viscosities for all the branches: η_1, η_2...
The initial growth multiplier, the polymerisation rate, the depolymerisation rate, the density of cytoskeletal filaments, and finally the whole vector of the direction of growth, everything separated by commas.

12. Viscoelastic fibre growth due to polymerisation with contractility, keyword ***ViscoMechContractAxialGrowth**. You will need 5 lines to define the model:
Bulk modulus: K
Number of branches in the generalised maxwell model
Shear moduli for all the branches: μ_∞, μ_1 ...
Viscosities for all the branches: η_1, η_2...
The initial growth multiplier, the polymerisation rate, the depolymerisation rate, the density of cytoskeletal filaments, the contractility rate, the stress threshold for contractility, and finally the whole vector of the direction of growth, everything separated by commas.

13. Volumetric growth as a consequence of variations in the deformation gradient (mechanical growth). There are two type of functions: the virtual ones inherited from the general constitutive law (they MUST be implemented by the user), and the particular ones that the user wants to implement for their convenience. Keyword ***VolMechDrivenGrowth**, Not yet implemented in *InputFile.cpp*.

14. Stochastic Kirchhoff-St-Venant model. In addition to the Hyperelastic St-Venant-Kirchhoff constitutive model, the Young's modulus and Poisson's ratio can be random variables. For the moment, they only exhibit a Gaussian distribution and are space-independent. To activate the stochastic analysis, the keyword ***Stochastic** must be specified before ***HyperElastic**. The order of the expansion needs to be specified after the keyword ***Order**. Then the randomness of variables are specified using this convention: the first number relates to which variable is supposed to be random (1 for Young's modulus and 2 for Poisson's ratio). Next the number of functions used to described this random variable. Finally, the ratio of the standard variation to the mean value is indicated for each of these functions. We take the following file as an example:

```
*Stochastic
*HyperElastic, St-Venant-Kirchhoff
1E7, 0.3
*ORDER
3
1, 1, 0.1
```

```
2, 1, 0.1
*POLYNOMIAL
CHAOS
```

In that case the Young's modulus is a Gaussian variable of mean 1E7 and standard deviation of 1E7 times 0.1. The Poisson's modulus is a Gaussian variable of mean 0.3 and standard deviation of 0.3times 0.1. The keyword **CHAOS** can be replaced by **Haar** if one needs a discontinuous polynomial expansion.

15. Thermomechanical model for 3D printing simulation, keyword ***3DPrinting-St-Venant-Kirchhoff-Thermoelasticity**. This constitutive model is St-Venant-Kirchhoff model with a thermomechanical coupling term and temperature-dependent properties. The model has been developed to capture the physics of a selective laser melting process. This 3D printing technology starts with powder as a starting material. A laser melts the powder along a predefined path, changing the material from powder to liquid, and to solid when the laser is far enough away. Thermal strains are neglected for powder and liquid phase. Finally, under the assumption of an open system, new powder is always filling the molten pool. This model considers a new reference configuration when the material turns to liquid, being an element in a stress-free configuration the first time that this turns from powder to liquid, at the same time that a change from ρ_{powder} to ρ_{liquid} is produced (see fig. 2.8 and for more information subsection 6.3.8).

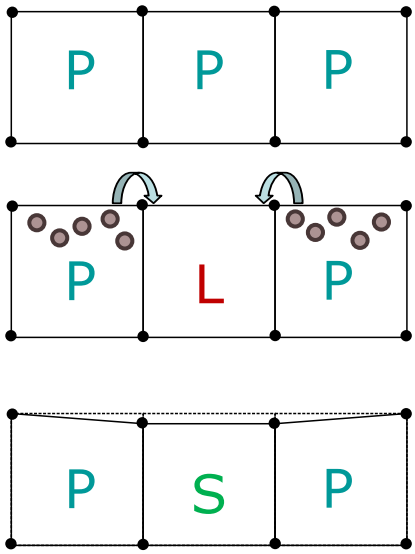


Figure 2.8: Description of the different combination of phases

You will need only one line to define the model (Everything separated by commas): $\rho_{newConf}$, $\rho_{transition}$, $E_{P_{ref}}$, E_{PL}, $E_{S_{ref}}$, E_{SL_1}, E_{SL_2}, $\upsilon_{S_{ref}}$, υ_{SL}, κ_{SL}, $\theta_{sigMidpoint}$, α_{solid}, θ_{ref}, θ_{SL} θ_{liq} (being P: powder, S: solid, PL: transition powder-liquid, SL: transition solid-liquid, *sigMidpoint*: value of the sigmoid's function midpoint) where (see Figure 2.9):

$$E(GPa) = \begin{cases} P: & E_{P_{ref}} + \frac{E_{PL}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} \\ S-L: & E_{S_{ref}} + \frac{E_{SL_1}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} + E_{SL_2}(\theta-\theta_{SL}) \quad if \quad \theta \leq \theta_{SL} \\ & E_{S_{ref}} + \frac{E_{S_1}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} \quad if \quad \theta > \theta_{SL} \end{cases}$$

$$\upsilon(-) = \upsilon_{S_{ref}} + \frac{\upsilon_{SL}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))}$$

$$\alpha(1/K) = \frac{\alpha_S}{1+\exp(\kappa_{SL}(\theta-\theta_{sigMidpoint}))}$$

This constitutive model is just used for *Thermomechanics* analysis.
The material phase (Powder = 0, Liquid = 1 and Solid= 2), is modelled through a internal variable (1^{st} =activation time, 2^{nd} =activation state, 3^{rd} =material phase, $4^{th} = E(\theta)$, $5^{th} = \upsilon(\theta)$, 6^{th} = thermal stretch ratio) as:

$$intVarsCurr[2] = \begin{cases} 0 & if \quad intVarsPrev[2] = 0 \ \& \ T \leq \theta_{Liquid} \\ 1 & if \quad intVarsPrev[2] \geq 0 \ or \ T > \theta_{Liquid} \\ 2 & if \quad intVarsPrev[2] \geq 1 \ or \ T \leq \theta_{Liquid} \end{cases}$$

*In addition, this constitutive model includes an activation criterion to activate elements which internal variable is greater than the simulation time at that moment. This option is only available using the keyword ***ACTIVATION** and defining the activation time as an internal variable using ***INITINTVARS**.

16. Temperature model for 3D printing simulation, keyword ***3DprintingTemperature**. This constitutive model has temperature-dependent properties. To be used in combination with the mechanical coupling is necessary to use the keyword ***3DprintingTemperatureCoupling** (to be used in combination with ***3DPrinting-St-Venant-Kirchhoff-Thermoelasticity**). You will need only one line to define the model (Everything separated by commas): $C_{S_{ref}}$, C_{SL_1}, C_{SL_2}, $L_{\_PL1}$, $L_{\_PL2}$, $L_{\_PL3}$, $L_{\_SL1}$, $L_{\_SL2}$, $L_{\_SL3}$, $k_{P_{ref}}$, k_{PL_1}, $k_{S_{ref}}$, k_{SL_1}, k_{SL_2}, k_{SL_3}, κ_{SL}, $\theta_{SigMidpoint}$, θ_{ref}, θ_{powder} θ_{SL}, θ_{liq} (being P: powder, S: solid, *PL*: transition powder-liquid, *SL*: transition solid-liquid, *sigMidpoint*: value of the sigmoid's function midpoint) where (see Figure 2.9):

$$C(J/Kkg) = \begin{cases} P: & C_{S_{ref}} + \frac{C_{S_1}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} & if & \theta \leq \theta_{ref} \\ & C_{S_{ref}} + \frac{C_{S_1}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} + C_{S_2}(\theta-\theta_{ref}) & if & \theta_{ref} \geq \theta \leq \theta_{powder} \\ & -L_{PL_1}\theta^2 + L_{PL_2}\theta - L_{PL_3} & if & \theta_{powder} > \theta < \theta_{liq} \\ & C_{S_{ref}} + \frac{C_{S_1}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} & if & \theta \geq \theta_{liq} \\ S-L: & C_{S_{ref}} + \frac{C_{S_1}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} & if & \theta \leq \theta_{ref} \\ & C_{S_{ref}} + \frac{C_{S_1}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} + C_{S_2}(\theta-\theta_{ref}) & if & \theta_{ref} \geq \theta \leq \theta_{SL} \\ & -L_{SL_1}\theta^2 + L_{SL_2}\theta - L_{SL_3} & if & \theta_{SL} > \theta < \theta_{liq} \\ & C_{S_{ref}} + \frac{C_{S_1}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} & if & \theta \geq \theta_{liq} \end{cases}$$

$$k(W/mK) = \begin{cases} P: k_{P_{ref}} + \frac{k_{P_1}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} & \\ & \\ \qquad k_{S_{ref}} + \frac{k_{S_1}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} & if \quad \theta \leq \theta_{ref} \\ S-L: \quad k_{S_{ref}} + \frac{k_{S_1}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} + k_{S_2}(\theta-\theta_{ref}) & if \quad \theta_{ref} \geq \theta \leq \theta_{liq} \\ \qquad k_{S_3} + \frac{k_{S_2}}{1+\exp(-\kappa_{SL}(\theta-\theta_{sigMidpoint}))} & if \quad \theta \geq \theta_{liq} \end{cases}$$

The value of some material properties, temperature-dependent, can be analysed through the internal variable ($intVar[\_intVarsSizeTemp]$ =conductivity, $intVar[\_intVarsSizeTemp + 1]$ =specific heat, $intVar[\_intVarsSizeTemp + 2]$ = density).

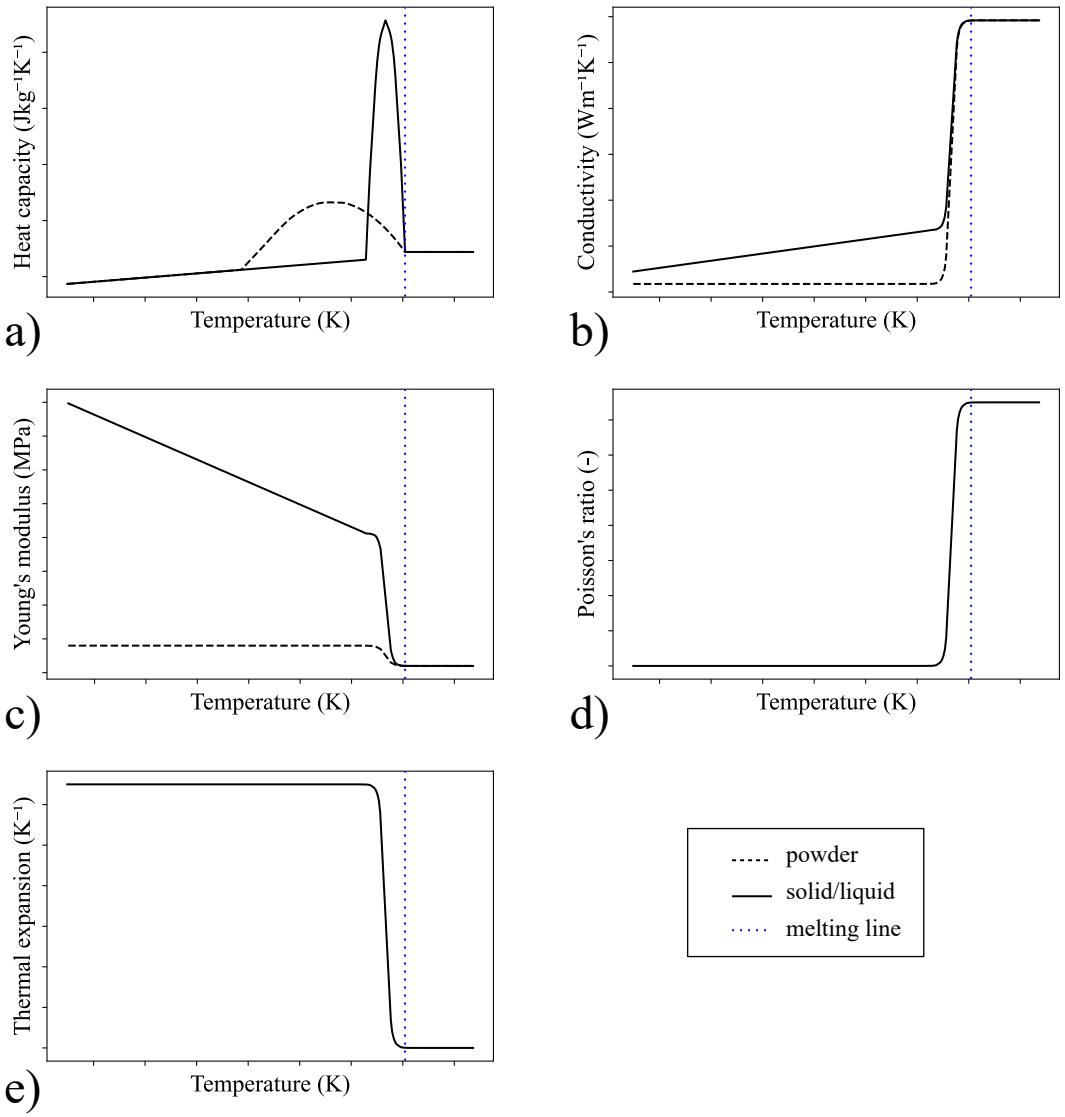


Figure 2.9: 3D printing constitutive models: a) and b) material properties of 3DPrintingTemperature model; c), d) and e) material properties of 3DPrinting-St-Venant-Kirchhoff-Thermoelasticity model

The list of elements for the specific material should be provided with the keyword ***List of Elements**, with one label of the element per line. You can also define a bounding box with the keyword ***Bounding box** and then list the corner nodes of the box. All of the nodes inside the box will be selected. **THIS FEATURE IS NOT YET IMPLEMENTED**.

Example:

```
*FLUX EXTRADOF, TYPE=CURRENT INST
0, 5, 0.3
1
2
*FLUX EXTRADOF, TYPE=CURRENT INST
100, 110, 0.2
40
41
42
43
```

Finally, to define the initial value of a specific internal variable, we must provide the element in whose GPs we want to define an internal variable followed by the position and the value of the internal variable. All elements defined here must be contained in the ***** **List of Elements** previously defined for the material. Example:

```
*INITINTVARS
1, 1, 0
2, 1, 0.2
3, 1, 0.4
4, 1, 0.6
```

Please note that all input files must be ended with the keyword ***END** at the end of the file. This is the list of input files provided with the program to illustrate a typical input file for *MuPhiSim*

2.4.9 Data extraction

The numerical results often need to be extracted, e.g. displacement at a point, reaction force at a boundary, total elastic energy in the problem, maximal stress in the model, *etc.* This code provides different possibilities for data extraction. The extracted data is written in the current folder under the **.csv** files specifying a value of a quantity over time, e.g.

```
Time;Value
...
...
```

The data at nodes can be extracted using ***EXTRACTION, NODE**. This block is defined as follows

```
*EXTRACTION, NODE
prefix,quantity,Operation,comp
n1
n2
...
```

```
nN
```

In this block, one has

- **prefix**: a string is defined by user and used as the first part of the file name.
- **quantity**: a quantity name must be chosen between **Unknown** for unknown field, **InternalForce** for internal force or **ExternalForce** for external force.
- **Operation**: an operation must be chosen between **Mean**, **Max**, **Min**, **Sum**, and **Rough**. When **Rough** is used, the output file includes all values at all nodes, where each value of each node is provided by a column.
- **comp**: the component to be extracted.
- **n1, n2, ..., nN**: list of nodes

Some useful cases:

- Extract the displacement following y of node 100

```
*EXTRACTION, NODE
node10Disp,Unknown,Mean,1
10
```

- Extract the total internal force following y on the boundary whose nodes are 10, 20, 30:

```
*EXTRACTION, NODE
node10Disp,InternalForce,Sum,1
10
20
30
```

The data at GPs can be extracted using ***EXTRACTION, ELEMENT**. This block is defined as follows

```
*EXTRACTION, ELEMENT
prefix,quantity,Operation
e1
...
eN
```

In this block, one has

- **prefix**: a string is defined by user and used as the first part of the file name.
- **quantity**: a quantity name must be specify. Currently, deformation gradient components (**FXX, ..., FZZ**), stress components (**PXX, ..., PZZ**), **DEFO_ENERGY, EXTERNAL_ENERGY, KIN_ENERGY** can be used.
- **Operation**: an operation must be chosen between **Mean**, **Max**, **Min**, and **Sum**.
- **e1, ..., eN**: list of elements

Some useful cases:

- Extract the total energy of the model

```
*EXTRACTION, ELEMENT
totalEnergy,DEFO_ENERGY,Sum
All
```

- Extract the average of **FXX**

```
*EXTRACTION, ELEMENT
totalEnergy,FXX,Mean
All
```

- Extract maximum value of PYY over a groups of elements: 10, 20, 30:

```
*EXTRACTION, NODE
maxPyy,PYY,Max
10
20
30
```

CHAPTER 3

EXAMPLES DOCUMENTATION

3.1 Test cases

The following chapter aims at illustrating the contents of 16 examples of an input file for *MuPhiSim*. For each example, the structure of elements, the solvers and the material are described through several figures where boundary conditions and a few values of stress or strain are shown for the purpose of a better understanding of the problem. There is a summary at the end of this document which presents the main data of each example. These 5 examples are built to study the following problems :

1. `example1.inp` Twisting of a beam made of FEM and MM elements
2. `example2.inp` Twisting of a beam made of a Visco-Growth model
3. `example3.inp` Vibration of a beam pulled up longitudinally and then transversely
4. `example4.inp` Stretching of a cell made of viscoMechContractAxialGrowth and viscoMechAxialGrowth models
5. `example5.inp` Flexion of an elastic bunny
6. `example6.inp` Electrophysiological model
7. `example7.inp` Artificial viscosity applied on the displacement of a piston with explicit solver
8. `example8.inp` Oscillation of a beam with an explicit solver after the application of a static pull
9. `example9.inp` Steady-state conduction heat transfer of a beam applying surface heat flux
10. `example10-11.inp` Transient conduction heat transfer cases
11. `example12.inp` Axial displacement of a beam due to heating
12. `example13.inp` Stochastic axial tip displacement of a 2D beam with random constitutive parameters

13. `example14.inp` Heat transfer analysis with progressively element activation
14. `example15.inp` 3D block with MM nodes and partitioned into 4 regions to be assigned to different processors
15. `example16.inp` Transient heat problem with convection BC on a 2D plate

3.1.1 Example 1 : `example1.inp`

This input file presents a 3D beam made of both FEM and MM nodes/elements. The model used is HyperElastic, St-Venant-Kirchhoff. During the first second, a displacement of the face $z = 0.6$ is computed by an Implicit Static solver. Then, at t = 1s, this face is released and the oscillations computed by an Implicit Dynamic solver for a two second period. In the case of parallel processing, the MM nodes/elements are assigned to one processor and the FEM nodes/elements are partitioned by METIS and allocated to the rest of the processors.

Figure 3.1 shows the initial configuration and the Dirichlet Boundary Conditions for the first solver. Figure 3.2 shows the beam at its maximum strain.

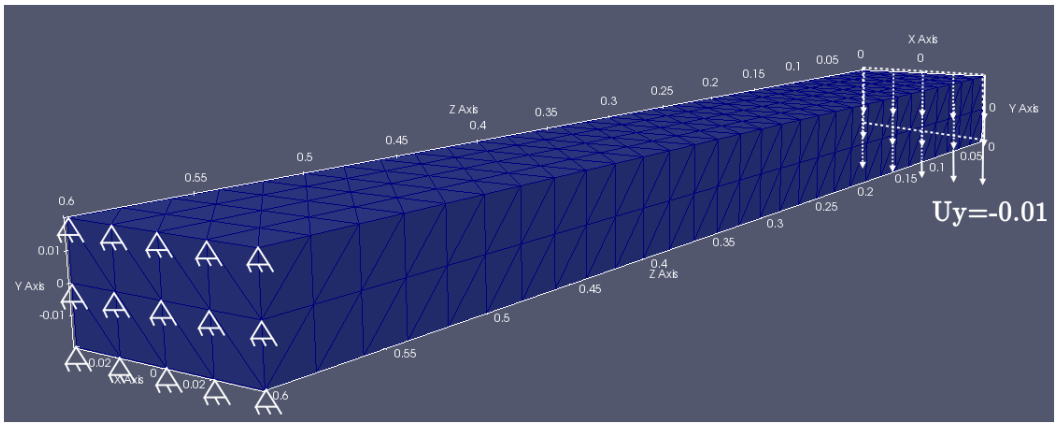


Figure 3.1: Initial configuration of the beam, example 1

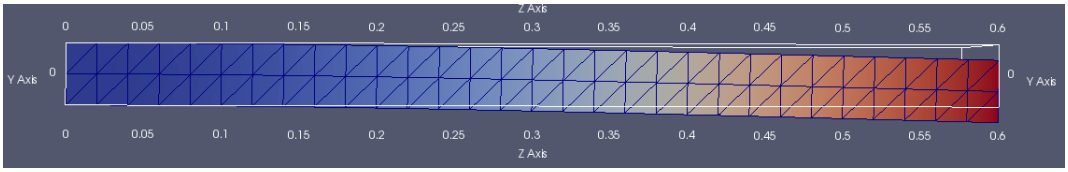


Figure 3.2: Maximum deformation of the beam, example 1

3.1.2 Example 2 : `example2.inp`

This input file presents a 3D beam made of FEM nodes and elements. The model used is ViscoGrowth. From t = 0 to t = 1s, a displacement of the face $z = 600$ is computed by an Explicit solver.

Figure 3.3 shows the initial configuration and the Dirichlet Boundary Conditions. Figure 3.4 shows the beam at the end of the study.

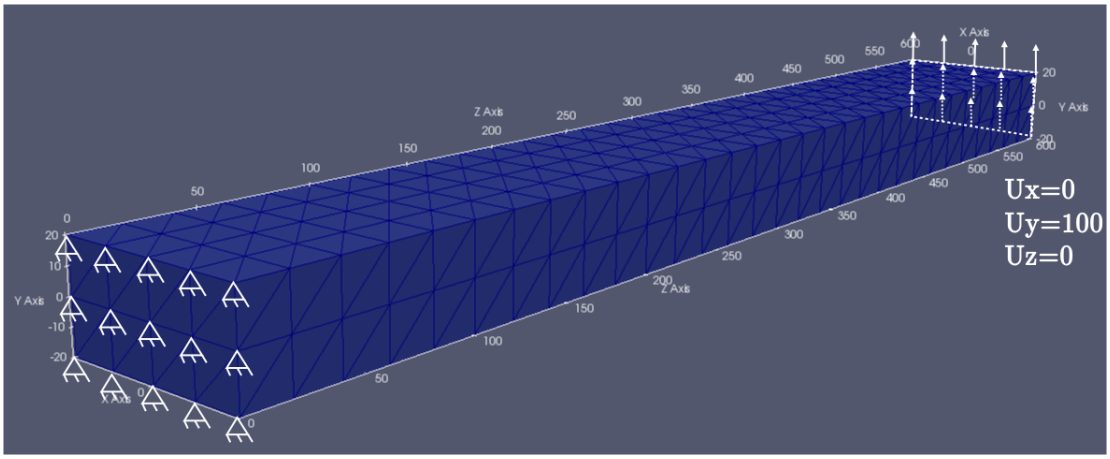
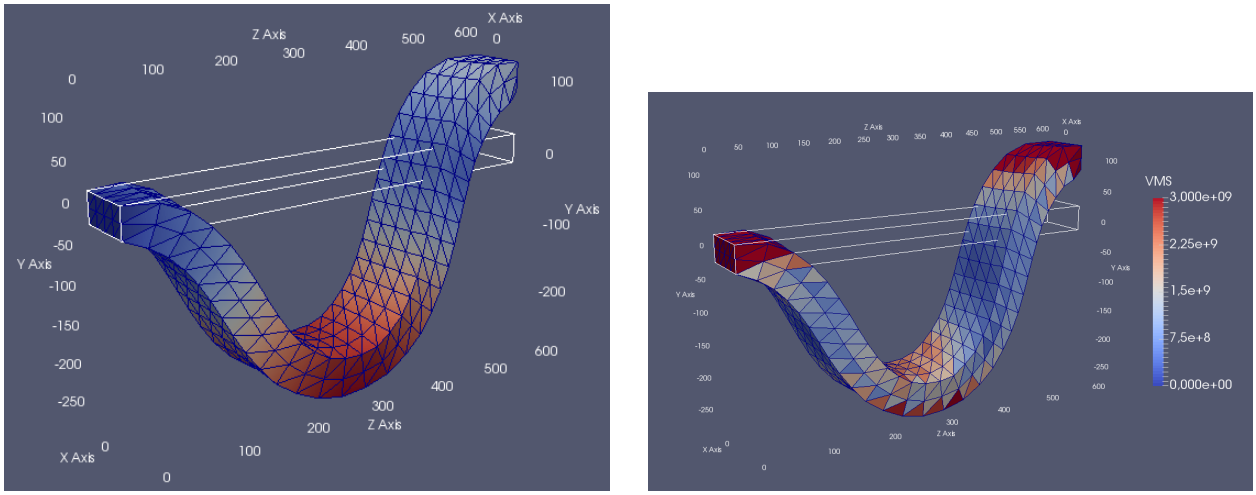


Figure 3.3: Initial configuration of the beam, example 2

Figure 3.4 show the strain and Von Mises stress on distorted configuration of the beam.



(a) Strain (b) Von Mises Stress

Figure 3.4: Distorted configuration of example 2

3.1.3 Example 3 : `example3.inp`

This input file presents a 2D beam made of FEM nodes and elements. The model used is HyperElastic, St-Venant-Kirchhoff. There are 4 solvers implemented here :

1. From 0 to 1s : Implicit Static, the right side of the beam is pulled up.
2. From 1 to 10s : Implicit Dynamic, the right side of the beam is released, the beam oscillates vertically.
3. From 10 to 14s : Implicit Static, the right side of the beam is pulled to the right.
4. From 14 to 20s : Implicit Dynamic, the right side of the beam is released, the beam oscillates horizontally.

It has to be noted that the Dirichlet Boundary Conditions are incremental, therefore the beam is pulled to the right at $t = 10s$ from a distorted configuration and not the initial.
Figures 3.5 to 3.8 show the distorted configuration of the beam before each solver.

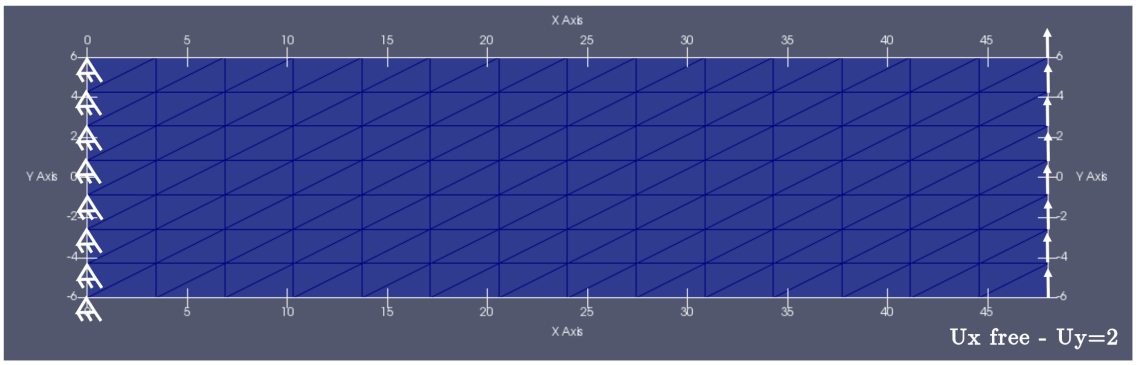


Figure 3.5: Initial configuration of the beam - $t = 0s$

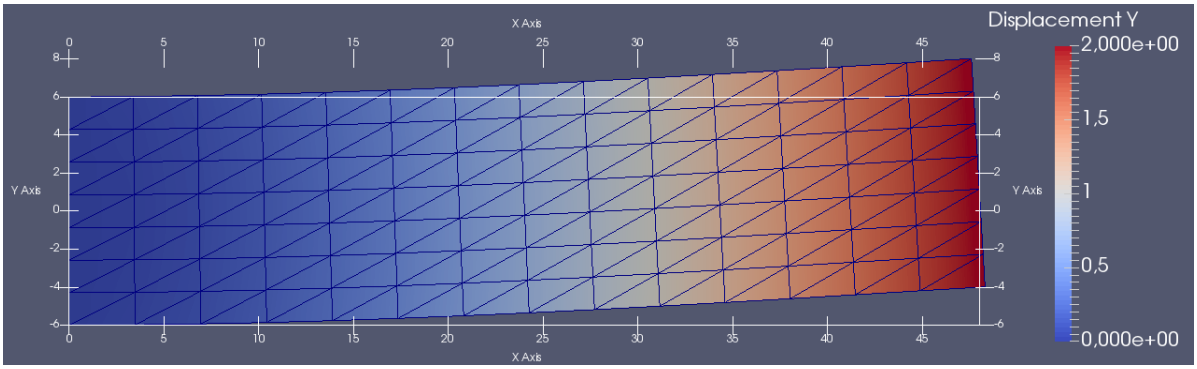


Figure 3.6: Distorted configuration of the beam - U_Y field - $t = 1s$

The nodal forces at the left side of the beam are saved in a .csv file. We are only registering the forces among x direction during the first and the third solvers each 0.01s.

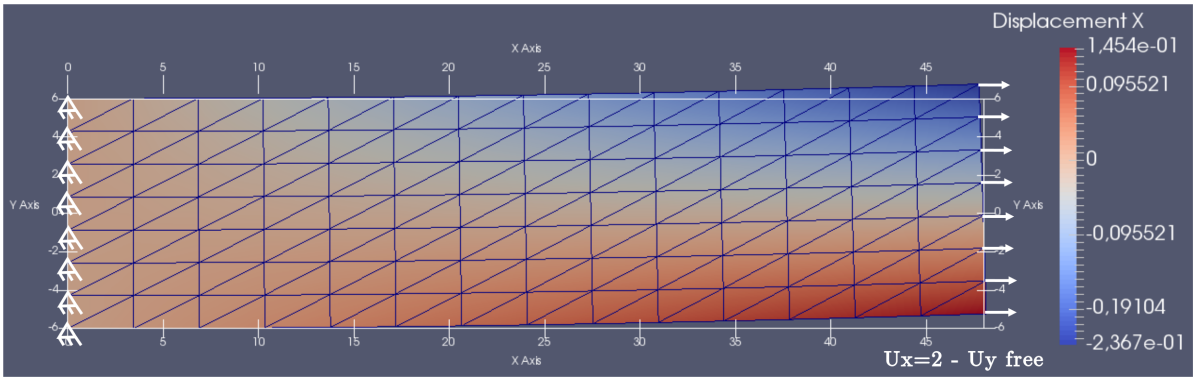


Figure 3.7: Distorted configuration of the beam - U_X field - $t = 10s$

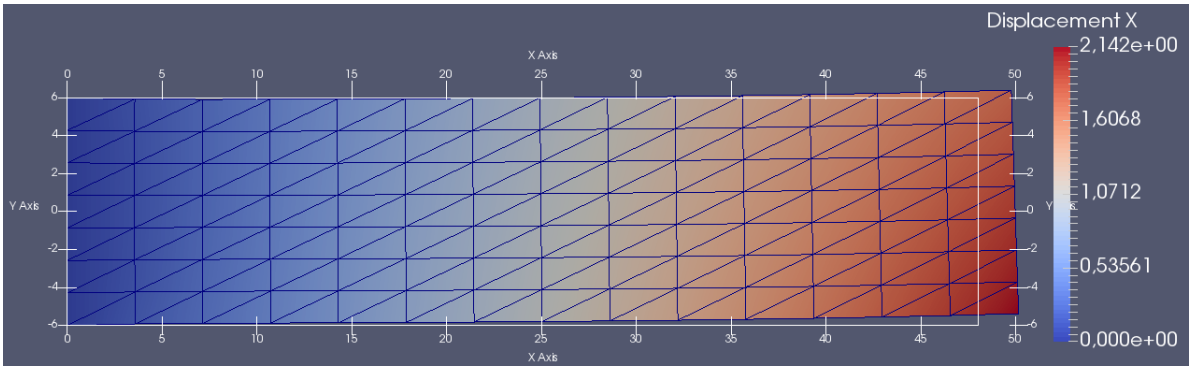


Figure 3.8: Distorted configuration of the beam - U_X field - $t = 14s$

3.1.4 Example 4 : `example4.inp`

This input file presents a 2D beam made of FEM nodes and elements. There are two different regions named "membraneCortex" and "axonShaft" which are made of a ViscoMechContractAxialGrowth model (outline of the beam) and a ViscoMechAxialGrowth model (the centre of the beam) respectivelly. The solver used is Implicit Static between 0 and 2500s.

During the computation, pressure is progressively applied on the right side of the beam as presented on Figure 3.9.

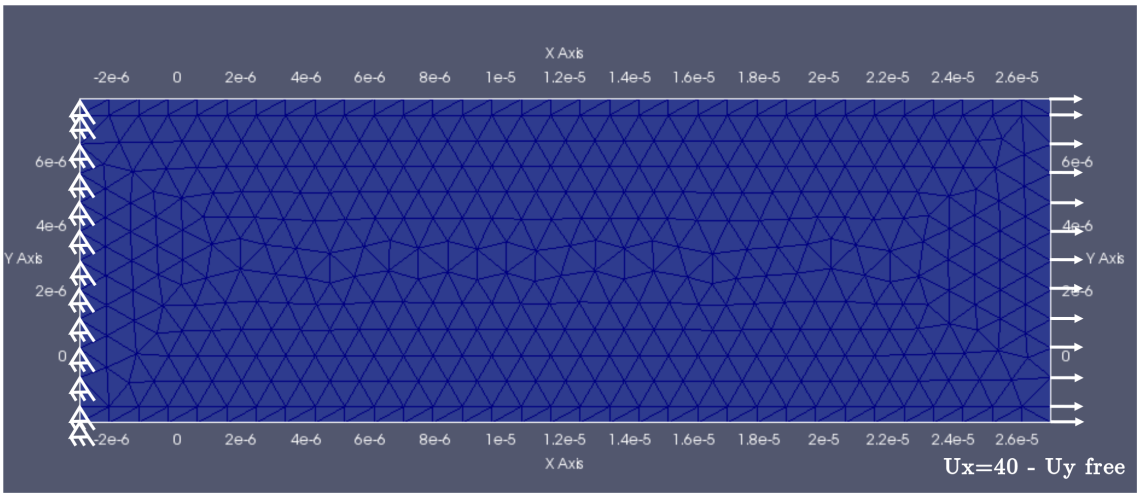


Figure 3.9: Initial configuration of the beam

Figures 3.10 to 3.12 show the distorted configuration of the beam at the end of the computation with U_X, U_Y and σ_{VMS} field.

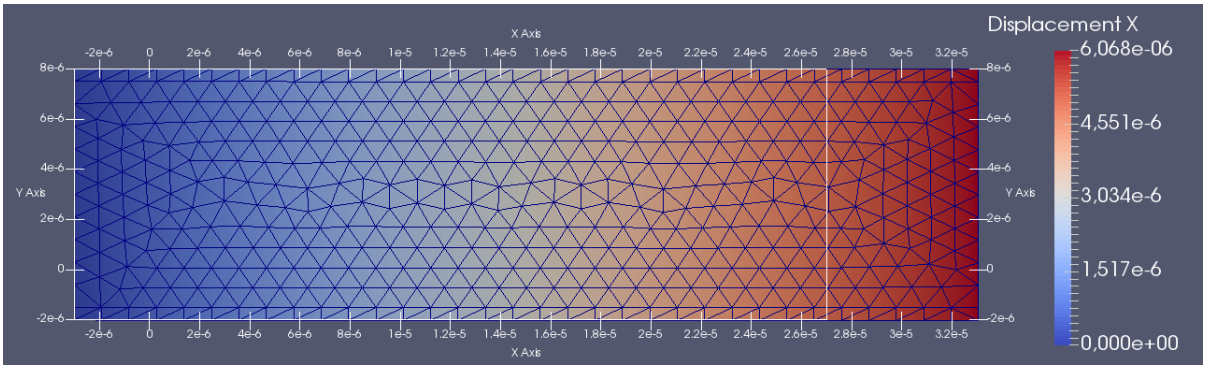


Figure 3.10: Distorted configuration of the beam - U_X field

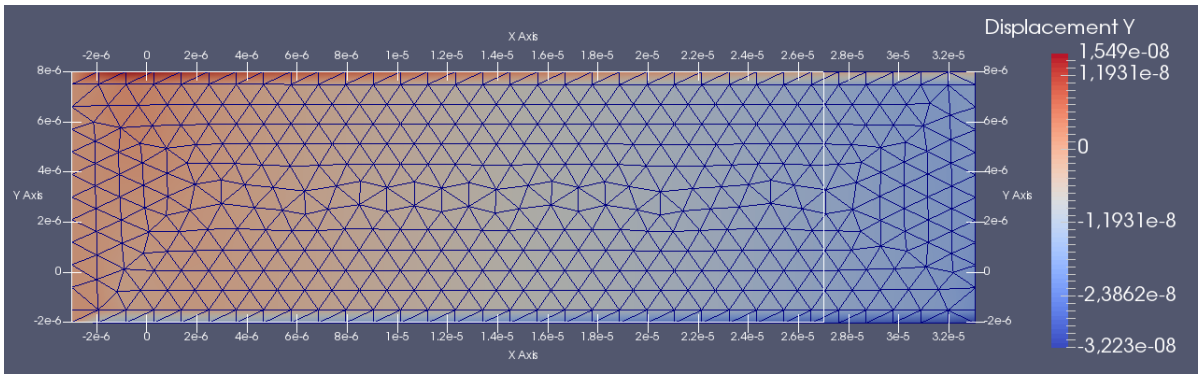


Figure 3.11: Distorted configuration of the beam - U_Y field

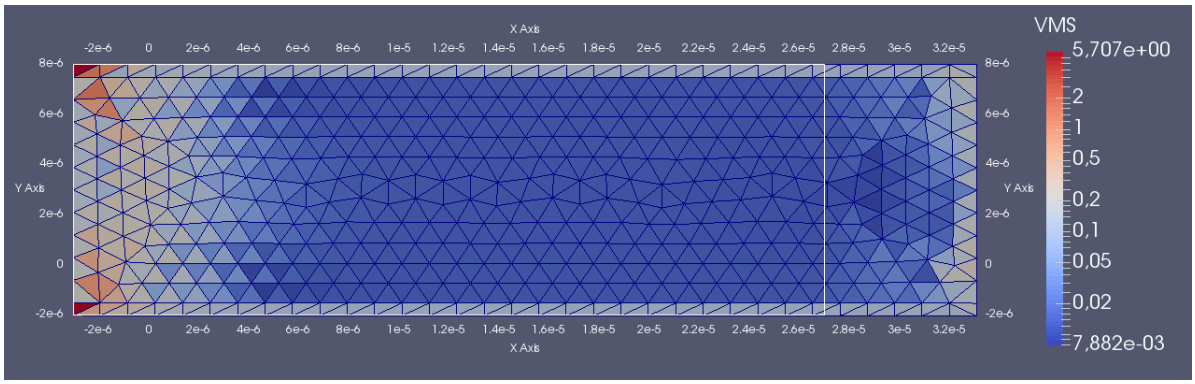


Figure 3.12: Distorted configuration of the beam - σ_{VMS} field

3.1.5 Example 5 : `example5.inp`

This input file presents a 2D bunny made of FEM nodes and elements. The model used is HyperElastic, St-Venant-Kirchhoff. From t=0-5s, a displacement of the nodes presented on Figure 3.13 is computed by an Implicit Static solver.

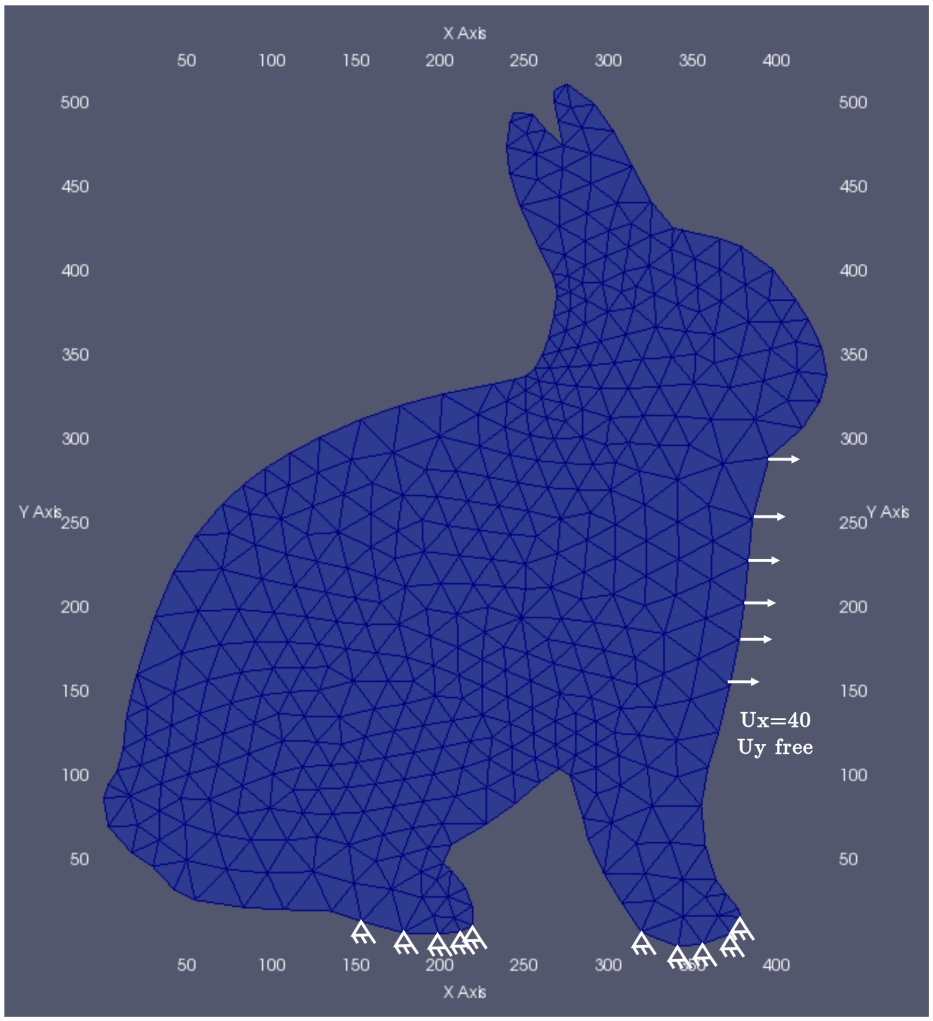


Figure 3.13: Initial configuration of the bunny

Figures 3.14 and 3.15 show the distorted configuration of the bunny at the end of the computation with $||\vec{u}||_2$ and σ_{VMS} field.

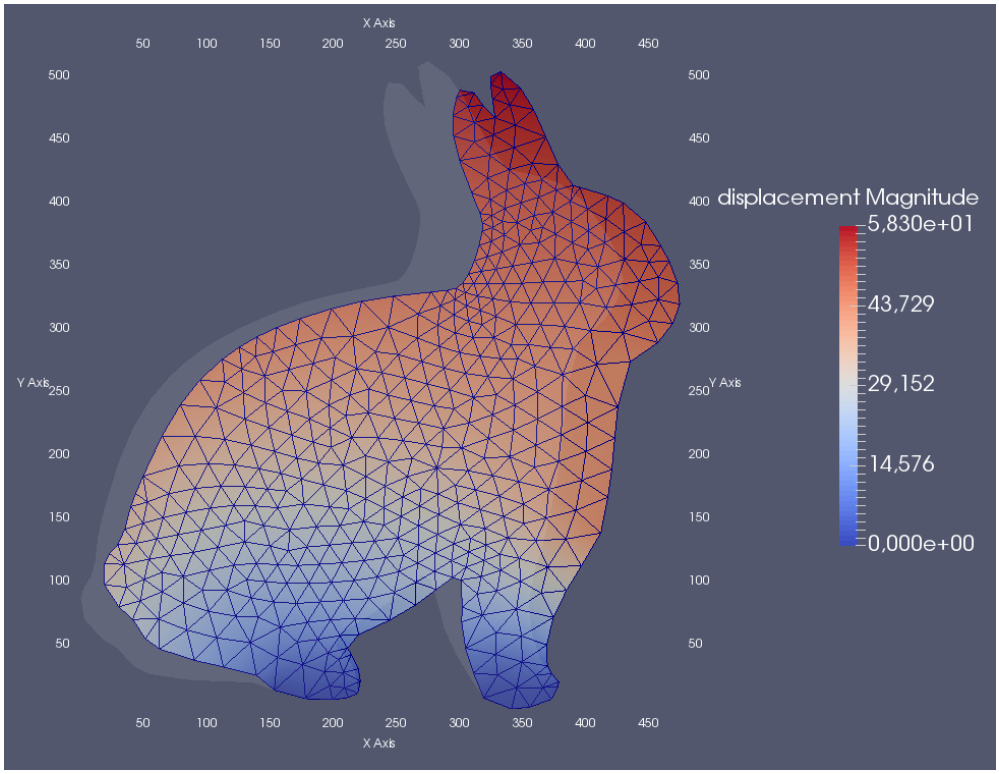


Figure 3.14: Distorted configuration of the bunny - $\|\overrightarrow{u}\|_2$ field

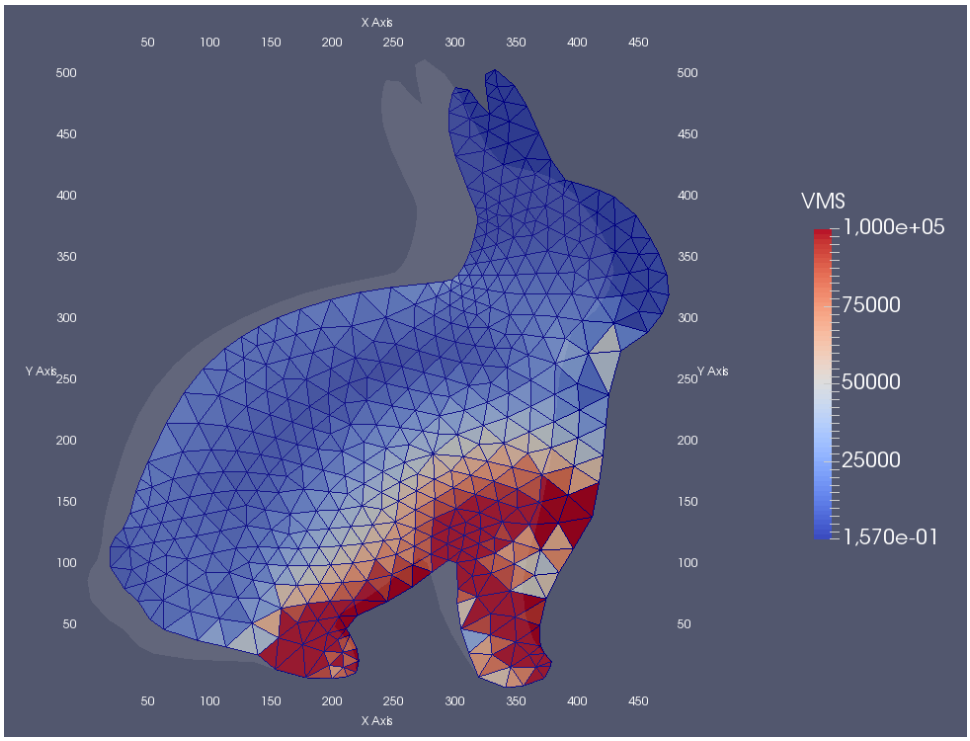


Figure 3.15: Distorted configuration of the bunny - σ_{VMS} field.

3.1.6 Example 6-6bis : `example6.inp and example6bis.inp`

This input file presents a 2D beam made of FEM nodes and elements. The model used is HyperElastic, St-Venant-Kirchhoff (ViscoMechAxialGrowth for 6bis), but it doesn't matter because there will not be any deformation. The extra dof model used is electrophysical, FitzHugh-Nagumo (FHN), the third dimension used is a voltage Φ. From t = 0-1s, a volumetric Neumann Current $I_N = 0.8$ is added to the elements on the left side of the beam. The propagation of the voltage wave through the beam is computed by an Implicit Static solver from 0s to 200s (see Figure 3.16).

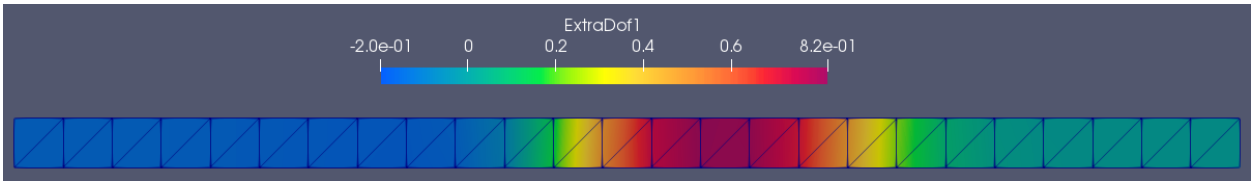


Figure 3.16: Voltage field at t=100

Figures 3.17 and 3.18 show the spatial evolution of the voltage and the ionic current at t=100 and the temporal evolution at the mid-node.

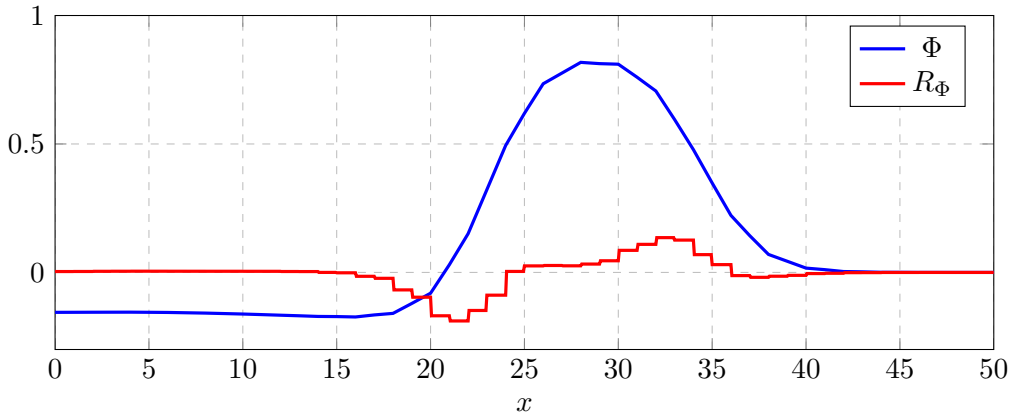


Figure 3.17: Voltage and Ionic current field at t=100

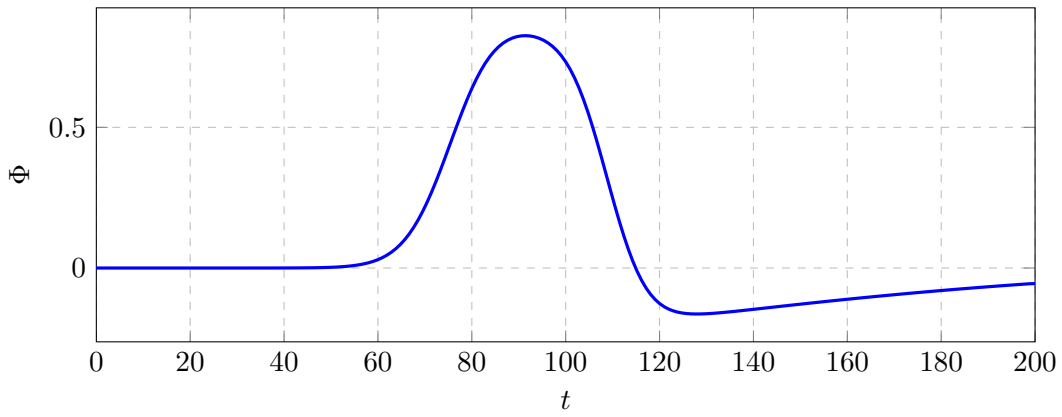


Figure 3.18: Voltage fields at x=25

3.1.7 Example 7 : `example7.inp`

This input file presents a 2D beam made of FEM nodes and elements. The model used is HyperElastic, St-Venant-Kirchhoff. The solvers are the following:

- From 0 to 2.5s, classic explicit solver.
- From 2.5 to 5s, explicit solver with an artificial bulk viscosity, coefficients [2, 3].

The left side of the beam is blocked and instantaneous pressure is applied on the right side of the beam:

- From 0 to 1.25: $P = -5 \cdot 10^5$
- From 2.5 to 3.75: $P = -5 \cdot 10^5$

Figure 3.19 shows the temporal evolution of U_x on the right side of the beam without and with artificial viscosity.

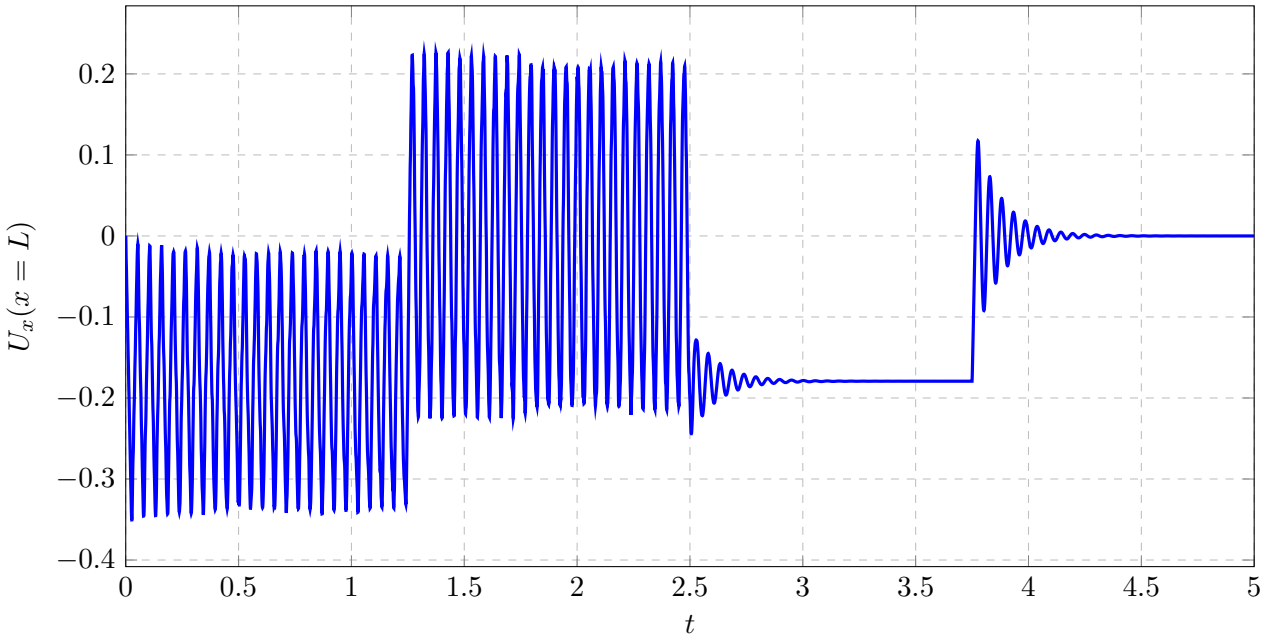


Figure 3.19: Temporal evolution of $U_x(x = L)$

3.1.8 Example 8 : `example8.inp`

This input file contains a 2-D beam made up of quadratic elements, and also exemplifies the output of forces. The input file contains two solvers, the first one is implicit static to pull the beam up, and the latter solver is an explicit solver to see the dynamic response of the beam. The forces written are for the second solver and can be used to measure the oscillation period of the beam. The beam is fixed at x=0 for the whole duration of the simulation.

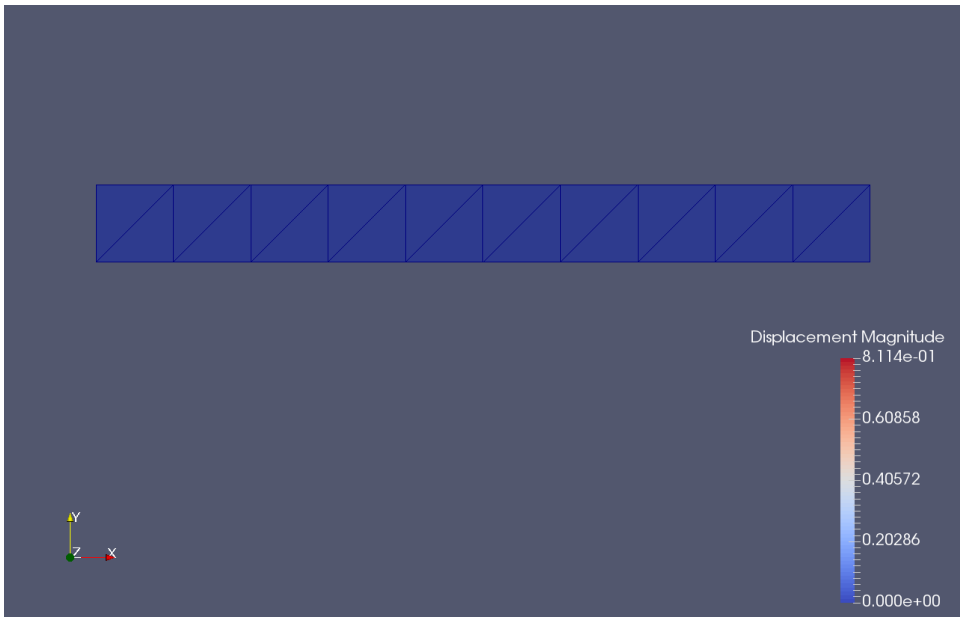


Figure 3.20: Initial configuration of the beam

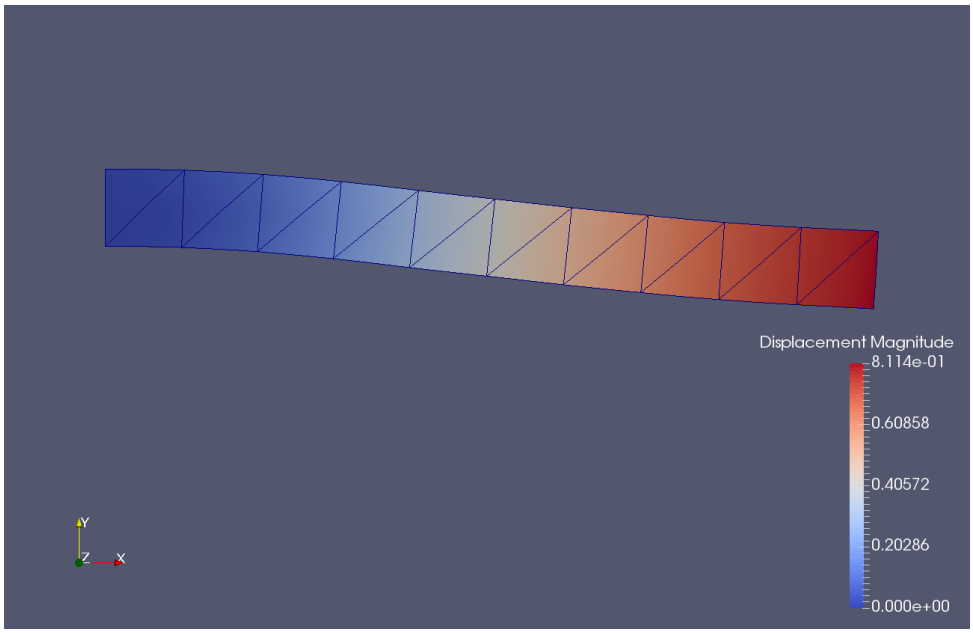


Figure 3.21: Distorted configuration of the beam - $\|\vec{u}\|_2$ field

Figures 3.20 and 3.21 show the initial and the distorted configuration of the beam with $||\vec{u}||_2$ field.

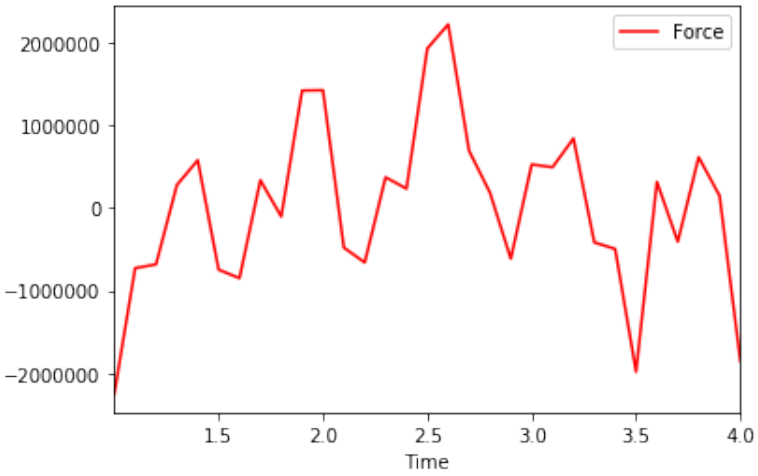


Figure 3.22: Forces on dirichlet node

Table 3.1: Forces .csv

Node	Direction	Force	Time
1	1	-2255059.486727	1.000029
1	1	-728291.758768	1.100021
1	1	-682609.980855	1.200018

Also, the forces can be found in the .csv file in the output folder. The graph of force vs time for the fixed node can be found on Figure 3.22, and a few rows of the .csv file on Table 3.1.

3.1.9 Example 9: `example9.inp`

This input file presents a 2D steady-state conduction heat transfer case made of FEM nodes and elements (same geometry as example6). The model used is HyperElastic, St-Venant-Kirchhoff, but it doesn't matter because there will not be any deformation. The extra dof model used is Temperature, the third dimension used is temperature θ. A prescribed surface heat flux, $\bar{q} = 802\ W/m^2$ is applied on the right side while prescribed temperature is defined on the two nodes on the left (see Figure 3.23). In addition, a prescribed volumetric heat flux $r = 100\ W/m^3$ is defined on the beam.

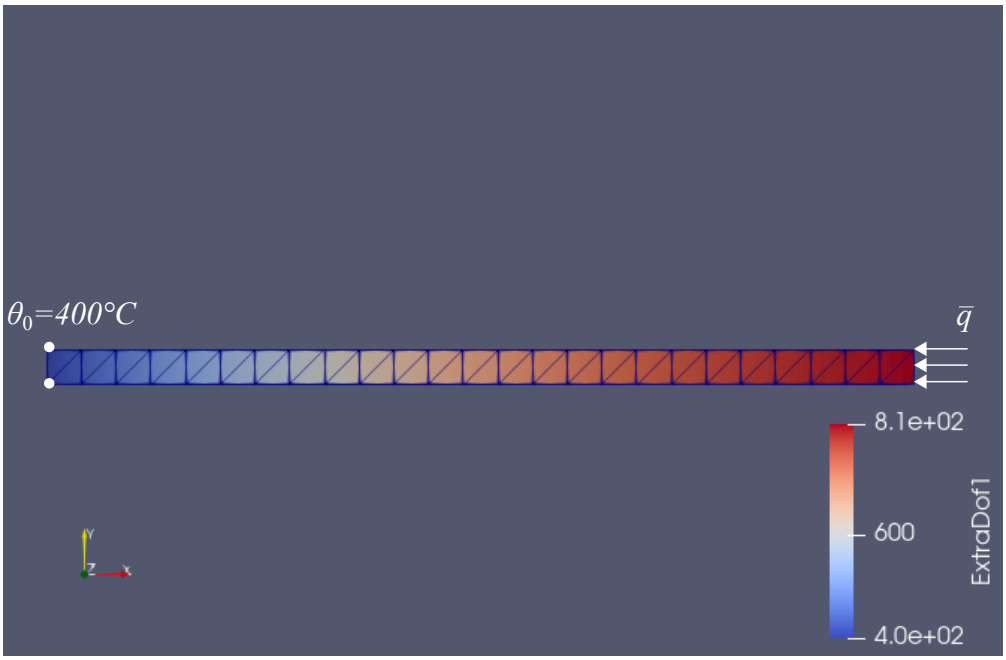


Figure 3.23: Temperature distribution of the beam

3.1.10 Example 10-11 : `example10-11.inp`

These input files present two 2D transient conduction heat transfer cases made of FEM nodes and elements. Example 10 presents a heat conduction on a semi-infinite body ($x > 0$) (the geometry used in this case beam 0.5x30m) with an initial temperature of $\theta = \theta_i = 0°C$ in all the domains. The temperature on the left side is defined as $\theta = \theta_0$, while the temperature far from this side is not affected and remains at the initial temperature $\theta = \theta_i$, (see Figure 3.24). Example 11 presents a nonhomogeneous problem, considering a finite slab with a thickness of $L = 2$m (the geometry used in this case square 2x2m), and where the boundary conditions are $\theta = \theta_0$ for $x = 0$ ($t > 0$), $\theta = \theta_i = 0°C$ for $x = L$ ($t > 0$) and $\theta = \theta_i = 0°C$ for $0 < x < L$ ($t = 0$), (see Figure 3.25). The model used is HyperElastic, St-Venant-Kirchhoff, but it doesn't matter because there will not be any deformation. The extra dof model used Temperature, the third dimension used is temperature θ. In both cases $r = 0$. The evolution of temperature is computed by an Implicit Static solver from 0s to 1000s. In order to solve the problem, two consecutive solvers are used:

1. From 0 to 1s : Implicit Static, the corresponding temperature is defined on the extremities of the bar.

2. From 1 to 1000s : Implicit Static, the temperature on the extremities of the bar is defined as zero (keeping, therefore, the temperature defined in the first solver).

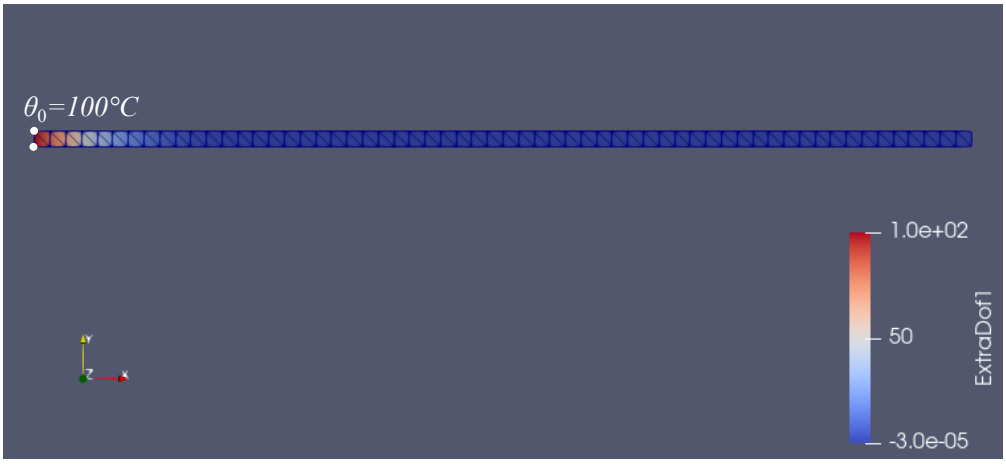


Figure 3.24: Temperature distribution at t=1000s

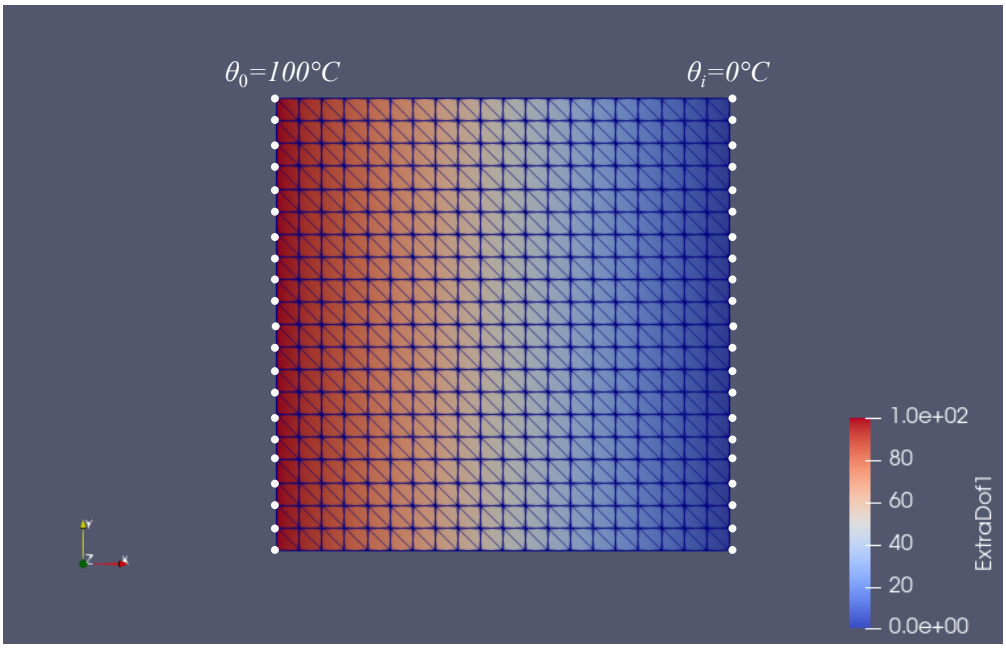


Figure 3.25: Temperature distribution at t=1000s

Figures 3.26 and 3.27 show a comparison of the evolution of temperature over time at different x for both cases described above, obtained analytically, using Abaqus (ABQ) and finally MuPhiSim (OX).

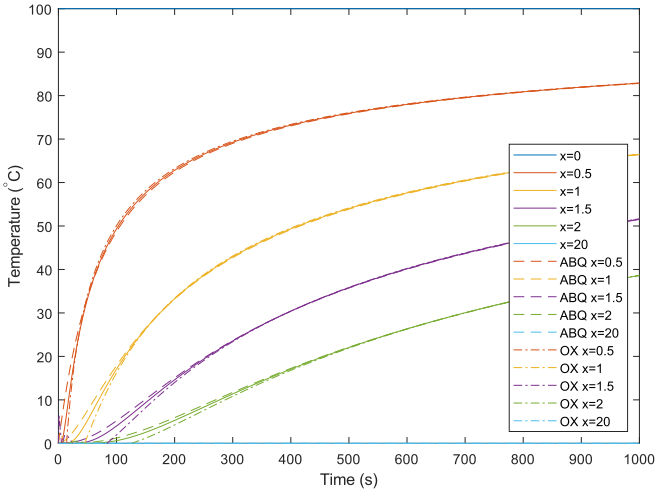


Figure 3.26: Comparison between solutions obtained analytically, from Abaqus and MuPhiSim (example10)

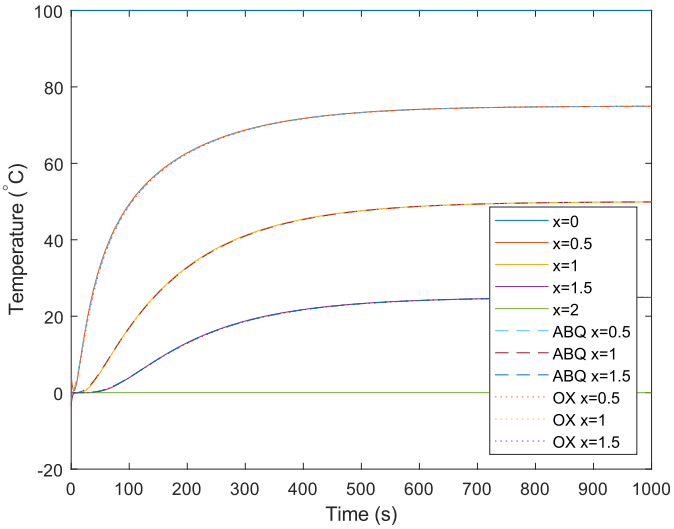


Figure 3.27: Comparison between solutions obtained analytically, from Abaqus and MuPhiSim (example11)

Finally, to illustrate the possibility of defining temperature-dependent parameters, the following Figure 3.28 shows the influence on the temperature profile when K increases at a rate of 0.2 W/mK^2 and when C increases at a rate of 0.05 J/kgK^2, for example10.

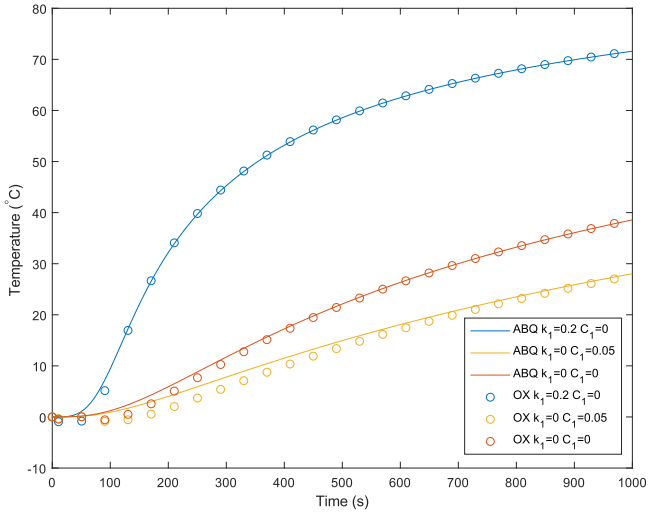
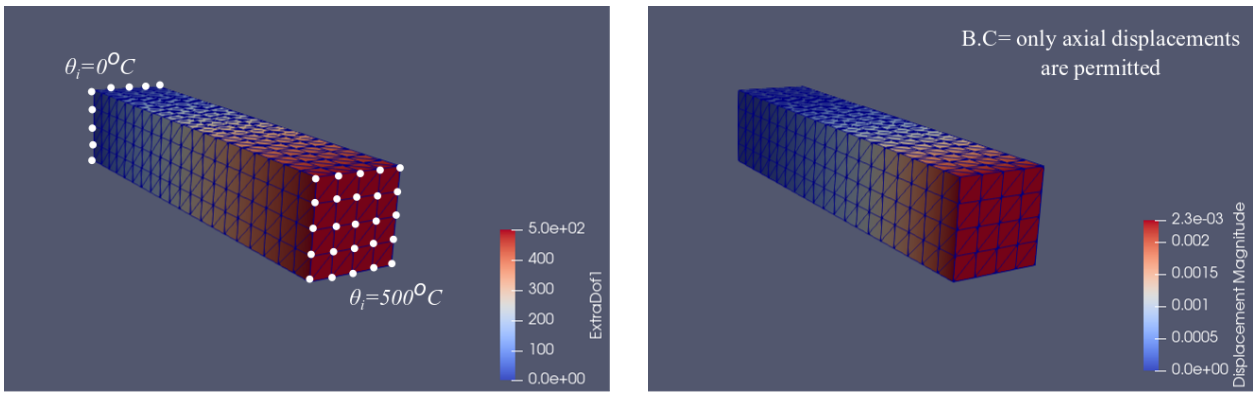


Figure 3.28: Influence of temperature dependence of K and C at x=2. Comparison between Abaqus and MuPhiSim solutions (example10)

3.1.11 Example 12: `example12.inp`

This input file presents thermo-mechanical analysis of a 3D rectangular block under heating conditions (example based on the one presented in [7]). The model is constrained such that only axial displacements are permitted. The mechanical constitutive model used is HyperElastic, St-Venant-Kirchhoff-thermomechanical, and the extra dof model used temperature coupling. One end of the block is held at a temperature of 0 $^{\circ}C$ while the opposite end is raised to 500 $^{\circ}C$ (see Figure 3.29).



(a) Temperature (b) Displacement

Figure 3.29: Temperature and displacement results of example12

3.1.12 Example 13: `example13.inp`

This input file presents a 2D beam made of FEM. The model used is the stochastic HyperElastic, St-Venant-Kirchhoff. The Young's modulus and the Poisson's ratio are supposed to be random Gaussian variables. The standard deviation of the Gaussian variables are ten percent of the mean value, which means that the uncertainties are moderate. An implicit static solver is used. The left side of the beam is blocked while the other side is submitted to a pressure equal to a tenth of the mean value of the Young's modulus.
Figure 3.30 shows the agreement between Monte Carlo and SFEM when it comes to capture the cumulative distribution function of the displacement at the tip of the beam.

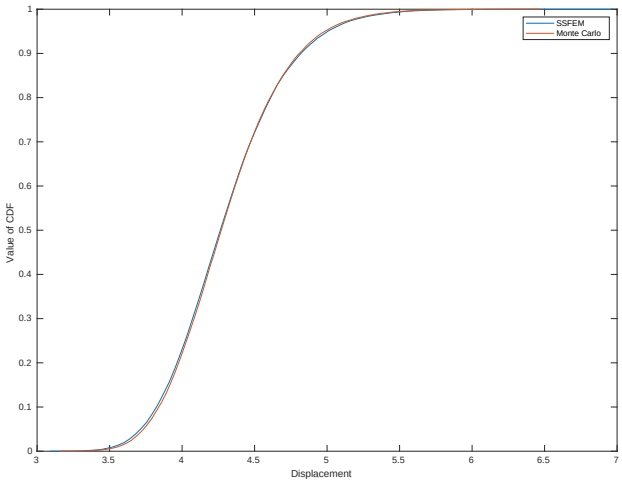
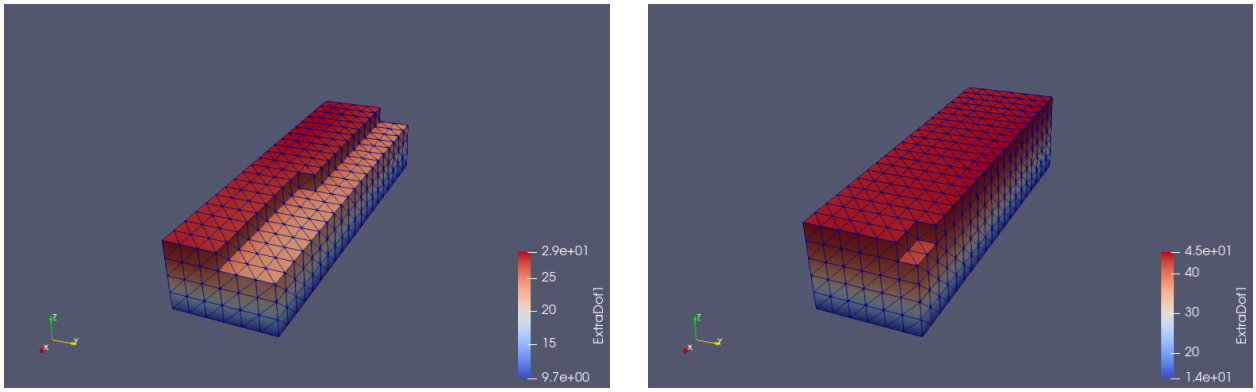


Figure 3.30: Stochastic end tip displacement

3.1.13 Example 14: `example14.inp`

This input file presents a heat transfer analysis with element activation on a 3D beam made of FEM. The elements are activated from the bottom to the top of the beam and following a longitudinal path in each layer of elements. The extradof model used is Temperature. A prescribed volumetric heat flux $r = 10\ W/m^3$ is defined on the beam. The temperature is set at 0 $°C$ on the bottom. The progressive activation of the elements is shown Figure 3.31.



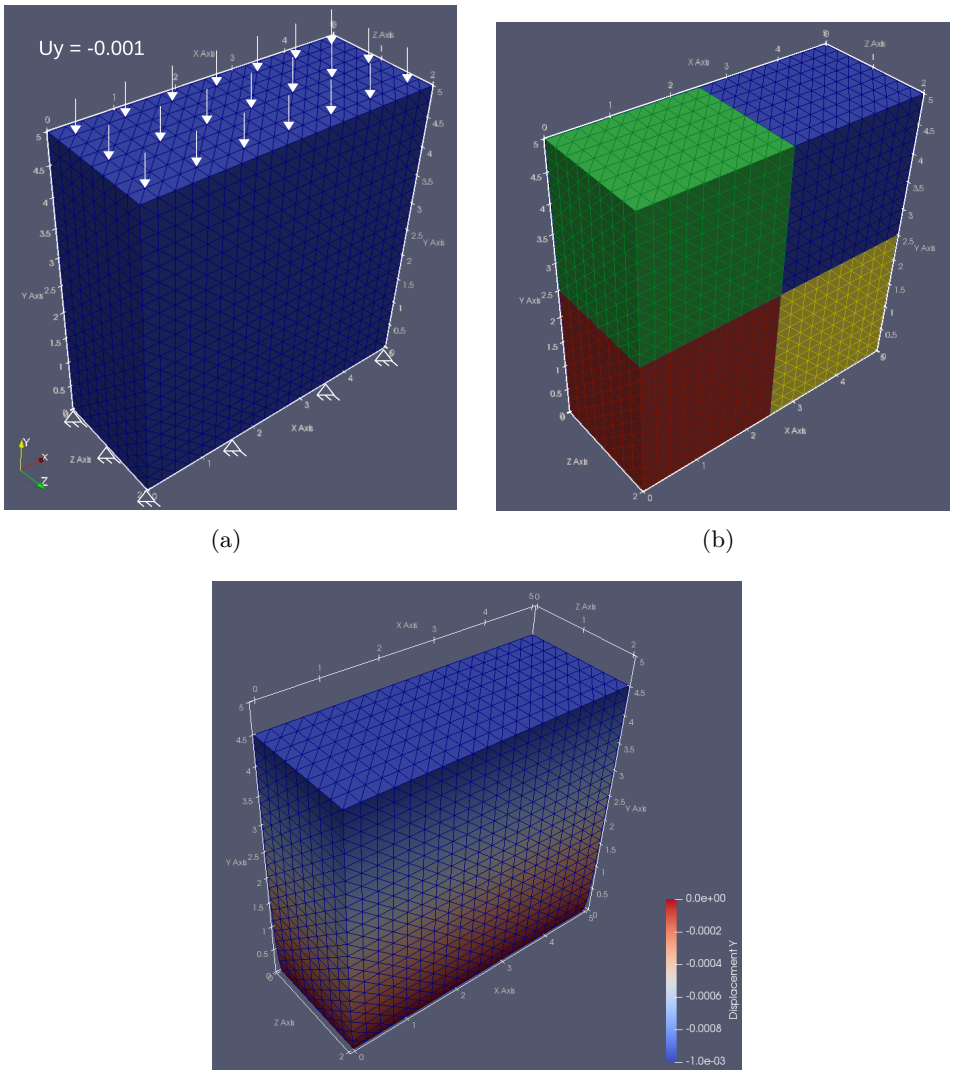
(a) t=11.24s (b) t=14.59s

Figure 3.31: Progressively activation of elements example 14

3.1.14 Example 15: `example15.inp`

This input file presents a 3D block with the dimensions 2x5x5m discretised with MM nodes, and partitioned manually into 4 regions of equal size that we assigned to different processors. Note that elements are only employed to obtain Gauss quadrature nodes and weights and post-processing purposes in this example. In our simulation, we fixed the bottom surface of the block in x, y, and z directions, and displaced the top surface by -0.001m in y direction. The model used is HyperElastic, St-Venant-Kirchhoff (E = 1e9 Pa, $\nu = 0.3$). The displacement is computed by an Implicit Static solver with a scale factor of 10 000 during 1 second.

Figure 3.32 shows the initial configuration with boundary conditions, partitioning, and the distorted configuration.



(a) (b)

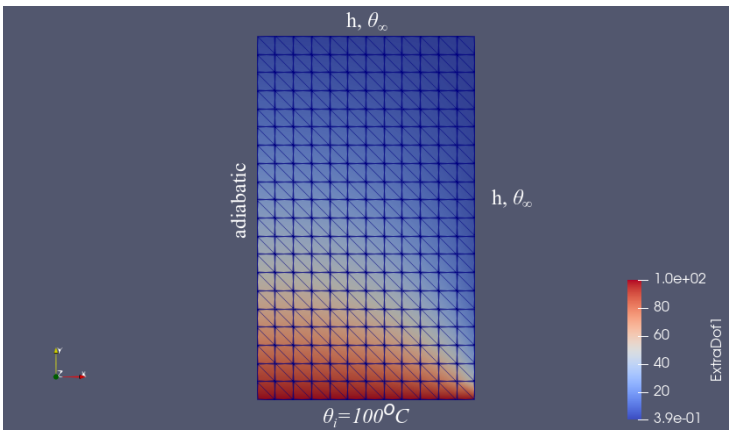
(c)

Figure 3.32: (a) Initial configuration, (b) partitioning into 4 regions assigned to different CPUs, and (c) distorted configuration of the block.

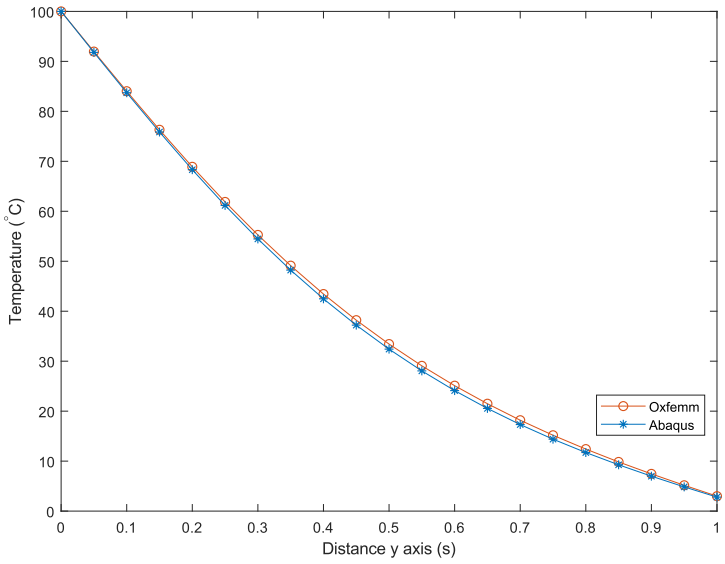
3.1.15 Example 16: `example16.inp`

This input file presents a transient heat problem on a 2D plate with the dimensions 0.6x1m. The temperature is fixed at the bottom surface at 100 $°C$, while convection BCs are imposed on right and top surfaces ($h = 750 \quad W/Km^2$). The temperature profile is computed by an Implicit Static solver with a scale factor of 10000 during 15000 second.

Figure 3.33 shows the temperature profile on the plate and the temperature along the left side (y axis).



(a)



(b)

Figure 3.33: (a) Temperature distribution t=15000s (b) temperature along the left side (y axis)

3.1.16 Summary

Table 3.2: Description of nodes and elements

Example	Dim	Nb_{nod}	$Type_{elt}$	Nb_{elt}	FEM_{nod}	MM_{nod}	$Shared_{nod}$	FEM_{elt}	MM_{elt}
1	3	612	C3D4	2324	354	278	20	1308	1016
2	3	612	C3D4	2324	All	-	-	All	-
3	2	120	CPS3	196	All	-	-	All	-
4	2	476	CPS3	860	All	-	-	All	-
5	2	464	CPS3	825	All	-	-	All	-
6	2	52	CPS3	50	All	-	-	All	-
6bis	2	52	CPS3	50	All	-	-	All	-
7	2	120	CPS3	196	All	-	-	All	-
8	2	63	CPS6	20	All	-	-	All	-
9	2	52	CPS3	50	All	-	-	All	-
10	2	122	CPS3	120	All	-	-	All	-
11	2	441	CPS3	800	All	-	-	All	-
12	3	622	C3D4	2517	All	-	-	All	-
13	2	52	CPS3	50	All	-	-	All	-
14	3	1029	C3D4	4320	All	-	-	All	
15	3	3969	C3D4	19200	-	All	-	-	All
16	2	622	C3D4	480	All	-	-	All	-

Table 3.3: Solvers

Example	Solver	Scale	Outputs	Int Var	Time	Boundary conditions
1	Implicit Static	100000	100	All, 0	0 -> 1	Displacement
	Implicit Dynamic	100	1000	All, 0	1 -> 3	Displacement
2	Explicit	0.01	1000	All, 22	0 -> 1	Displacement
3	Implicit Static	10	1000	All, 0	0 -> 1	Displacement
	Implicit Dynamic	10	100	All, 0	1 -> 10	Displacement
	Implicit Static	10	1000	All, 0	10 -> 14	Displacement
	Implicit Dynamic	10	1000	All, 0	14 -> 20	Displacement
4	Implicit Static	1000000	1000000	All, 15	0 -> 2500	Displacement Pressure Ramp & Inst
5	Implicit Static	1	500	All, 0	0 -> 5	Displacement
6	Implicit Static	10	200	All, 0	0 -> 200	Displacement Flux ExtraDof-Current
6bis	Implicit Static	10	200	All, 0	0 -> 200	Displacement Flux ExtraDof-Current
7	Explicit	0.1	5000	All, 0	0 -> 2.5	Displacement Pressure Inst
	Explicit Bulk viscosity 2, 3	0.1	5000	All, 0	2.5 -> 5	Displacement Pressure Inst
8	Implicit Static	10	1000	All, 0	0 -> 1	Displacement
	Explicit	0.1	1000	All, 0	1 -> 4	Displacement
9	Implicit Static	1	200	All, 0	0 -> 1	Displacement/temperature Flux ExtraDof-Heat
10	Implicit Static	100	200	All, 0	0 -> 1	Displacement/temperature
	Implicit Static	100	200	All, 0	1 -> 1000	Displacement/temperature
11	Implicit Static	1000	200	All, 0	0 -> 1	Displacement/Temperature
	Implicit Static	1000	200	All, 0	1 -> 1000	Displacement/Temperature
12	Implicit Static	10	200	All, 0	0 -> 1	Displacement ExtraDof-Temperature
13	Implicit Static	100	200	All, 0	0 -> 1	Displacement Pressure Inst
14	Implicit Static	400	200	All, 0	0 -> 15	Temperature Volumetric Flux-Heat
15	Implicit Static	10000	100	All, 0	0 -> 1	Displacement
16	Implicit Static	10000	500	All, 0	0 -> 15000	Temperature Convection

3.2 Error analysis

The error analyses can be performed by computing the relative L_2 and H_1 errors between the current numerical simulation and an analytical solution. These error are estimated as

$$\varepsilon_{L_2} = \frac{\left|\mathbf{u} - \mathbf{u}^{\text{exact}}\right|_{L_2}}{\left|\mathbf{u}^{\text{exact}}\right|_{L_2}} \text{ and } \varepsilon_{H_1} = \frac{\left|\mathbf{u} - \mathbf{u}^{\text{exact}}\right|_{H_1}}{\left|\mathbf{u}^{\text{exact}}\right|_{H_1}}, \tag{3.1}$$

where $\mathbf{u}$ and $\mathbf{u}^{\text{exact}}$ are respectively the simulated and exact solutions, and the L_2 and H_1 norms are defined by

$$|\mathbf{a}|_{L_2} = \sqrt{\int_{\Omega_0} \mathbf{a} \cdot \mathbf{a}\, dV} \text{ and } |\mathbf{a}|_{H_1} = \sqrt{\int_{\Omega_0} \left(\mathbf{a} \cdot \mathbf{a} + L^2 \left(\mathbf{a} \otimes \nabla\right) : \left(\mathbf{a} \otimes \nabla\right)\right) dV}, \tag{3.2}$$

with $\mathbf{a}$ being an arbitrary vector field and Ω_0 being the domain under investigation.

The error analysis is introduced in the input file as

```
*ERROR ANALYSIS
CantileverBeamUnderParabolicTraction,3.0,1.0,2000.0,0.3,150.0
```

where ***ERROR ANALYSIS** is the mandatory keyword. The next line specifies the analytic solution of the problem. Here, **CantileverBeamUnderParabolicTraction** is the name of the problem followed by a list of the parameters. Other problems need to be coded similarly in the source by deriving the virtual class *trueDisplacementField*.

The values of the relative L_2 and H_1 errors are stored in *errorData*.csv* in the output folder.

Table 3.4: Materials

Example	Name	Density	Material	Coefficients	Nb_{elt}
1	Region1	1000	HyperElastic, St-Venant-Kirchhoff	1E9, 0.3	All
2	Region1	1000	ViscoGrowth	158E9 - 1 - 73E8, 73E8 - 1000E6 - 2	All
3	Region1	1000	HyperElastic, St-Venant-Kirchhoff	1E9, 0.3	All
4	membraneCortex	1000	ViscoMechContract-AxialGrowth	158 - 1 - 73, 73 - 1000 - 1, 0.0295, 0.0265, 1.1101, 0.00001, 30, 1, 0	132
	axonShaft	1000	ViscoMech-AxialGrowth	250 - 1 - 57.5, 57.5 - 1000E-3 - 1, 0.0295, 0.0265, 1.1101, 1, 0	728
5	All	1000	HyperElastic, St-Venant-Kirchhoff	1E6, 0.3	All
6	Region1	1000	HyperElastic, St-Venant-Kirchhoff	1E7, 0.4	All
	-	-	FHNelec	1, 0, 0, 1, 0, 0.4, 0.4, 0.01, 0.085, 0	All
6bis	Region1	1000	ViscoMech-AxialGrowth	250 - 1 - 57.5, 57.5 - 1000E-3 - 1, 0.0295, 0.0265, 1.1101, 1, 0	All
	-	-	FHNelec	1, 0, 0, 1, 0, 0.4, 0.4, 0.01, 0.085, 0	All
7	Region1	100	HyperElastic, St-Venant-Kirchhoff	1E9, 0.3	All
8	my-mat	1000	HyperElastic, St-Venant-Kirchhoff	1E9, 0.3	All
9	Region1	8930	HyperElastic, Neo-Hookean	1E7, 0.4	All
	-	-	Temperature	0, 0, 401, 0, 0, 0, 0	All
10	Region1	920	HyperElastic, Neo-Hookean	1E7, 0.4	All
	-	-	Temperature	2.04, 0, 5, 0, 0, 0, 0	All
11	Region1	920	HyperElastic, Neo-Hookean	1E7, 0.4	All
	-	-	Temperature	2.04, 0, 5, 0, 0, 0, 0	All
12	Region1	1000	HyperElastic, St-Venant Kirchhoff-thermomecha	1E6, 0.3, 1E-6, 0	All
	-	-	TemperatureCoupling	0, 0, 100, 0, 0, 0, 0	All
13	Region1	1000	Stochastic HyperElastic, St-Venant-Kirchhoff	1E7, 0.4	All
14	Region1	1000	HyperElastic, St-Venant -Kirchhoff-3Dprinting	120E6, 0.3	All
	-	-	Temperature	0, 0, 1, 0, 0, 0, 0	All
15	Region1	1000	HyperElastic, St-Venant-Kirchhoff	1E9, 0.3	All
16	Region1	7850	HyperElastic, Neo-Hookean	1E7, 0.4	All
	-	-	Temperature	434, 0, 52, 0, 0, 0, 0	All

CHAPTER 4

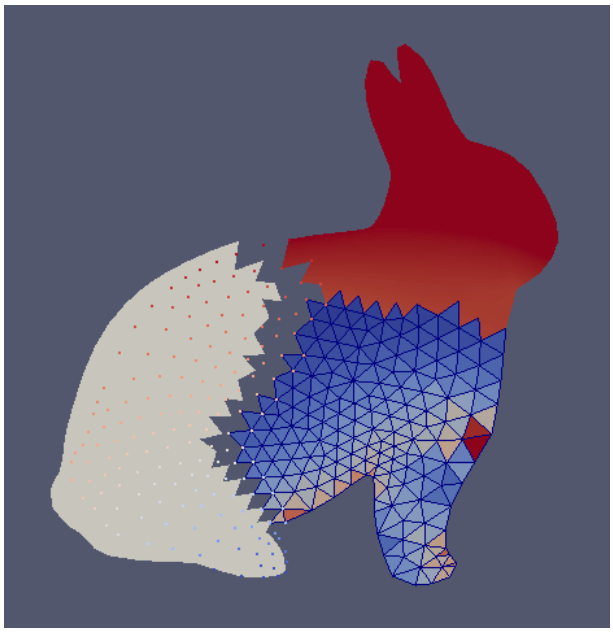
OUTPUT FILES AND PARAVIEW

The data output of all simulations is the in .vtk format. In the VTK format, all information is assigned to each node as well as to each element/cell integration presented in the spatial discretisation.

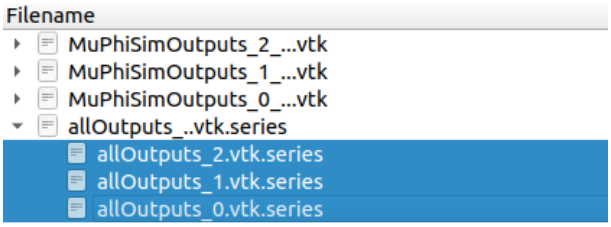
For sequential simulations, the simulation results can be visualised in Paraview by loading the series file (allOutputs_0.vtk.series) only.

For parallel simulations, each processor will have its own set of outputs and series file. In Paraview series files should be loaded independently and not as a chain of files (by expanding the tree and selecting all the series file one by one). The deformed configuration can be seen by selecting the "Wrap by Vector" icon, Figure 4.2.

The most frequently used filters in Paraview are: "Plot selection over time", "ProbeLocation","Plot over line" and "Plot global variables over time". Figure 4.3 shows velocity vectors plot on 3 processors. Figure 4.4 shows how to plot the current time. Details about each filter are explained in the Paraview documents.

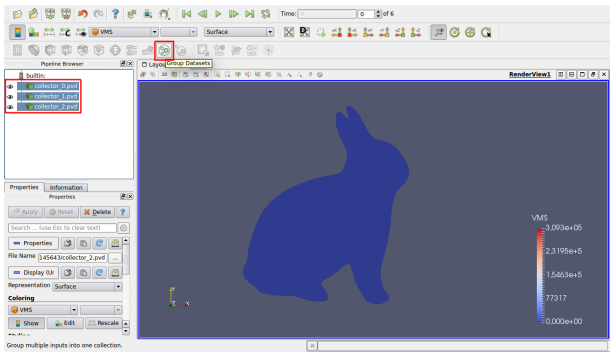


(a) Three collectors in parallel

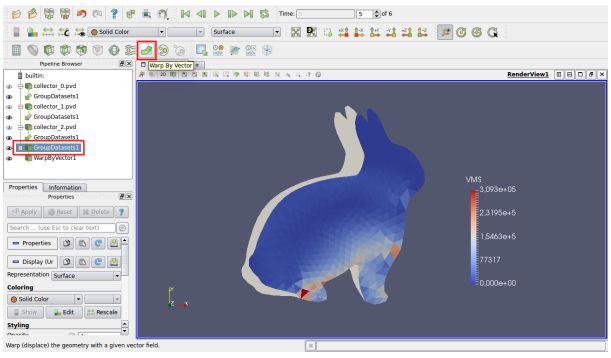


(b) In parallel, select all the series

Figure 4.1: Series files in parallel



(a) Group Datasets



(b) Wrap by vector, get the deformed object

Figure 4.2: Visualisation of the deformed grouped regions

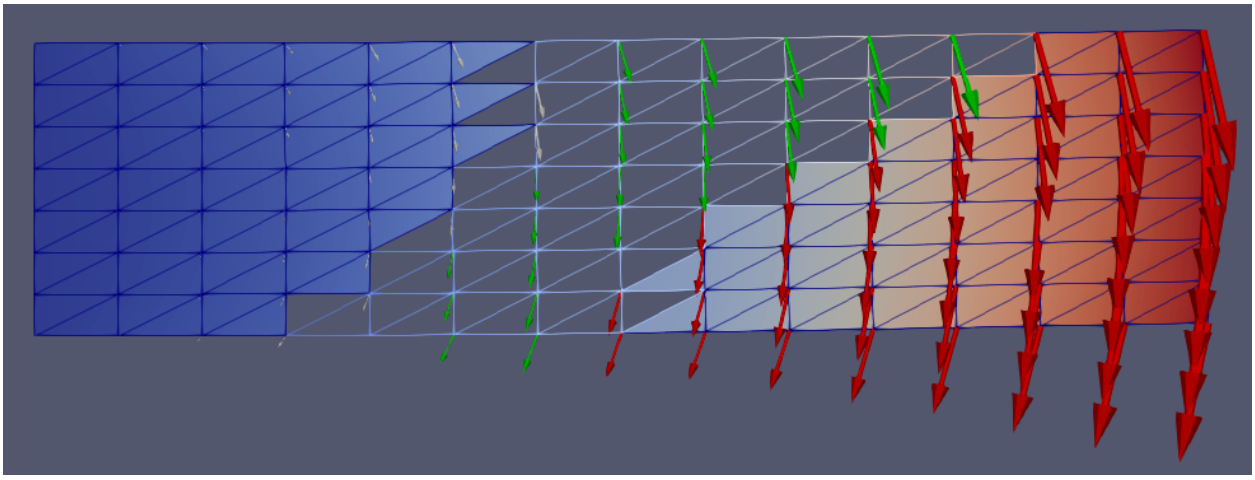


Figure 4.3: Velocity vectors displayed for 3 processors

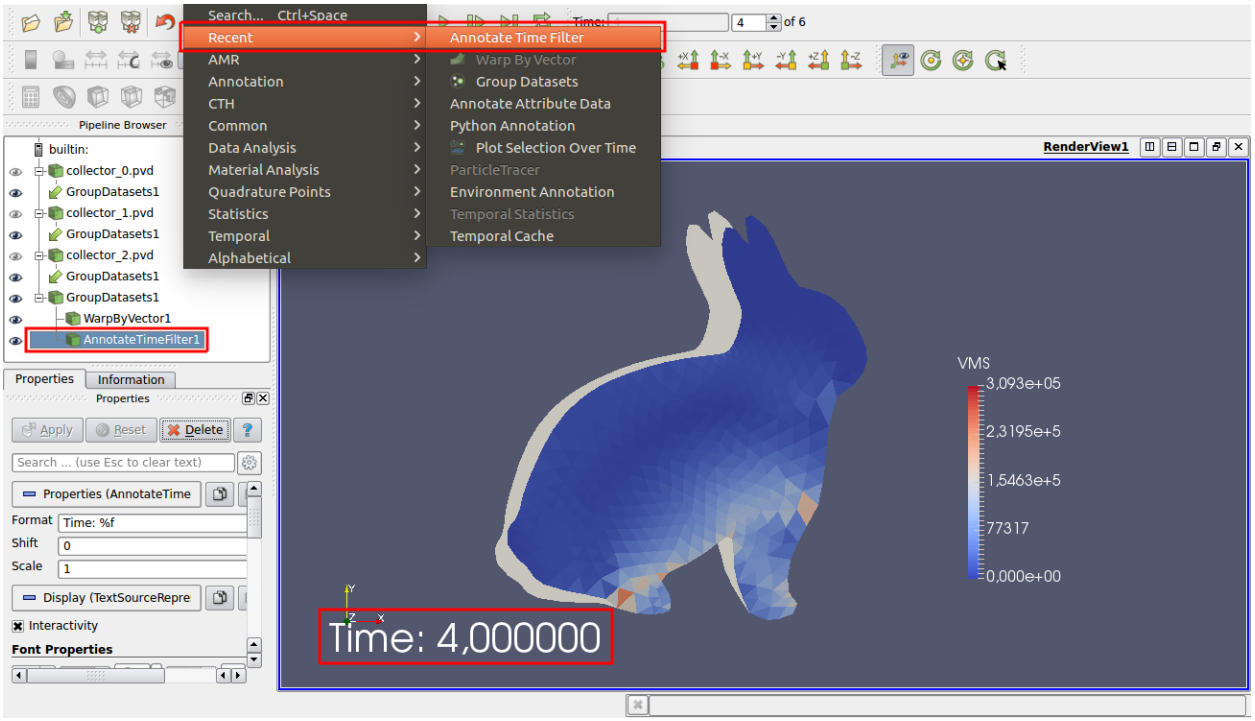


Figure 4.4: Annotate time filter

CHAPTER 5

Finite element framework theory

5.1 SPATIAL DISCRETISATION

Spatial discretisation: coupling FEM and MM methods

This chapter aims at explaining the concepts related to the spatial discretisation of the problem. *MuPhiSim* couples finite element and meshless methods, FEM and MM respectively.

5.1.1 Introduction

Before starting with the concept of finite deformation, we first define the general scheme of notation used in this report. Scalars are written as lightface letters, vectors (1D tensors) as boldface lower case letters (e.g., $\boldsymbol{a}$, $\boldsymbol{b}$) and higher dimensional tensors as boldface upper case letters (e.g., $\boldsymbol{A}$, $\boldsymbol{B}$).

The dot product (e.g., scalar product for two vectors or matrix product for two second-order tensors) is denoted by "$\cdot$" (e.g., $\boldsymbol{a} \cdot \boldsymbol{b}$ and $\boldsymbol{A} \cdot \boldsymbol{B}$, respectively). More generally, all inner products are indicated by a number of similar dots corresponding to the number of indices relating to the product. For example, the double-dot product between two vectors matrices (or Frobenius inner product) is given by $\boldsymbol{A} : \boldsymbol{B}$. The transpose of a tensor is denoted by a superscript "T" and its inverse by a superscript "-1".

A modified Einstein index notation where indices are not necessarily alternatively superscript and subscript is adopted when convenient, e.g., $\boldsymbol{a} \cdot \boldsymbol{b} = a_i b_i$, $\boldsymbol{A} \cdot \boldsymbol{b} = A_{ij} b_j$, $\boldsymbol{A} : \boldsymbol{B} = A_{ij} B_{ij}$ or $\boldsymbol{K} : \boldsymbol{B} = K_{ijkl} B_{kl}$ where $\sum_i$, $\sum_j$, $\sum_{ij}$ and $\sum_{kl}$, respectively, are omitted. Finally, when possible, upper case letters are used for the reference (undeformed) configuration, e.g., X, Y, Z, I, J, K, etc. and lower case letters for the current (deformed) configuration, e.g., x, y, z, i, j, k, etc.

Other notations will be introduced later when required.

Let $\boldsymbol{\varphi} : \mathbb{R}^3 \to \mathbb{R}^3$ be a function that maps a material point $\boldsymbol{X} \in \Omega_0$ in the reference configuration to its corresponding point $\boldsymbol{x} = \boldsymbol{\varphi}(\boldsymbol{X}) \in \Omega$ in the current configuration (see Figure 5.1). The deformation gradient tensor $\boldsymbol{F}$ is defined as

$$\boldsymbol{F} = \operatorname{Grad} \boldsymbol{x} = \boldsymbol{\nabla}_0 \boldsymbol{x} \tag{5.1}$$

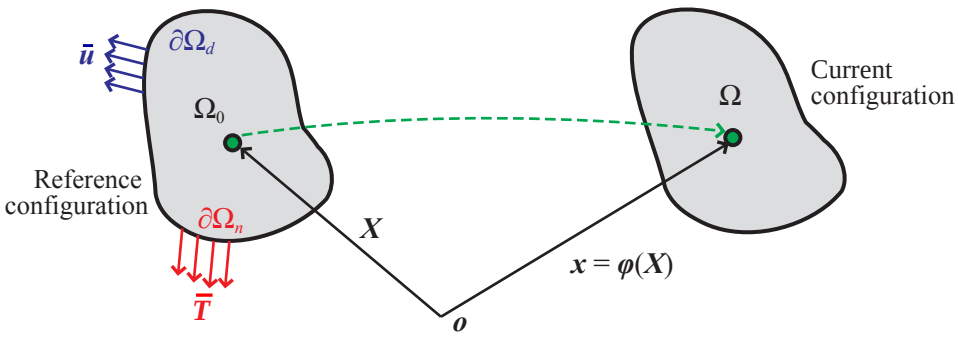


Figure 5.1: Deformable body and the applied boundary conditions in the reference (undeformed) and current (deformed) configurations.

where Grad or $\boldsymbol{\nabla}_0$ is the gradient operator with respect to the reference configuration. In Cartesian basis, $\boldsymbol{F} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}}$ and its components are given by $F_{iJ} = \frac{\partial x_i}{\partial X_J}$. Since $\boldsymbol{x}(\boldsymbol{X}) = \boldsymbol{X} + \boldsymbol{u}(\boldsymbol{X})$, where $\boldsymbol{u}$ is the displacement vector, the deformation gradient can be given by $\boldsymbol{F} = \boldsymbol{I} + \boldsymbol{\nabla}_0 \boldsymbol{u}$.

5.1.1.1 Balance of linear momentum with respect to the reference configuration

The equation of the balance of linear momentum with respect to the reference (or undeformed) configuration can be written as

$$\operatorname{Div} \boldsymbol{P} + \rho_0 \boldsymbol{b} = \rho_0 \ddot{\boldsymbol{x}} \,, \quad \forall \boldsymbol{X} \in \Omega_0 \tag{5.2}$$

where ρ_0 is the mass density (per unit reference volume) and Div is the divergence operator with respect to the reference configuration, such that $(\operatorname{Div} \boldsymbol{P})_i = \frac{\partial P_{iJ}}{\partial X_J} = P_{iJ,J}$. In this expression, the second-order tensor $\boldsymbol{P}$ and vector $\boldsymbol{b}$ stand for, respectively, the first Piola-Kirchhoff stress and the body force per unit mass.

Alternative stress measures typically used in the finite deformation framework include Cauchy stress (or true stress) $\boldsymbol{\sigma} = J^{-1} \boldsymbol{P} \cdot \boldsymbol{F}^T$, Kirchhoff stress $\boldsymbol{\tau} = J\boldsymbol{\sigma} = \boldsymbol{P} \cdot \boldsymbol{F}^T$, and the second Piola-Kirchhoff stress $\boldsymbol{S} = \boldsymbol{F}^{-1} \cdot \boldsymbol{P}$, where $J = \det \boldsymbol{F}$ is the Jacobian, i.e., the volumetric change relative to the original configuration.

5.1.1.2 Balance of angular momentum with respect to the reference configuration

The balance of angular momentum is generally not imposed directly through the weak form, but through the constitutive model, by prescribing that $\boldsymbol{\sigma}$, $\boldsymbol{S}$ or $\boldsymbol{P} \cdot \boldsymbol{F}^T$ is symmetric.

5.1.1.3 Boundary conditions

The Neumann and Dirichlet boundary conditions, $\bar{\boldsymbol{T}}$ on $\partial\Omega_n$ and $\bar{\boldsymbol{u}}$ on $\partial\Omega_d$, respectively, are imposed in the reference configuration by:

$$\begin{cases} \boldsymbol{P} \cdot \boldsymbol{N} = \bar{\boldsymbol{T}}, \ \forall \boldsymbol{X} \in \partial\Omega_n \\ \boldsymbol{u} = \bar{\boldsymbol{u}}, \ \forall \boldsymbol{X} \in \partial\Omega_d \end{cases} \tag{5.3}$$

where $\boldsymbol{N}$ is the normal to the boundary in the reference configuration, and where the subscript "0" (indicating the reference configuration) of $\partial\Omega_n$ and $\partial\Omega_d$ was dropped for clarity, see Figure 5.1.

5.1.1.4 Weak form of the balance of momentum

The weak form of the balance of linear momentum (see Equation (5.2)) can be formulated as follows: For all arbitrary admissible virtual displacement $\boldsymbol{\eta}$, with $\boldsymbol{\eta}(\boldsymbol{X}) = \boldsymbol{0}$ for all $\boldsymbol{X} \in \partial\Omega_d$,

$$\int_{\Omega_0} \operatorname{Div} \boldsymbol{P} \cdot \boldsymbol{\eta} \, \mathrm{d}V + \int_{\Omega_0} \rho_0 \boldsymbol{b} \cdot \boldsymbol{\eta} \, \mathrm{d}V = \int_{\Omega_0} \rho_0 \ddot{\boldsymbol{x}} \cdot \boldsymbol{\eta} \, \mathrm{d}V \tag{5.4}$$

Equation (5.4) is called the weak form, since it requires the condition of balance of linear momentum to be satisfied only as as an integral over the entire domain, whereas Equation (5.2) is referred to as the strong form (or canonical form) because it fulfills the conditions locally at each point $\boldsymbol{X} \in \Omega_0$.

Integrating by parts and using Green's theorem, the above expression can be rewritten as

$$\int_{\partial\Omega_0} (\boldsymbol{P} \cdot \boldsymbol{N}) \cdot \boldsymbol{\eta} \, \mathrm{d}S + \int_{\Omega_0} \rho_0 \boldsymbol{b} \cdot \boldsymbol{\eta} \, \mathrm{d}V = \int_{\Omega_0} \boldsymbol{P} : \operatorname{Grad} \boldsymbol{\eta} \, \mathrm{d}V + \int_{\Omega_0} \rho_0 \ddot{\boldsymbol{x}} \cdot \boldsymbol{\eta} \, \mathrm{d}V \tag{5.5}$$

which, noting that $\boldsymbol{\eta} = \boldsymbol{0}$ for all $\boldsymbol{X} \in \partial\Omega_d$ and using Equation (7.5), leads to

$$\int_{\partial\Omega_n} \bar{\boldsymbol{T}} \cdot \boldsymbol{\eta} \, \mathrm{d}S + \int_{\Omega_0} \rho_0 \boldsymbol{b} \cdot \boldsymbol{\eta} \, \mathrm{d}V = \int_{\Omega_0} \boldsymbol{P} : \operatorname{Grad} \boldsymbol{\eta} \, \mathrm{d}V + \int_{\Omega_0} \rho_0 \ddot{\boldsymbol{x}} \cdot \boldsymbol{\eta} \, \mathrm{d}V \tag{5.6}$$

Note that the first Piola-Kirchhoff stress $\boldsymbol{P}$ is a function of the deformation gradient $\boldsymbol{F}$ and, thus, a function of the displacement $\boldsymbol{u}$. The relation between $\boldsymbol{P}$ and $\boldsymbol{F}$ is defined by the material constitutive law $\boldsymbol{P} = \tilde{\boldsymbol{P}}(\boldsymbol{F})$.

5.1.2 Numerical integration

Either Finite Elements Method (FEM) or Meshless Method (MM) is used, the actual undeformed body Ω_0 is discretised over the integration points, which can be related to elements in case of FEM. Lets begin with the numerical integration over the elements and then generalise to the integration points.

In Finite element method, polynomial shape functions $N_a^e(\boldsymbol{\xi})$ are defined for each element e (a is the node number and $\boldsymbol{\xi}$ is the coordinate vector in the element basis—or isoparametric coordinate system), such that, in each element,

$$\boldsymbol{x}^e(\boldsymbol{X}) = \sum_{a \in \Omega_0^e} N_a^e(\boldsymbol{\xi}) \, \boldsymbol{x}^a \quad \text{and} \quad \boldsymbol{\eta}^e(\boldsymbol{X}) = \sum_{a \in \Omega_0^e} N_a^e(\boldsymbol{\xi}) \, \boldsymbol{\eta}^a, \tag{5.7}$$

where $\boldsymbol{x}^a$ is the nodal position (or the solution) of node a. Note that the mapping from the element basis $\boldsymbol{\xi}$ to the reference coordinate system $\boldsymbol{X} = \boldsymbol{\Phi}(\boldsymbol{\xi})$ is prescribed for all elements.

Finite element problem: Find all $\boldsymbol{x}^a \in \mathbb{R}^3$ such that for all admissible virtual displacement $\boldsymbol{\eta}^b \in \mathbb{R}^3$:

$$\begin{aligned} \sum_e \int_{\Omega_0^e} \rho_0 N_a^e N_b^e \ddot{\boldsymbol{x}}^a \cdot \boldsymbol{\eta}^b \, \mathrm{d}V + \sum_e \int_{\Omega_0^e} \left(\tilde{\boldsymbol{P}} \left(\sum_{a \in \Omega_0^e} N_a^e(\boldsymbol{\xi}) \, \boldsymbol{x}^a \right) \cdot \boldsymbol{\nabla}_0 N_b^e \right) \cdot \boldsymbol{\eta}^b \, \mathrm{d}V \\ = \sum_e \int_{\partial\Omega_n^e} N_b^e \bar{\boldsymbol{T}} \cdot \boldsymbol{\eta}^b \, \mathrm{d}S + \sum_e \int_{\Omega_0^e} \rho_0 N_b^e \boldsymbol{b} \cdot \boldsymbol{\eta}^b \, \mathrm{d}V \end{aligned} \tag{5.8}$$

Since the above equation must be valid for all admissible $\boldsymbol{\eta}^b$, the finite element problem now reads: Find $\boldsymbol{x}^a$ such that, for all b,

$$\sum_e \boldsymbol{M}^e_{ba}\ddot{\boldsymbol{x}}^a + \sum_e \boldsymbol{f}^{e\,\mathrm{int}}_b(\boldsymbol{x}^a) = \sum_e \boldsymbol{f}^{e\,\mathrm{ext}}_b \tag{5.9}$$

where the mass matrix $\boldsymbol{M}^e$, the element internal force vector $\boldsymbol{f}^{e\,\mathrm{int}}_b(\boldsymbol{x}^a)$ and external force vector $\boldsymbol{f}^{e\,\mathrm{ext}}_b$ are

$$\boldsymbol{M}^e_{ba} = \int_{\Omega^e_0} \rho_0 N^e_a N^e_b \,\mathrm{d}V \tag{5.10}$$

$$\boldsymbol{f}^{e\,\mathrm{int}}_b(\boldsymbol{x}^a) = \int_{\Omega^e_0} \tilde{\boldsymbol{P}} \left(\sum_{a\in\Omega^e_0} N^e_a(\boldsymbol{\xi})\,\boldsymbol{x}^a \right) \cdot \boldsymbol{\nabla}_0 N^e_b \,\mathrm{d}V \tag{5.11}$$

$$\boldsymbol{f}^{e\,\mathrm{ext}}_b = \int_{\partial\Omega^e_{nh}} N^e_b \bar{\boldsymbol{T}} \,\mathrm{d}S + \int_{\Omega^e_0} \rho_0 N^e_b \boldsymbol{b} \,\mathrm{d}V \tag{5.12}$$

These element tensors are finally calculated using Gauss quadrature approximation in the isoparametric coordinate system. This means that each element e can be integrated upon with a function $f(\boldsymbol{x})$ by assuming that:

$$\int_{\Omega^e_0} f(\boldsymbol{\varphi}(\boldsymbol{X}))\,\mathrm{d}V = \sum_{q\in\Omega^e_0} w_q f(\boldsymbol{\varphi}(\boldsymbol{\Phi}(\boldsymbol{\xi}_q)))J_0(\boldsymbol{\xi}_q) \tag{5.13}$$

where $\boldsymbol{\xi}_q$ is the coordinate vector of Gauss point q of element e in the isoparametric coordinate system, w_q is its corresponding weight, and J_0 is the Jacobian between the isoparametric coordinate system and the reference configuration. Noting that each mass matrix element M^e_{ba} relates $\ddot{\boldsymbol{x}}^a$ to the other force vectors in b coordinate-by-coordinate, the matrix can be extended to the nodal coordinates by $M^e_{ba} = \delta_{ik} M^e_{ibka}$ where δ_{ik} is the Kronecker symbol. Swapping a and b, one has in indicial notation:

$$M^e_{iakb} = \textstyle\sum_q w_q \rho_0 \delta_{ik} N^e_a(\boldsymbol{\xi}_q) N^e_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{5.14}$$

$$f^{e\,\mathrm{int}}_{ia} = \textstyle\sum_q w_q P_{iJ}(\boldsymbol{\xi}_q) N^e_{a,J}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{5.15}$$

$$f^{e\,\mathrm{ext}}_{ia} = \textstyle\sum_p w_p T_i(\boldsymbol{\xi}_p) N^e_a(\boldsymbol{\xi}_p) J_0(\boldsymbol{\xi}_p) + \tag{5.16}$$

$$\sum_q w_q \rho_0 b_i(\boldsymbol{\xi}_q) N^e_a(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{5.17}$$

The notations with subscripts p (first term of Equation (5.17)) are for the surface elements belonging to $\partial\Omega^e_{dh}$ mapped onto the corresponding faces of the volume elements.

As the internal force vector $\boldsymbol{f}^{e\,\mathrm{int}}$ is not necessarily linear, a Newton-Raphson procedure can be used in conjunction with the time discretisation scheme (see next section). In this case, the element stiffness matrix $\boldsymbol{K}^e$ is needed:

$$Res^e_{ia} = f^{e\,\mathrm{int}}_{ia} - f^{e\,\mathrm{ext}}_{ia} \tag{5.18}$$

$$K^e_{iakb} = \frac{\partial Res^e_{ia}}{\partial x_{kb}} = \frac{\partial f^{e\,\mathrm{int}}_{ia}}{\partial x_{kb}} = \sum_q w_q \frac{\partial P_{iJ}}{\partial F_{kL}}(\boldsymbol{\xi}_q) N^e_{a,J}(\boldsymbol{\xi}_q) N^e_{b,L}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{5.19}$$

where Res is the remainder (or residual). The tangent moduli $\frac{\partial P_{iJ}}{\partial F_{kL}}$ are provided by the constitutive law at each time step.

Meshless Method: In the same way, for Meshless method, polynomial shape functions can be written $N_a^q(\boldsymbol{\xi})$ and are defined for each Gauss point q of the body Ω_0, such that in the neighbourhood of this integration point Ω_0^q,

$$\boldsymbol{x}^q(\boldsymbol{X}) = \sum_{a \in \Omega_0^q} N_a^q(\boldsymbol{\xi})\, \boldsymbol{x}^a \quad \text{and} \quad \boldsymbol{\eta}^q(\boldsymbol{X}) = \sum_{a \in \Omega_0^q} N_a^q(\boldsymbol{\xi})\, \boldsymbol{\eta}^a, \tag{5.20}$$

where $\boldsymbol{x}^a$ is the nodal position (or the solution) of node a that is a closed neighbour of q.

Since this formulation is more general and works for FEM, it has been implemented in *MuPhiSim* for both methods. Independent of the numerical approximation (i.e., FEM and MM) chosen, the numerical approximations of the of the degrees of freedom can be written in the same way. The shape functions N and their derivatives will be different for FEM and MM, as well as the summation in the case of FEM will be for the number of nodes in the element, whereas for MM will be over the number of nodes in the neighborhood of the evaluation point. The summation convention is assumed from now on, but take into account the n is different for the methods explored. Equations 5.14 to 5.19 are the same for MM but to obtain the global matrix, the summation is directly made on the Gauss point and not on the elements.

5.1.3 Shape functions for FEM and MM

The domain Ω can be discretised by means of FEM, MM or both (Coupling). It is possible to have two different domains: FEM and MM. Exploiting the Kronecker delta property of the *max-ent* shape functions of the MM domain, there is a group of nodes that are shared by the FEM and MM domains and appear in both lists of nodes of each domain. The values of the unknown variables at the shared nodes will be equal, and then the coupling between both regions is naturally obtained.

Regardless of the method used (FEM/MM/Coupling), the unknown is approximated as follows:

$$\boldsymbol{u}^h(\boldsymbol{x}) = \sum_{i=1}^{n} N_i(\boldsymbol{x}) u_i = \boldsymbol{N}(\boldsymbol{x})\boldsymbol{u} \tag{5.21}$$

where $u^h(\boldsymbol{x})$ is the approximation of the function u, with u_i coefficients, $\boldsymbol{x}$ is the position of the point at which these quantities are being evaluated, and n is the number of points in the neighbourhood of $\boldsymbol{x}$ (in other words, n is the number of nodes for which the evaluating point $\boldsymbol{x}$ is inside their domains of influence). For FEM, n is the number of nodes of the element of which the evaluating point belongs. The shape function $N(\boldsymbol{x})$ will depend again on the method used. This is one of the main differences between *MuPhiSim* and other FEM programs: the whole program has been built around the GPs data structures and NOT around elements.

5.1.3.1 FEM shape functions

Let $\Omega_{0h} = \bigcup_e \Omega_{0h}^e$ be the numerical approximation to the actual undeformed body Ω_0 (see Figure 5.2), and $\boldsymbol{\varphi}_h$ be a piecewise polynomial approximation of degree k to the actual deformation $\boldsymbol{x} = \boldsymbol{\varphi}(\boldsymbol{X})$, such that

$$\boldsymbol{\varphi}_h \in \left\{ \boldsymbol{\varphi}_h \in C^0(\Omega_{0h}) \,|\, \boldsymbol{\varphi}_h^e = \boldsymbol{\varphi}_h|_{\Omega_{0h}^e} \in \mathbb{P}^k(\Omega_{0h}^e) \,\forall\, \Omega_{0h}^e \in \Omega_{0h} \right\} \tag{5.22}$$

Following the Galerkin method, $\boldsymbol{\eta}_h$, the polynomial approximation to the virtual displacement $\boldsymbol{\eta}$, is chosen in the same ensemble with the additional condition that $\boldsymbol{\eta}_h^e = \boldsymbol{\eta}_h|_{\Omega^e_{0h}} = \mathbf{0}$ on the discretised boundary $\partial\Omega_{dh}$ of $\partial\Omega_d$ for all elements e.

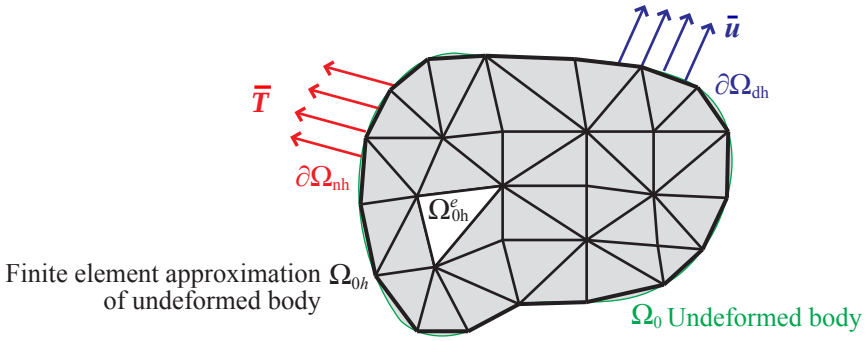


Figure 5.2: Finite element discretisation of the undeformed body Ω_0 into elements Ω^e_{0h}, such that $\Omega_{0h} = \bigcup_e \Omega^e_{0h}$.

We define the shape function $N_a(\xi)$ as a polynomial of order k such that,

$$N_a(\xi_b) = \delta_{ab} \tag{5.23}$$

for nodes a and b of element e. By knowing the displacement of all the nodes a, we can approximate $\varphi(\boldsymbol{X})$ by:

$$\varphi_h(\boldsymbol{X}) = \varphi^e_h(\boldsymbol{X}) = \sum_{a \in \Omega^e_{0h}} N^e_a(\xi) x^a \tag{5.24}$$

5.1.3.2 MM Max-Entropy shape functions

The formulation exposed in this section is based on the one shown by Sukumar [8]. The local *max-ent* shape approach comes from information theory which measures the uncertainty of a finite scheme, in this case the set of nodes, which is termed information entropy [9]. In such a framework, the shape functions of the nodes are thought as a set of probabilities of N independent variables. N is the total number of nodes in which the domain has been discretized. The problem is also subjected to the zero and first order conditions, and also as the shape functions are considered as probabilities, that implies that the shape functions should be positive. The final problem can be summarized as follows:

$$\begin{aligned} \max\, H(N, w) &= -\sum_{i=1}^{n} N_i(\boldsymbol{x}) ln\left(\frac{N_i(\boldsymbol{x})}{w_i(\boldsymbol{x})}\right) \\ N_i(\boldsymbol{x}) &\geq 0 \\ \sum_{i=1}^{n} N_i(\boldsymbol{x}) &= 1 \\ \sum_{i=1}^{n} N_i(\boldsymbol{x}) \boldsymbol{x}_i &= \boldsymbol{x} \end{aligned} \tag{5.25}$$

where the shape functions $N_i(\boldsymbol{x})$ will depend on the prior function $w_i(\boldsymbol{x})$ used. The main advantage of this approach is that the shape functions have the Kronecker delta property at the convex Hull of the domain, thus simplifying the imposition of the essential boundary conditions.

The formulation can be stated as an optimization problem in which the unknown variables are d Lagrange multipliers $\boldsymbol{\lambda}$, where d is the dimension of the problem. The solutions of the optimization problem are guaranteed for the nodal distribution inside the convex hull, see [1] for more details. The shape functions can be written as follows:

$$\begin{aligned} N_i(\boldsymbol{x}) &= \frac{Z_i(\boldsymbol{x})}{Z(\boldsymbol{x})} \\ Z(\boldsymbol{x}) &= \sum_{i=1}^{n} Z_i(\boldsymbol{x}) \end{aligned} \tag{5.26}$$

where $Z_i(\boldsymbol{x})$ will depend on the type of compact support used.

Finally the problem is to find $\boldsymbol{\lambda}$ such that:

$$\boldsymbol{\lambda} = \arg\min\, ln(Z(\boldsymbol{x}, \boldsymbol{\lambda})) \tag{5.27}$$

can be solved with many different algorithms such as steepest descent, Newton's method, quasi-Newton, and so forth.

We use the formulation proposed by Sukumar in order to provide the locality to the shape functions. The method of giving locality to the shape functions by [8] is as simple as using a prior estimate $w_i(\boldsymbol{x})$, in this case a weight function, inside the formulae shown in Equations (5.25), (5.26) and (5.27). In this study, the cubic spline weight function is used (although other weight functions can be plugged in the formulation). The 1D version of such a weight function is defined as following (see also Figure 5.1.3.2):

$$w(r) = \begin{cases} \frac{2}{3} - 4r^2 + 4r^3 & \text{if } r \leq \frac{1}{2} \\ \frac{4}{3} - 4r + 4r^2 - \frac{4}{3}r^3 & \text{if } \frac{1}{2} < r \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.28}$$

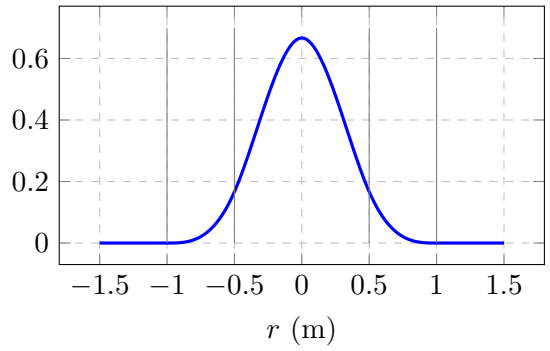


Figure 5.3: Weight function for Sukumar formulation

where $r = \frac{|x - x_i|}{d_{max} c_i}$ with d_{max} as a parameter for the size of the support domain and c_i the maximum distance to the neighbouring nodes. For 2D and 3D versions of this weight function, see References [6, 9]. The parameter that determines the size of the support domain, d_{max}, is set as constant for the whole discretization and for all simulations. The implementation of these shape functions is not straightforward. *MuPhiSim* uses an external open source library provided by [8], which is programmed in Fortran 90. It was necessary to rewrite some functions to be able to call them properly in *MuPhiSim*.

A function was written to calculate the support domain of each node. The implementation of this had to be handled with care. A sophisticated algorithm was implemented for this purpose, which provided r_{max} to the Sukumar library. The algorithm is a "*brute force*" algorithm and can be summarised as follows:

1. The entire set of nodes is considered. For each node i, the euclidean distance is calculated by means of the dot product of the distance to each other node and the corresponding square root.

2. For node i, the distances are sorted out from the closest to the farthest nodes
3. The first *juliS* elements are taken and a weighted average is used to calculate to calculate the average distance around the node i.
4. The previous value, which is independent from the units and the size of the problem, is multiplied to the *dm* factor to defining the neighbouring and radius of the support domain. Values of 3.2 works.
5. Finally, the rmax(i) array is filled with such value for each node i.

Once the support domains are evaluated, for each GP, the support domains of the nodes in which it lies are stored in each neigh-boring nodes array of the GP. Finally, the shape functions as well as the first and second derivatives are calculated by calling the Sukumar library.

5.1.4 Numerical integration: distribution of the GPs

In this subsection, all distribution algorithms implemented in *MuPhiSim* are explained with the corresponding links and references. Currently, *MuPhiSim* supports finite element discretisation of the triangularly arranged sets of nodes in 2D simulations, and tetrahedral sets in 3D. *MuPhiSim* can also run simulations with both linear and quadratic interpolation, and below can be found a complete set of elements implemented:

- **2D - Triangles :** CPS3, CPS6
- **3D - Tetrahedras :** C3D4, C3D10

After discretisation, various quantities are left as integrals. The method chosen is Gauss quadrature which is a numerical method that bases on the idea that an integral of a function $f(\boldsymbol{x})$ over a region Ω can be written as a weighted sum of some functional values at different points in the domain, and can be mathematically expressed as:

$$\int_{\Omega} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \sum_{q} w_q f(\boldsymbol{x}_q) \tag{5.29}$$

These points are called Gauss points (often abbreviated as GPs), and the biggest advantage of *MuPhiSim* is that it bases the calculations on the Gauss points instead of the nodes. The idea is to approximate the integral quantities with some function over the element, and then use Gauss quadrature to compute its value. When the domain is discretised, interpolation functions are used to obtain the displacement within the element based on the displacement of the nodes. The interpolation functions are chosen to be polynomials $P_n(\mathbf{x})$, and the degree of these polynomials are what determine the order of the integral approximations. Higher order approximations yield a more accurate value of the integral, and ideally mimic the function as n tends to infinity, but are computationally more expensive as more gauss points and elemental nodes are required.

5.1.4.1 Bulk elements

5.1.4.2 2D simulations

The linear element in 2D currently implemented in CPS3, and is shown on Figure 5.4. The element has 3 corner nodes and one GP.

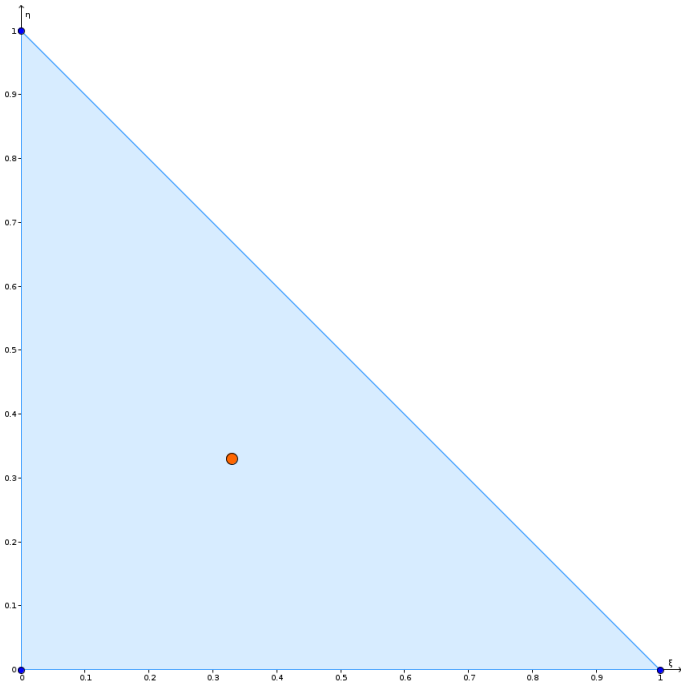


Figure 5.4: CPS3 element boundaries (blue) and GPs (orange)

Quadratic elements also available, and the ones used in *MuPhiSim* are CPS6, shown on Figure 5.5. These elements consist of 6 nodes and 3 GPs. The use of quadratic elements will improve computational accuracy, as well as capture important non-linearities and boundry curvatures, which can be quite important when looking at variables such as stress distributions.

5.1.4.3 3D simulations

Linear elements in 3D that are currently available are of tetrahedral arrangement, and are called C3D4. Their configuration is shown in Figure 5.6. They consist of 4 nodes, which form a pyramid, and have 1 GP.
The quadratic analogue is C3D10 and has 4 corner nodes, in addition to 6 mid nodes. 4 GPs are considered. A figure of the isoparametric configuration is displayed in Figure 5.7.
These elements were created following the work of [3].

5.1.4.4 Surface elements

Certain Neumann boundary conditions such as pressure are applied to the *surface* of the elements as opposed to the nodes directly. It is therefore imperative to convert these to equivalent nodal quantities, which requires us to mesh these bulk elements with corresponding surface elements.
Triangular bulk elements can be meshed with lines, which requires the creation of a 1D line element, and such a linear line will contain two end nodes, and 1 GP (figure 5.8)
For 3D, the surface of C3D4 is a linear triangle, so a new element is not required.
In the case of the quadratic simulations; CPS6 is meshed with a quadratic line (figure 5.10), and C3D10 can use the quadratic CPS6 for its surface, as the node topology can be exactly matched.
However, practically quadratic surfaces can cause symmetry issues at low mesh resolutions, so *MuPhiSim* also supports use of linear surfaces on quadratic bulk elements. That is done through

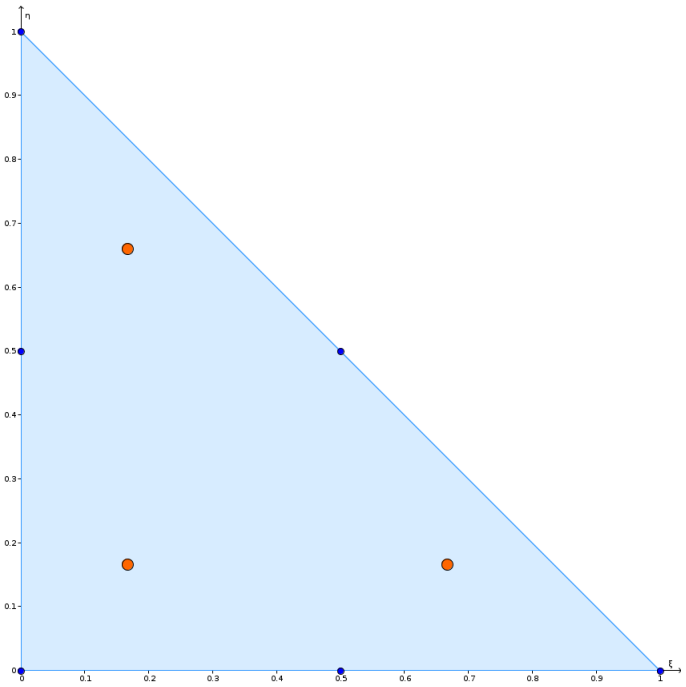


Figure 5.5: CPS6 element boundaries (blue) and GPs (orange)

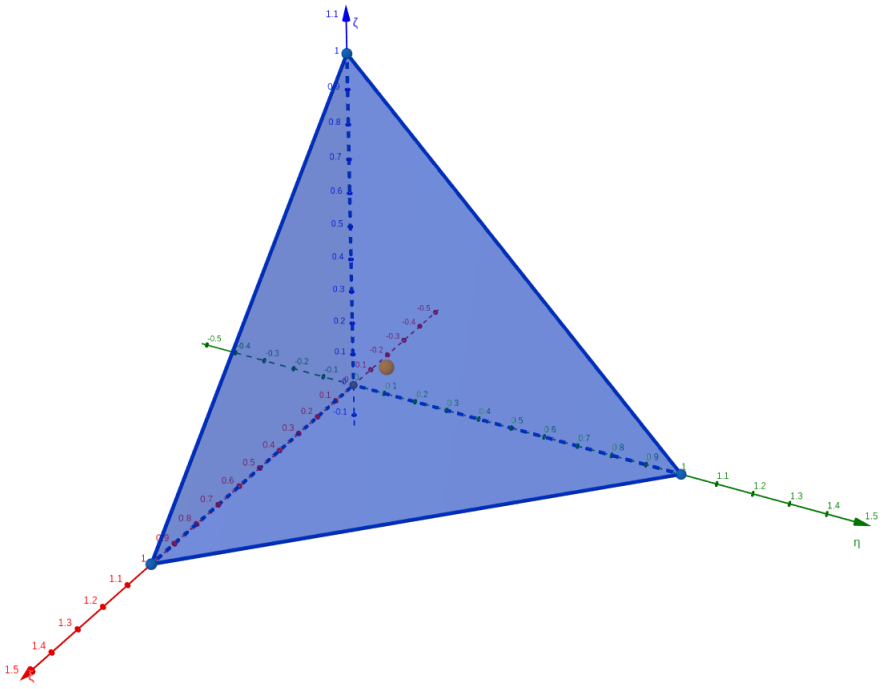


Figure 5.6: C3D4 element boundaries (blue) and GPs (orange)

creating surface subelements so the node topology between bulk and surface is matched.

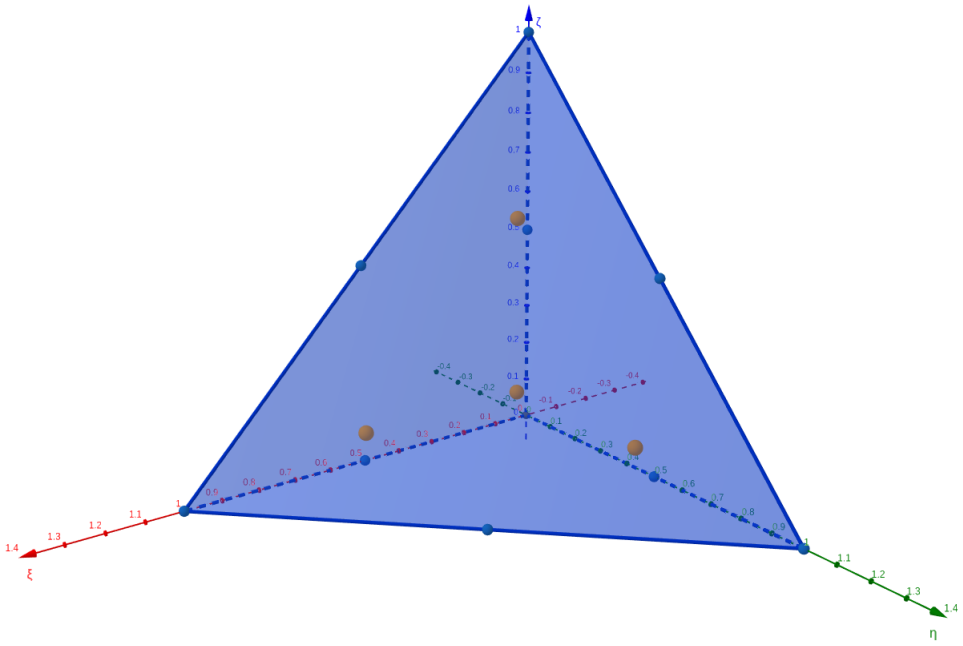


Figure 5.7: C3D10 element boundaries (blue) and GPs (orange)

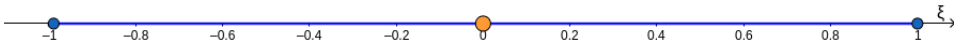


Figure 5.8: Linear line element boundaries (blue) and GPs (orange)

Figure 5.9: Quadratic line element boundaries (blue) and GPs (orange)

5.2 TEMPORAL DISCRETISATION

Temporal discretisation, solvers in MuPhiSim

All concepts related to the temporal discretisation are explained here. For further information, the references cited in this document are strongly recommended. These references have already been filtered and so are the minimum documentation required to understand the material. Knowledge of all concepts related to the spatial discretization is assumed (e.g., shape functions, numerical integration).

5.2.1 Introduction

MuPhiSim is mainly focused on non-linear problems in large deformation scenarios. Consequently, the program implements several algorithms and solvers to encompass such problems: explicit (particular case of the Newmark algorithm) and implicit (static and dynamic non-linear solvers, with alpha-generalised Newmark method for the latter) [2, 5]. All methods rely on the finite difference

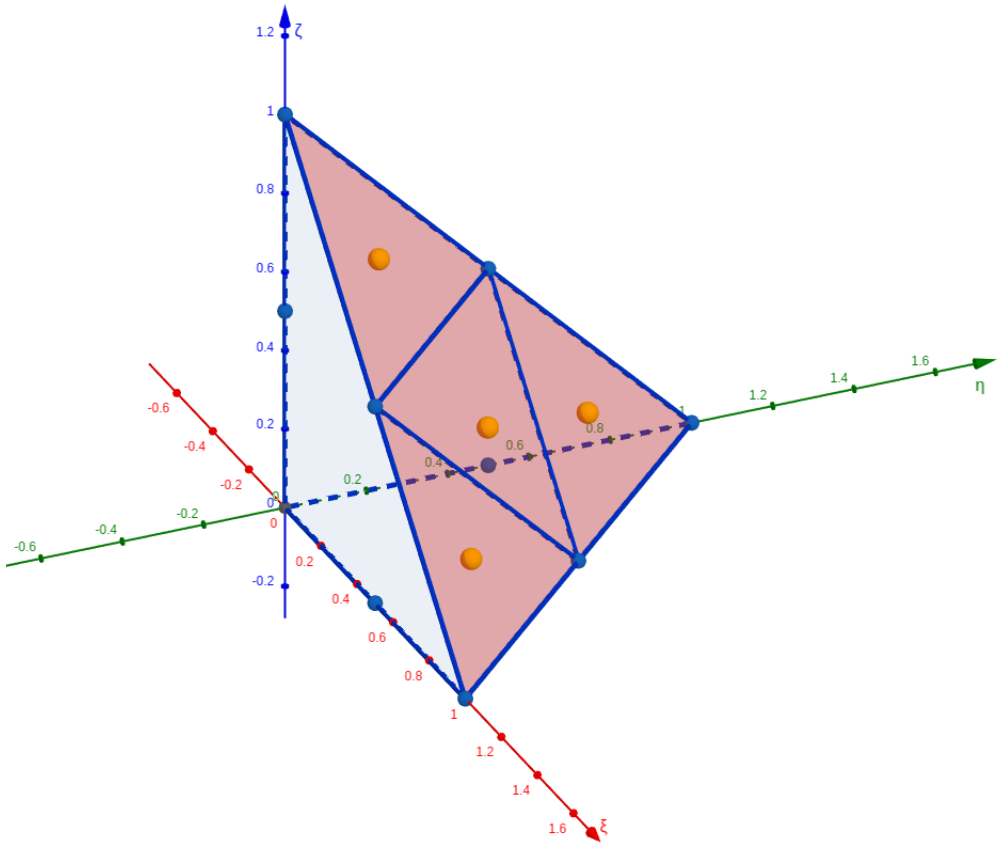


Figure 5.10: 4 linear triangles form a surface on a quadratic tetrahedron

method.
The governing equations stated in Section 5.1 can be written in a matrix-vector form as

$$\begin{aligned} \boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{f}^{int}(\boldsymbol{x}) &= \boldsymbol{f}^{ext} \\ M_{ab} &= \int_{\Omega_0} \rho_0 N_b N_a \, \mathrm{d}V \\ \boldsymbol{f}^{int}(\boldsymbol{x}) &= \int_{\Omega_0} \boldsymbol{P}_J N_{a,J} \, \mathrm{d}V \\ \boldsymbol{f}^{ext} &= \int_{\Omega_0} \rho_0 N_a \boldsymbol{b} \, \mathrm{d}V + \int_{\delta\Omega_{0n}} N_a \overline{\boldsymbol{T}} \mathrm{d}S \end{aligned} \tag{5.30}$$

where $\boldsymbol{M}$, $\boldsymbol{f}^{int}$, and $\boldsymbol{f}^{ext}$ are the mass matrix, the internal and external force vectors respectively, assembled from the elementary quantities derived in Section 5.1, and where $\boldsymbol{x}$ is redefined for this section as the vector of nodal coordinates. The integrals are approximated by numerical integration as follows:

$$M_{iakb} = \sum_q w_q \rho_0 \delta_{ik} N_a(\boldsymbol{\xi}_q) N_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{5.31}$$

$$f^{int}_{ia} = \sum_q w_q P_{iJ}(\boldsymbol{\xi}_q) N_{a,J}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{5.32}$$

$$f^{ext}_{ia} = \sum_p w_p T_i(\boldsymbol{\xi}_p) N_a(\boldsymbol{\xi}_p) J_0(\boldsymbol{\xi}_p) + \sum_q w_q \rho_0 b_i(\boldsymbol{\xi}_q) N_a(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{5.33}$$

5.2.2 Non-linear static problems

Using a non-linear static analysis, the effect of acceleration is neglected. For that kind of problem, non-linear static solutions can be a powerful tool and saves computational calculation time as a consequence of the trivial system of equations to solve.
As the accelerations can be neglected, Equation (5.30) can be written as

$$\boldsymbol{f}^{int}(\boldsymbol{x}) - \boldsymbol{f}^{ext} = 0 \tag{5.34}$$

Newton-Raphson iterations: Defining the following residual:

$$\boldsymbol{r}^k = \boldsymbol{f}^{int}(\boldsymbol{x}^k_{n+1}) - \boldsymbol{f}^{ext}_{n+1} \tag{5.35}$$

where superscript "k" indicates the iteration step ($k = 0$ is the initial guess before iterations). Using a first order Taylor expansion of $\boldsymbol{f}^{int}$ around $\boldsymbol{x}^k_{n+1}$, Equation (5.41) can be linearized into

$$\boldsymbol{f}^{int}(\boldsymbol{x}^k_{n+1}) + \frac{\partial \boldsymbol{f}^{int}}{\partial \boldsymbol{x}}(\boldsymbol{x}^k_{n+1})\Delta\boldsymbol{x}_{n+1} = \boldsymbol{f}^{ext}_{n+1} \tag{5.36}$$

and then the updated nodal deformation can be computed as

$$\Delta\boldsymbol{x}_{n+1} = -[\boldsymbol{K}^{eq}(\boldsymbol{x}^k_{n+1})]^{-1} \cdot \boldsymbol{r}^k \tag{5.37}$$

where $\boldsymbol{K}^{eq}(\boldsymbol{x}^k_{n+1})$ is given by

$$\boldsymbol{K}^{eq}(\boldsymbol{x}^k_{n+1}) = \boldsymbol{K}(\boldsymbol{x}^k_{n+1}) \tag{5.38}$$

where $\boldsymbol{K} = \frac{\partial \boldsymbol{f}^{int}}{\partial \boldsymbol{x}}$ is the stiffness matrix. The updated nodal displacements can then be updated:

$$\boldsymbol{x}^{k+1}_{n+1} = \boldsymbol{x}^k_{n+1} + \Delta\boldsymbol{x}_{n+1} \tag{5.39}$$

If the norm of the residual is larger than a given tolerance ϵ_{tol}, another iteration can be made, otherwise the iterations can be stopped. The algorithm for implicit Newmark time integration scheme is presented in Table 5.1.

5.2.3 Dynamic problems: Newmark algorithms

5.2.3.1 Traditional Newmark scheme

In order to solve Equation (5.30) the Newmark time integration scheme is defined as follows:

$$\boldsymbol{M} \cdot \ddot{\boldsymbol{x}}_{n+1} + \boldsymbol{f}^{int}(\boldsymbol{x}_{n+1}) = \boldsymbol{f}^{ext}_{n+1} \tag{5.40}$$

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta t\dot{\boldsymbol{x}}_n + \Delta t^2 \left(\left(\frac{1}{2} - \beta \right) \ddot{\boldsymbol{x}}_n + \beta \ddot{\boldsymbol{x}}_{n+1} \right) \tag{5.41}$$

$$\dot{\boldsymbol{x}}_{n+1} = \dot{\boldsymbol{x}}_n + \Delta t \left((1-\gamma)\, \ddot{\boldsymbol{x}}_n + \gamma \ddot{\boldsymbol{x}}_{n+1} \right) \tag{5.42}$$

where subscripts "n" and "$n+1$" indicate that the quantities are evaluated at time t_n and t_{n+1}, respectively, and where $\Delta t = t_{n+1} - t_n$ is the time step. Parameters $0 \leq \beta \leq 1/2$ and $0 \leq \gamma \leq 1$ set the characteristics of the Newmark scheme and can be tuned to manage the stability and accuracy of the solution.
As $\boldsymbol{f}^{int}$ is often formulated as a function of the displacement vector $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{X}$ (e.g., $\boldsymbol{f}^{int} = \boldsymbol{K} \cdot \boldsymbol{u}$), and noting that $\dot{\boldsymbol{u}} = \dot{\boldsymbol{x}}$ and $\ddot{\boldsymbol{u}} = \ddot{\boldsymbol{x}}$, it is sometime more convenient to solve this system in $\boldsymbol{u}$ instead of $\boldsymbol{x}$.

Table 5.1: Algorithm for the static non-linear time integration scheme with Newton-Raphson iteration.

1.	Get $\boldsymbol{x}_n$ from the previous time step and set it as the initial guess for the new time step.
2.	Compute external force $\boldsymbol{f}_{n+1}^{ext}$ using (5.33).
3.	Iteration step $k+1$: evaluate deformation gradient $\boldsymbol{F}(\boldsymbol{x}_{n+1}^k)$ at Gauss points.
4.	Get the stress at Gauss points $\boldsymbol{P}(\boldsymbol{F}(\boldsymbol{x}_{n+1}^k))$ from material constitutive model.
5.	Calculate internal force $\boldsymbol{f}_{n+1}^{k\,int}$ from $\boldsymbol{P}_{n+1}^k$ using (7.19).
6.	Compute the residual $\boldsymbol{r}^k$ from (5.35), and check: a. if $\|\boldsymbol{r}^k\| \leq \epsilon_{\text{tol}}$ then exit. Solution $\boldsymbol{x}_{n+1}$. b. if $\|\boldsymbol{r}^k\| > \epsilon_{\text{tol}}$ then continue.
7.	Get the stress tangent moduli $\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{F}}(\boldsymbol{x}_{n+1}^k)$ from material constitutive model, and compute $\boldsymbol{K}(\boldsymbol{x}_{n+1}^k)$ and thus $\boldsymbol{K}_{n+1}^{k\,eq}$.
10.	Compute $\Delta\boldsymbol{x}_{n+1}$ from (5.37), and update $\boldsymbol{x}_{n+1}^{k+1}$ from (5.39).
11.	Return to step 3 with $k = k+1$.

5.2.3.2 Explicit Newmark integration scheme

If $\beta = 0$, the generalized Newmark scheme (Equations (7.32)–(5.42)) becomes explicit. Indeed, for any given $\boldsymbol{x}_n$, $\dot{\boldsymbol{x}}_n$, and $\ddot{\boldsymbol{x}}_n$ at time t_n, the solution of $\boldsymbol{x}_{n+1}$ is obtained from Equation (5.41), then $\ddot{\boldsymbol{x}}_{n+1}$ from Equation (7.32), and finally $\dot{\boldsymbol{x}}_{n+1}$ is obtained from Equation (5.42). γ must be larger or equal to $1/2$ for a stable scheme. When $\gamma = 1/2$ is chosen, this scheme is second-order accurate and is called a central difference scheme. This scheme is conditionally stable: Δt needs to be smaller than a critical time step Δt_c, determined as:

$$\Delta t_c = \min_e \left(\frac{l_e}{C_e} \right) \tag{5.43}$$

where for each element e, l_e is its characteristic size and C_e is the dilatational velocity of sound in the material (provided by the constitutive law). In the case of linear elasticity, $C_e = \sqrt{\frac{\lambda+2\mu}{\rho}}$ where ρ, λ and μ are the density of the element and its Lamé constants.
Note that in this case, an alternative central difference scheme considering a mid-point velocity vector $\dot{\boldsymbol{x}}_{n-1/2} = \dot{\boldsymbol{x}}_n - \Delta t \ddot{\boldsymbol{x}}_n/2$ is often used, and the variables $\boldsymbol{x}_{n+1}$, $\dot{\boldsymbol{x}}_{n+1/2}$, and $\ddot{\boldsymbol{x}}_{n+1}$ are given by

$$\dot{\boldsymbol{x}}_{n+1/2} = \dot{\boldsymbol{x}}_{n-1/2} + \Delta t \ddot{\boldsymbol{x}}_n \tag{5.44}$$

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta t \dot{\boldsymbol{x}}_{n+1/2} \tag{5.45}$$

$$\ddot{\boldsymbol{x}}_{n+1} = \boldsymbol{M}^{-1} \cdot \left(\boldsymbol{f}_{n+1}^{ext} - \boldsymbol{f}^{int}(\boldsymbol{x}_{n+1}) \right) \tag{5.46}$$

For an easier implementation, the variable $\boldsymbol{x}_{n+1}$ can be directly computed as

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta t \dot{\boldsymbol{x}}_n + \frac{\Delta t^2}{2} \ddot{\boldsymbol{x}}_n \tag{5.47}$$

In order to avoid inverting the mass matrix $\boldsymbol{M}$, the matrix can be lumped, i.e., diagonalized. This can be done by using a nodal quadrature (the quadrature points $\boldsymbol{\xi}_q$ are not taken at the Gauss points, but at the nodes). In this case, the matrix is naturally diagonal. Another method is the

row (or column) method where the sum all the elements on each row (or column) is put on the corresponding diagonal. In MuPhiSim, for linear elements the mass matrix is lumped by performing a row sum and assigning it to the digonals, i.e. $M_{ij}^{lump} = \delta ij \sum_k M_{ik}$. For quadratic elements, the lumping scheme assumes the elements are regular (regular triangle and tetrahedron), and assigns a proportion of the elemental mass is dictated by the Voronoi diagram, so the elemental mass matrix can be defined as $M_{ij}^e = \delta_{ij} f_i \rho^e V^e$, where f_i is the fraction of the elemental mass assigned to node i.
Once the mass matrix has been lumped, Equation (5.46) can be solved node by node.

5.2.3.3 Implicit Newmark integration scheme

A value of $\beta > 0$ leads to an implicit scheme, since Equations (7.32)–(5.42) now have to be solved simultaneously using an iterative method such as the Newton-Raphson. For the scheme to be (unconditionally, i.e., independently of Δt) stable, $\beta \geq \frac{(\gamma+1/2)^2}{4}$ and $\gamma \geq 1/2$. Again, for $\gamma = 1/2$ and $\beta = 1/4$, the scheme is second-order accurate. Finally, for $\beta = \frac{(\gamma+1/2)^2}{4}$, the low frequency (physical) modes are shown to be best retained.

Newton-Raphson iterations: To begin the Newton-Raphson iteration, the initial guess for nodal accelerations is set to zero,

$$\ddot{\boldsymbol{x}}_{n+1}^{k=0} = \mathbf{0} \tag{5.48}$$

where superscript "k" indicates the iteration step ($k = 0$ is the initial guess before iterations). The initial value of the nodal positions and velocities are then:

$$\boldsymbol{x}_{n+1}^{k=0} = \boldsymbol{x}_n + \Delta t \dot{\boldsymbol{x}}_n + \Delta t^2 \left(\frac{1}{2} - \beta\right) \ddot{\boldsymbol{x}}_n \tag{5.49}$$

$$\dot{\boldsymbol{x}}_{n+1}^{k=0} = \dot{\boldsymbol{x}}_n + \Delta t \left(1 - \gamma\right) \ddot{\boldsymbol{x}}_n \tag{5.50}$$

Furthermore, during any subsequent iteration step $k+1$, the correction for nodal positions and velocities can be computed from Equations (5.41) and (5.42) as

$$\Delta \boldsymbol{x}_{n+1} = \boldsymbol{x}_{n+1}^{k+1} - \boldsymbol{x}_{n+1}^{k} = \beta \Delta t^2 \Delta \ddot{\boldsymbol{x}}_{n+1} \tag{5.51}$$

$$\Delta \dot{\boldsymbol{x}}_{n+1} = \dot{\boldsymbol{x}}_{n+1}^{k+1} - \dot{\boldsymbol{x}}_{n+1}^{k} = \gamma \Delta t \, \Delta \ddot{\boldsymbol{x}}_{n+1} \tag{5.52}$$

where $\Delta \ddot{\boldsymbol{x}}_{n+1} = \ddot{\boldsymbol{x}}_{n+1}^{k+1} - \ddot{\boldsymbol{x}}_{n+1}^{k}$.
Let $\boldsymbol{r}^k$ be the remainder (or residual) of the balance of linear momentum equation at iteration step k,

$$\boldsymbol{r}^k = \boldsymbol{M} \cdot \ddot{\boldsymbol{x}}_{n+1}^{k} + \boldsymbol{f}^{int}(\boldsymbol{x}_{n+1}^{k}) - \boldsymbol{f}_{n+1}^{ext} \tag{5.53}$$

Using a first order Taylor expansion of $\boldsymbol{f}^{int}$ around $\boldsymbol{x}_{n+1}^{k}$, Equation (5.41) can be linearized into

$$\boldsymbol{M} \cdot \{\ddot{\boldsymbol{x}}_{n+1}^{k} + \Delta \ddot{\boldsymbol{x}}_{n+1}\} + \boldsymbol{f}^{int}(\boldsymbol{x}_{n+1}^{k}) + \frac{\partial \boldsymbol{f}^{int}}{\partial \boldsymbol{x}}(\boldsymbol{x}_{n+1}^{k}) \Delta \boldsymbol{x}_{n+1} = \boldsymbol{f}_{n+1}^{ext} \tag{5.54}$$

Substituting $\Delta \ddot{\boldsymbol{x}}_{n+1}$ from Equation (5.51) into Equation (7.25), the updated nodal deformation can be computed as

$$\boldsymbol{x}_{n+1}^{k+1} = \boldsymbol{x}_{n+1}^{k} - [\boldsymbol{K}^{eq}(\boldsymbol{x}_{n+1}^{k})]^{-1} \cdot \boldsymbol{r}^k \tag{5.55}$$

where $\boldsymbol{K}^{eq}(\boldsymbol{x}_{n+1}^{k})$ is given by

$$\boldsymbol{K}^{eq}(\boldsymbol{x}_{n+1}^{k}) = \frac{1}{\beta \Delta t^2} \boldsymbol{M} + \boldsymbol{K}(\boldsymbol{x}_{n+1}^{k}) \tag{5.56}$$

and $\boldsymbol{K} = \frac{\partial \boldsymbol{f}^{int}}{\partial \boldsymbol{x}}$ is the stiffness matrix. The nodal accelerations and velocities can then be updated:

$$\ddot{\boldsymbol{x}}_{n+1}^{k+1} = \ddot{\boldsymbol{x}}_{n+1}^{k} + \frac{1}{\beta \Delta t^2} \Delta \boldsymbol{x}_{n+1} \quad \text{and} \quad \dot{\boldsymbol{x}}_{n+1}^{k+1} = \dot{\boldsymbol{x}}_{n+1}^{k} + \frac{\gamma}{\beta \Delta t} \Delta \boldsymbol{x}_{n+1} \tag{5.57}$$

If the norm of the residual is larger than a given tolerance ϵ_{tol}, another iteration can be made, otherwise the iterations can be stopped. The algorithm for implicit Newmark time integration scheme is presented in Table 5.2.

Table 5.2: Algorithm for the implicit Newmark time integration scheme with Newton-Raphson iteration.

1.	Choose β and γ and get $\boldsymbol{x}_n$, $\dot{\boldsymbol{x}}_n$, and $\ddot{\boldsymbol{x}}_n$ from the previous time step.
2.	Get time step Δt, compute external force $\boldsymbol{f}_{n+1}^{ext}$ using (5.33).
3.	Set initial guess $\ddot{\boldsymbol{x}}_{n+1}^{0} = \boldsymbol{0}$, $\boldsymbol{x}_{n+1}^{0}$ and $\dot{\boldsymbol{x}}_{n+1}^{0}$ using (5.49) and (5.50).
4.	Iteration step $k+1$: Calculate deformation interpolation $\boldsymbol{x}_h(\boldsymbol{x}_{n+1}^{k})$ using (5.7), and evaluate deformation gradient $\boldsymbol{F}(\boldsymbol{x}_{n+1}^{k})$ at Gauss points.
5.	Get the stress at Gauss points $\boldsymbol{P}(\boldsymbol{F}(\boldsymbol{x}_{n+1}^{k}))$ from material constitutive model.
6.	Calculate internal force $\boldsymbol{f}_{n+1}^{k\,int}$ from $\boldsymbol{P}_{n+1}^{k}$ using (7.19).
7.	Compute the residual $\boldsymbol{r}^k$ from (7.29), and check: a. if $\|\boldsymbol{r}^k\| \leq \epsilon_{\text{tol}}$ then exit. Solution $\boldsymbol{x}_{n+1}$, $\dot{\boldsymbol{x}}_{n+1}$, and $\ddot{\boldsymbol{x}}_{n+1}$ obtained! b. if $\|\boldsymbol{r}^k\| > \epsilon_{\text{tol}}$ then continue.
8.	Get the stress tangent moduli $\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{F}}(\boldsymbol{x}_{n+1}^{k})$ from material constitutive model, and compute $\boldsymbol{K}(\boldsymbol{x}_{n+1}^{k})$ using (5.19).
9.	Construct $\boldsymbol{K}_{n+1}^{k\,eq}$ using (7.28).
10.	Compute updates $\boldsymbol{x}_{n+1}^{k+1}$ from (7.27), and $\ddot{\boldsymbol{x}}_{n+1}^{k+1}$ and $\dot{\boldsymbol{x}}_{n+1}^{k+1}$ from (5.57).
11.	Return to step 4 with $k = k+1$.

5.2.3.4 Alpha-generalized Newmark scheme

Although the Newmark's algorithm appears to be a simple and efficient integration method, the possible numerical instabilities and numerical dissipations can make it no longer second-order accurate. An elegant, and widely used, way of including damping in the Newmark method without degrading the order of accuracy is the *Alpha-generalized Newmark scheme*.
Numerical damping (e.g., $\gamma > 1/2$) is often required in order to filter higher frequency responses. However, the application of numerical damping is typically obtained at the expense of the accuracy of the scheme. To avoid this drawback, an alpha-generalized Newmark scheme is sometime introduced by modifying Equation (7.32) as follows:

$$\begin{aligned}(1-\alpha_m)\boldsymbol{M}\cdot\ddot{\boldsymbol{x}}_{n+1} + (1-\alpha_f)\left(\boldsymbol{f}^{int}(\boldsymbol{x}_{n+1}) - \boldsymbol{f}_{n+1}^{ext}\right) \\ + \alpha_m \boldsymbol{M}\cdot\ddot{\boldsymbol{x}}_{n} + \alpha_f\left(\boldsymbol{f}^{int}(\boldsymbol{x}_{n}) - \boldsymbol{f}_{n}^{ext}\right) = \boldsymbol{0}\end{aligned} \tag{5.58}$$

where α_m and α_f are scalars that define the characteristics of the alpha-generalized Newmark scheme. They can be seen as averaging paramenters that will affect the inertia and internal/external forces. We distinguish three cases:

1. $\alpha_m = \alpha_f = 0$, the Nemark's scheme is recovered

2. $\alpha_m = 0$ and $\alpha_f = \alpha$, the integration method corresponds to the Hilber-Hughes-Taylor α-method (HHT)

3. $\alpha_m \neq 0$ and $\alpha_f \neq 0$, the integration method is the original work proposed by Chung and Hulbert (1993) [2, 5]

Implementation

Applying the same procedure than the Newmark's method, the equilibrium equation Eq. (7.32) represents the balance momentum. Rewriting such equation as follows:

$$\begin{aligned}(1-\alpha_m)\boldsymbol{M}\cdot\ddot{\boldsymbol{x}}_{n+1} + (1-\alpha_f)\left(\boldsymbol{f}^{int}(\boldsymbol{x}_{n+1}) - \boldsymbol{f}^{ext}_{n+1}\right)\\ + \alpha_m\boldsymbol{M}\cdot\ddot{\boldsymbol{x}}_n + \alpha_f\left(\boldsymbol{f}^{int}(\boldsymbol{x}_n) - \boldsymbol{f}^{ext}_n\right) = \boldsymbol{0}\end{aligned} \tag{5.59}$$

and using again a first order Taylor expansion of $\boldsymbol{f}^{int}$ around $\boldsymbol{x}^k_{n+1}$:

$$\begin{aligned}(1-\alpha_m)\boldsymbol{M}\cdot(\ddot{\boldsymbol{x}}_{n+1} + \boldsymbol{\Delta}\ddot{\boldsymbol{x}}_{n+1}) + \alpha_m\boldsymbol{M}\cdot\ddot{\boldsymbol{x}}_n\\ +(1-\alpha_f)\left(\boldsymbol{f}^{int}(\boldsymbol{x}_{n+1}) + \frac{\partial\boldsymbol{f}^{int}(x_{n+1})}{\partial\boldsymbol{x}}\boldsymbol{\Delta}\ddot{\boldsymbol{x}}_{n+1} - \boldsymbol{f}^{ext}_{n+1}\right)\\ +\alpha_f\left(\boldsymbol{f}^{int}(\boldsymbol{x}_n) - \boldsymbol{f}^{ext}_n\right) = \boldsymbol{0}\end{aligned} \tag{5.60}$$

and taking into account that $\boldsymbol{\Delta\ddot{x}_{n+1}} = \frac{1}{\beta\Delta t^2}\boldsymbol{\Delta x_{n+1}}$, finally:

$$\begin{aligned}\left(\frac{(1-\alpha_m)}{\beta\Delta t^2}\boldsymbol{M} + (1-\alpha_f)\frac{\partial\boldsymbol{f}^{int}(x_{n+1})}{\partial\boldsymbol{x}}\right)\boldsymbol{\Delta x}_{n+1}\\ = -(1-\alpha_m)\boldsymbol{M}\cdot\ddot{\boldsymbol{x}}_{n+1} - \alpha_m\boldsymbol{M}\cdot\ddot{\boldsymbol{x}}_n - (1-\alpha_f)\left(\boldsymbol{f}^{int}(\boldsymbol{x}_{n+1}) - \boldsymbol{f}^{ext}_{n+1}\right) - \alpha_f\left(\boldsymbol{f}^{int}(\boldsymbol{x}_n) - \boldsymbol{f}^{ext}_n\right)\end{aligned} \tag{5.61}$$

and finally:

$$\boldsymbol{K}^{eq}\cdot\boldsymbol{\Delta x}_{n+1} = \boldsymbol{r} \tag{5.62}$$

with

$$\boldsymbol{r} = \boldsymbol{M}\cdot((\alpha_m-1)\ddot{\boldsymbol{x}}_{n+1} - \alpha_m\ddot{\boldsymbol{x}}_n) + (\alpha_f-1)\boldsymbol{f}^{int}(\boldsymbol{x}_{n+1}) + (1-\alpha_f)\boldsymbol{f}^{ext}_{n+1} - \alpha_f\boldsymbol{f}^{int}(\boldsymbol{x}_n) + \alpha_f\boldsymbol{f}^{ext}_n \tag{5.63}$$

and

$$\boldsymbol{K^{eq}} = \frac{(1-\alpha_m)}{\beta\Delta t^2}\boldsymbol{M} + (1-\alpha_f)\frac{\partial\boldsymbol{f}^{int}(x_{n+1})}{\partial\boldsymbol{x}} \tag{5.64}$$

and finally the increment is calculated as follows:

$$\boldsymbol{\Delta x}_{n+1} = (\boldsymbol{K}^{eq})^{-1}\cdot\boldsymbol{r} \tag{5.65}$$

Substituting the residual of the Newmark's scheme by Eq.(5.63) and the equivalent stiffness matrix by Eq.(5.64), the time stepping is modified, while the iteration steps remain the same than the original method.

CHAPTER 6

Multiphysics framework

MuPhiSim provides a general framework for multi-physics simulations. The strong forms of the coupling problems are provided

- For mechanical part

$$\rho_0 \ddot{\mathbf{u}} - \mathbf{P} \cdot \boldsymbol{\nabla}_0 - \mathbf{B}_0 = \mathbf{0} \tag{6.1}$$
$$\mathbf{P} \cdot \mathbf{N} = \mathbf{T}_0 \text{ on } \Gamma_0^N \tag{6.2}$$
$$\mathbf{u} = \mathbf{u}_0 \text{ on } \Gamma_0^D \tag{6.3}$$

- For N extra-field parts

$$w_i + \mathbf{q}_i \cdot \boldsymbol{\nabla}_0 - \mathbf{r}_i = \mathbf{0} \text{ with } i = 0, .., N-1 \tag{6.4}$$
$$\mathbf{q}_i \cdot \mathbf{N} = q_{i0} \text{ on } \Gamma_N^{\varphi_i} \tag{6.5}$$
$$\varphi_i = \varphi_{i0} \text{ on } \Gamma_D^{\varphi_i} \tag{6.6}$$

  For each extra-field, w_i is the field source, $\mathbf{q}_i$ is the extra-field flux, and r_i is the external source. The extra field source includes different parts such as

$$w_i = c_i \dot{\varphi}_i + w_i^i + w_i^{mech} + \sum_{j-0, j \neq i}^{N-1} w_i^j \tag{6.7}$$

  where c_i is the field density, w_i^i is its internal source, w_i^{mech} is the source from the mechanical loading, and w_i^j is the contribution of the jth extra-field into the field ith.

The full-coupling materials laws need to be provided and can be rewritten under the general forms

- For mechanical part

$$\mathbf{P} = \mathbf{P}\left(\mathbf{F}, \varphi_0, \ldots, \varphi_{N-1}; \mathbf{Z}\right)$$

- For the extra-fields parts

$$\mathbf{q}_i = \mathbf{q}_i\left(\boldsymbol{\nabla}_0 \varphi_0 \ \ldots, \boldsymbol{\nabla}_0 \varphi_{N-1}, \mathbf{F}, \varphi_0, \ldots, \varphi_{N-1}; \mathbf{Z}\right) \text{ for } i = 0, \ldots, N-1$$
$$w_i = w_i\left(\mathbf{F}, \varphi_0, \ldots, \varphi_{N-1}; \mathbf{Z}\right)$$

where $\mathbf{Z}$ is the vector including all internal variables. When uncoupled problems are considered, they becomes

- For mechanical part

$$\mathbf{P} = \mathbf{P}\left(\mathbf{F}; \mathbf{Z}^{mech}\right)$$

- For the extra-fields parts

$$\begin{aligned} \mathbf{q}_i &= \mathbf{q}_i\left(\boldsymbol{\nabla}_0 \varphi_i, \varphi_i; \mathbf{Z}^{\varphi_i}\right) \text{ for } i = 0, \ldots, N-1 \\ w_i &= w_i\left(\varphi_i, \mathbf{Z}^{\varphi_i}\right) \end{aligned}$$

In the following sections, different cases are considered by specifying the extra-field equations Eq. (6.4).

6.1 ELECTROPHYSIOLOGY

This section provides the formulation of electrophysiology coupled to mechanics and its implementation step by step for Implicit Static, Dynamic and Explicit solvers.

Although the implementation of the extra degree of freedom allows for simulating different physical behaviours in parallel with mechanics, we specify here constitutive equations and anisotropic conduction of the action potential (AP) to model electrophysiology according to the formulation proposed in [4]. The balance equation for the electrophysiology is defined by means of a non-local contribution associated to conduction ($\boldsymbol{\nabla}_{\boldsymbol{X}} \cdot [\boldsymbol{D}\boldsymbol{\nabla}_{\boldsymbol{X}}\Phi]$); and a local contribution that refers to the dynamics of ionic currents in the local volume (R_Φ):

$$\frac{\partial \Phi}{\partial t} - R_\Phi - \boldsymbol{\nabla}_{\boldsymbol{X}} \cdot [\boldsymbol{D}\boldsymbol{\nabla}_{\boldsymbol{X}}\Phi] = 0 \tag{6.8}$$

where Φ is the normalised voltage or normalised transmembrane potential, $\boldsymbol{\nabla}_{\boldsymbol{X}}$ is the material gradient with respect to the reference configuration, $\boldsymbol{D}$ is the conductivity tensor and R_Φ is the ionic current, both expressed in the reference configuration. The ionic current R_Φ is based here on the FitzHugh-Nagumo (FHN) model as:

$$R_\Phi = (\Phi + \Phi_{equi})(\Phi + \Phi_{equi} - a)(1 - \Phi - \Phi_{equi}) - (\Phi_{equi})(\Phi_{equi} - a)(1 - \Phi_{equi}) - r + I_{stim} \tag{6.9}$$

where a is an activation parameter, Φ_{equi} is an equilibrium potential, I_{stim} is a stimulus current and r is the recovery current. This recovery variable is defined to evolve according to:

$$\frac{\partial r}{\partial t} = \varepsilon(-r + b\Phi) \tag{6.10}$$

with ε being the time-scale difference and b an activation parameter.

Regarding the conduction term, this is defined by the spatial distribution of the transmembrane potential Φ and the conductivity tensor $\boldsymbol{D}$. This conductivity tensor can be expressed alternatively in both material Eq.(6.11) and spatial forms Eq.(6.12):

$$\boldsymbol{D} = D_{iso}\boldsymbol{C^{-1}} + D_{ani}\boldsymbol{a_o} \otimes \boldsymbol{a_o}/\lambda^2 \tag{6.11}$$

$$\boldsymbol{d} = d_{iso}\boldsymbol{I} + d_{ani}\hat{\boldsymbol{a}} \otimes \hat{\boldsymbol{a}} \tag{6.12}$$

The terms d_{iso} and d_{ani} correspond to an isotropic conductivity contribution, and to an anisotropic contribution for faster conduction along the current axonal direction $\hat{\boldsymbol{a}}$ respectively. Regarding the material form of $\boldsymbol{D}$, the isotropic and anisotropic contributions can be obtained by using the deformation Jacobian $J = \det \boldsymbol{F}$, where $\boldsymbol{F}$ is the deformation gradient between the initial and the deformed configuration, as $D_{iso} = Jd_{iso}$ and $D_{ani} = Jd_{ani}$, respectively. $\boldsymbol{a_o}$ is the axonal direction in the reference configuration, $\boldsymbol{C} = \boldsymbol{F^t F}$ is the right Cauchy-Green deformation tensor and $\lambda = \sqrt{\hat{\boldsymbol{a}} \cdot \hat{\boldsymbol{a}}}$ the axonal stretch. Note that Eqs.(6.11) and (6.12) can be modified by changing the terms $\hat{\boldsymbol{a}} \otimes \hat{\boldsymbol{a}}$ and $\boldsymbol{a_o} \otimes \boldsymbol{a_o}$ for the corresponding structure tensor to include axonal dispersion dependency (see [4] for more details).

6.1.1 Numerical integration

Once the constitutive equations are defined, the governing equations are discretised both spatially and temporally. For this, the strong and weak forms of the balance equations in their residual forms are given.

6.1.2 Strong form

For the sake of simplicity, the residual equations expressed in the reference configuration and in their strong form are given as:

$$Div\,(\boldsymbol{F} \cdot \boldsymbol{S}) + \boldsymbol{R_\varphi} = \boldsymbol{0} \tag{6.13}$$

$$\dot{\Phi} - R_\Phi - \boldsymbol{\nabla_X} \cdot [\boldsymbol{D}\boldsymbol{\nabla_X}\Phi] = 0 \tag{6.14}$$

where $\boldsymbol{R_\varphi}$ are the external mechanical forces and $\boldsymbol{S}$ is the second Piola-Kirchhoff stress tensor. Note that Eq.(6.13) does not incorporate the term $\rho\ddot{\boldsymbol{\varphi}}$, thus neglecting the inertial terms and, consequently, being only valid for the integration of the mechano-electrophysiological coupling in Implicit/Static methods. Hereafter, only the development of the weak form for the electrophysiological part is presented.

6.1.3 Weak form

The weak form of the electrophysiological balance can be derived from Eq.(6.14) by multiplying the equation by the scalar-valued test function $\delta\Phi$ and integrating over the reference volume Ω_o:

$$\int_{\Omega_o} \delta\Phi\dot{\Phi}dV - \int_{\Omega_o} \delta\Phi R_\Phi dV - \int_{\Omega_o} \delta\Phi\boldsymbol{\nabla_X} \cdot [\boldsymbol{D}\boldsymbol{\nabla_X}\Phi]dV = 0 \tag{6.15}$$

Applying interation by parts and Gauss's theorem on the conductivity term lead to the final weak form :

$$\int_{\Omega_o} \delta\Phi\dot{\Phi}dV - \int_{\Omega_o} \delta\Phi R_\Phi dV + \int_{\Omega_o} \boldsymbol{\nabla_X}\delta\Phi \boldsymbol{D}\boldsymbol{\nabla_X}\Phi dV = 0 \tag{6.16}$$

6.1.4 Spatial discretisation

Regarding the spatial discretisation, the test functions $\delta\Phi$ and trial functions Φ are discretised as:

$$\begin{aligned} \delta\Phi^e &= \sum_{a=1}^{n_{el}} N_a \delta\Phi_a^e \\ \Phi^e &= \sum_{a=1}^{n_{el}} N_a \Phi_a^e \end{aligned} \tag{6.17}$$

where n_{el} is the number of nodes per element and N_a are the shape functions.

Therefore, combining equation (6.17) with (6.16), the following expression for the electrophysiological residual is obtained:

$$\sum_e \int_{\Omega_0^e} N_a^e \dot{\Phi} \cdot \delta\Phi^b \,\mathrm{d}V - \sum_e \int_{\Omega_0^e} N_a^e R_\Phi \cdot \delta\Phi^b \,\mathrm{d}V + \sum_e \int_{\Omega_0^e} \boldsymbol{\nabla_X} N_a^e \boldsymbol{D}\boldsymbol{\nabla_X}\Phi \cdot \delta\Phi^b \,\mathrm{d}V = 0 \tag{6.18}$$

Since the above equation must be valid for all admissible $\delta\Phi^b$, and using the approximation of (7.18), the finite element problem generalised to the integration points q now reads: Find Φ such that, for all b (then replaced by a),

$$F_b^{\text{int}} = 0 \tag{6.19}$$

where the external and internal electrophysiological forces are defined for each node a as:

$$F_a^{\text{int}} = \sum_q w_q N_a(\boldsymbol{\xi}_q)\dot{\Phi}(\boldsymbol{\xi}_q)J_0(\boldsymbol{\xi}_q) - \sum_q w_q N_a(\boldsymbol{\xi}_q)R_\Phi J_0(\boldsymbol{\xi}_q) + \sum_q w_q N_{a,i}(\boldsymbol{\xi}_q)D_{ij}\Phi_{,j}(\boldsymbol{\xi}_q)J_0(\boldsymbol{\xi}_q) \tag{6.20}$$

where $\boldsymbol{\xi}_q$ is the coordinate vector of Gauss point q of element e in the isoparametric coordinate system, w_q is its corresponding weight, and J_0 is the Jacobian between the isoparametric coordinate system and the reference configuration.
More details about the index notation used in MuPhiSim for the implementation of these internal and external forces can be found in *Appendix*.

6.1.5 Temporal discretisation - Implicit Dynamic and Static solvers

For the temporal discretisation an implicit Euler backward integration scheme is used, defining the temporal rate of the transmembrane potential as:

$$\dot{\Phi}^{n+1} = \frac{\Phi^{n+1} - \Phi^n}{\Delta t} \tag{6.21}$$

where Δt is the time increment used in the integration process.

The temporal integration of the internal variable r, Eq.(6.10), is computed implicitly by:

$$\dot{r}^{n+1} = \frac{r^{n+1} - r^n}{\Delta t} \Longleftrightarrow r^{n+1} = \frac{r^n + \varepsilon b \Phi^{n+1} \Delta t}{1 + \varepsilon \Delta t} \tag{6.22}$$

In *MuPhiSim*, the recovery current r is not updated once Φ^{n+1}, and so the previous equation is computed with Φ^n and not Φ^{n+1}.

For the implicit formulation, the unknown distribution of the field Φ can be solved by an iterative Newton-Raphson algorithm as:

$$Res_a^{\Phi} = F_a^{\text{int}} \tag{6.23}$$

$$Res_a^{\Phi} + \sum_b K_{ab}^{\Phi\Phi} \Delta\Phi_b + \sum_b \sum_k K_{akb}^{\Phi u} \Delta u_{kb} = 0 \tag{6.24}$$

The stiffness matrices are computed as:

$$K_{ab}^{\Phi\Phi} = \frac{\partial Res_a^{\Phi}}{\partial \Phi_b} = \sum_q w_q \left(N_a(\boldsymbol{\xi}_q) \left(\frac{\partial \dot{\Phi}}{\partial \Phi}(\boldsymbol{\xi}_q) - \frac{\partial R_{\Phi}}{\partial \Phi}(\boldsymbol{\xi}_q) \right) N_b(\boldsymbol{\xi}_q) + N_{a,i} D_{ij} N_{b,j} \right) J_0(\boldsymbol{\xi}_q) \tag{6.25}$$

$$K_{akb}^{\Phi u} = \frac{\partial Res_a^{\Phi}}{\partial \xi_{kb}} = \sum_q w_q N_{a,i}(\boldsymbol{\xi}_q) \frac{\partial D_{ij}}{\partial \xi_{kb}} \Phi_{,j}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{6.26}$$

$$K_{iab}^{u\Phi} = \frac{\partial Res_{ia}^{u}}{\partial \Phi_b} = 0 \tag{6.27}$$

See section 5.19 for the computation of K_{iakb}^{uu}. The global stiffness matrix is defined as:

$$\boldsymbol{K}^{\text{total}} = \begin{pmatrix} K_{iakb}^{uu} & K_{iab}^{u\Phi} \\ K_{akb}^{\Phi u} & K_{ab}^{\Phi\Phi} \end{pmatrix} \tag{6.28}$$

More details about the derivation of the terms $\frac{\partial \dot{\Phi}}{\partial \Phi}$, $\frac{\partial R_{\Phi}}{\partial \Phi}$ and $\frac{\partial D_{ij}}{\partial x_{kb}}$ can be found in *Appendix*. In addition, more details about the index notation used in MuPhiSim for the implementation of these stiffness matrices can be found in *Appendix*.

6.1.6 Temporal discretisation - Explicit solver

For the temporal discretisation an explicit Euler backward integration scheme is used, defining the temporal rate of the transmembrane potential as:

$$\dot{\Phi}^n = \frac{\Phi^{n+1} - \Phi^n}{\Delta t} \tag{6.29}$$

where Δt is the time increment used in the integration process.

The temporal integration of the internal variable r, Eq.(6.10), is computed explicitly by:

$$\dot{r}^n = \frac{r^{n+1} - r^n}{\Delta t} \Longleftrightarrow r^{n+1} = r^n(1 - \varepsilon\Delta t) + \varepsilon b \Phi^n \Delta t \tag{6.30}$$

For the explicit formulation, the unknown distribution of the field Φ can be solved directly by solving equation (6.19) with:

$$F_b^{\text{int}} = F_b^{\text{int},n+1} - F_b^{\text{int},n} \tag{6.31}$$

since $F_b^{\text{int},n}$ and $F_b^{\text{int},n+1}$ can be computed with the variables of the previous time step as following:

$$F_a^{\text{int},n} = \sum_q w_q N_a(\boldsymbol{\xi}_q)\Phi^n(\boldsymbol{\xi}_q)\frac{1}{\Delta t}J_0(\boldsymbol{\xi}_q) + \sum_q w_q N_a(\boldsymbol{\xi}_q) R_\Phi J_0(\boldsymbol{\xi}_q) - \sum_q w_q N_{a,i}(\boldsymbol{\xi}_q) D_{ij}\Phi_{,j}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{6.32}$$

$$F_a^{\text{int},n+1} = \sum_q w_q N_a(\boldsymbol{\xi}_q)\Phi^{n+1}(\boldsymbol{\xi}_q)\frac{1}{\Delta t}J_0(\boldsymbol{\xi}_q) \tag{6.33}$$

The term $F_a^{\text{int},n+1}$ can be written as following:

$$F_a^{\text{int},n+1} = \sum_q w_q N_a(\boldsymbol{\xi}_q)\left(\sum_b N_b(\boldsymbol{\xi}_q)\Phi_b^{n+1}\right)\frac{1}{\Delta t}J_0(\boldsymbol{\xi}_q) \tag{6.34}$$

$$= \sum_b \frac{1}{\Delta t} M_{ab}^{\text{extra}} \Phi_b^{n+1} \tag{6.35}$$

where the extra mass matrix $\boldsymbol{M}^{\text{extra}}$ is defined in the same way than in (7.11) and can be lumped in $\boldsymbol{M}_{\text{lump}}^{\text{extra}}$:

$$M_{ab}^{\text{extra}} = \sum_q w_q N_a(\boldsymbol{\xi}_q) N_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{6.36}$$

Equation (6.18) is evaluated at time step n to compute Φ^{n+1} at each node a according to the equation (6.35) :

$$\Phi_a^{n+1} = \Delta t \frac{F_a^{\text{int},n}}{M_{\text{lump},a}^{\text{extra}}} \tag{6.37}$$

This scheme is conditionally stable, i.e., Δt needs to be smaller than a critical time step Δt_c, determined as:

$$\Delta t_c = \frac{{l_e}^2}{2\left\|\boldsymbol{D}\right\|_2} \tag{6.38}$$

where l_e is its characteristic size and $\left\|\boldsymbol{D}\right\|_2$ a norm of the conductivity tensor (e.g. $\left\|\boldsymbol{D}\right\|_2 = \sqrt{tr(\boldsymbol{D}^T\boldsymbol{D})}$).

6.1.7 Appendix

This appendix provides the derivation in index notation of some components of the stiffness matrices.

6.1.7.1 Derivation of $\frac{\partial \dot{\Phi}}{\partial \Phi}$

The term $\frac{\partial \dot{\Phi}}{\partial \Phi}$ in (6.25) can be derived from (6.21) as:

$$\frac{\partial \dot{\Phi}}{\partial \Phi} = \frac{1}{\Delta t} \tag{6.39}$$

6.1.7.2 Derivation of $\frac{\partial R_\Phi}{\partial \Phi}$

The term $\frac{\partial R_\Phi}{\partial \Phi}$ in (6.25) can be derived from (6.9) and (6.22) as:

$$\frac{\partial R_\Phi}{\partial \Phi} = -3\Phi^2 + 2\Phi(1 - 3\Phi_{equi} + a) - 3\Phi_{equi}^2 + 2\Phi_{equi} + 2\Phi_{equi}a - a - \frac{\varepsilon b \Delta t}{1 + \varepsilon \Delta t} \tag{6.40}$$

6.1.7.3 Derivation of $\frac{\partial D_{ij}}{\partial x_{kb}}$

The term $\frac{\partial D_{ij}}{\partial x_{kb}}$ of equation (6.26) can be computed as:

$$\frac{\partial D_{ij}}{\partial x_{kb}} = \frac{\partial D_{ij}}{\partial C_{lm}} \frac{\partial C_{lm}}{\partial F_{no}} \frac{\partial F_{no}}{\partial x_{kb}} \tag{6.41}$$

1. The term $\frac{\partial D_{ij}}{\partial C_{lm}}$ can be computed as:

$$\begin{aligned}
\frac{\partial D_{ij}}{\partial C_{lm}} =& D_{iso} \frac{\partial C_{ij}^{-1}}{\partial C_{lm}} + D_{ani} a_{oi} a_{oj} \frac{\partial tr(\boldsymbol{a_o} \otimes \boldsymbol{a_o C})^{-1}}{\partial C_{lm}} \\
\frac{\partial C_{ij}^{-1}}{\partial C_{lm}} =& -\frac{1}{2}\left(C_{il}^{-1} C_{mj}^{-1} + C_{im}^{-1} C_{lj}^{-1}\right) \\
\frac{\partial tr(\boldsymbol{a_o} \otimes \boldsymbol{a_o C})^{-1}}{\partial C_{lm}} =& -\frac{1}{\lambda^4} a_{oi} a_{oj} \frac{\partial a_{on} a_{op} C_{pn}}{\partial C_{lm}} = -\frac{1}{\lambda^4} a_{oi} a_{oj} a_{on} a_{op} \delta_{lp} \delta_{mn} = -\frac{1}{\lambda^4} a_{oi} a_{oj} a_{ol} a_{om}
\end{aligned} \tag{6.42}$$

   therefore:

$$\frac{\partial D_{ij}}{\partial C_{lm}} = -\frac{D_{iso}}{2}\left(C_{il}^{-1} C_{mj}^{-1} + C_{im}^{-1} C_{lj}^{-1}\right) - \frac{D_{ani}}{\lambda^4} a_{oi} a_{oj} a_{ol} a_{om} \tag{6.43}$$

   which can also be directly written:

$$\frac{\partial \boldsymbol{D}}{\partial \boldsymbol{C}} = -\frac{D_{iso}}{2}\left(\boldsymbol{C^{-1}} \overline{\otimes} \boldsymbol{C^{-1}} + \boldsymbol{C^{-1}} \underline{\otimes} \boldsymbol{C^{-1}}\right) - \frac{D_{ani}}{\lambda^4}\left(\boldsymbol{a_o} \otimes \boldsymbol{a_o}\right) \otimes \left(\boldsymbol{a_o} \otimes \boldsymbol{a_o}\right) \tag{6.44}$$

   where $[\circ \overline{\otimes} \bullet]_{ijkl} = [\circ]_{ik} [\bullet]_{jl}$ and $[\circ \underline{\otimes} \bullet]_{ijkl} = [\circ]_{il} [\bullet]_{jk}$.

2. The term $\frac{\partial C_{lm}}{\partial F_{no}}$ can be computed as:

$$\frac{\partial C_{lm}}{\partial F_{no}} = (\delta_{lo} F_{nm} + \delta_{mo} F_{nl}) \tag{6.45}$$

3. The term $\dfrac{\partial F_{no}}{\partial x_{kb}}$ can be computed as:

$$\frac{\partial F_{no}}{\partial x_{kb}} = N_{b,o}\delta_{kn} \tag{6.46}$$

Finally, combining equation (6.42) with (6.45) and (6.46), we can reach the following expression for the term $\dfrac{\partial D_{ij}}{\partial x_{kb}}$:

$$\frac{\partial D_{ij}}{\partial x_{kb}} = \left[-\frac{D_{iso}}{2}\left(C^{-1}_{il}C^{-1}_{mj} + C^{-1}_{im}C^{-1}_{lj}\right) - \frac{D_{ani}}{\lambda^4}a_{oi}a_{oj}a_{ol}a_{om}\right](F_{km}N_{b,l} + F_{kl}N_{b,m}) \tag{6.47}$$

6.2 UNCOUPLED HEAT TRANSFER ANALYSIS

This chapter gives the formulation of thermal analysis without coupling to mechanics and details its implementation step by step for Implicit Static, Dynamic and Explicit solvers.

Here the equation of heat conduction obtained from the balances of energy (First law of thermodynamics) and entropy (Second law of thermodynamics) are specified, and are formulated in the reference configuration as:

$$\rho_0\dot{U} = \boldsymbol{P} : \dot{\boldsymbol{F}} - \boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{q}_0 + r \tag{6.48}$$

$$\rho_0\dot{\eta} + \frac{1}{\theta}\boldsymbol{P} : \dot{\boldsymbol{F}} - \frac{1}{\theta}\rho_0\dot{U} - \frac{1}{\theta^2}\boldsymbol{q}_0 \cdot \boldsymbol{\nabla}_{\boldsymbol{X}}\theta \geq 0 \tag{6.49}$$

where θ is the temperature, η is the entropy, $\boldsymbol{\nabla}_{\boldsymbol{X}}$ is the material gradient with respect to the reference configuration, $\boldsymbol{q}_0$ is the heat flux vector, U is the specific internal energy and r is the heat source per unit volume. Due to the direction of heat flux vector $\boldsymbol{q}_0$ is always opposite to the one of temperature gradient $\boldsymbol{\nabla}_{\boldsymbol{X}}\theta$, and so the last term of the second law always satisfies:

$$-\boldsymbol{q}_0 \cdot \boldsymbol{\nabla}_{\boldsymbol{X}}\theta \geq 0 \tag{6.50}$$

and; therefore, it can be neglected. Thus, combining Eq. (6.89) and (6.90), the equation of heat conduction is:

$$\rho_0\dot{\eta} + \frac{1}{\theta}\boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{q}_0 - \frac{1}{\theta}r \geq 0 \tag{6.51}$$

In the case of a conservative process, Eq. (6.92) can be expressed as an equality:

$$\rho_0\dot{\eta} + \frac{1}{\theta}\boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{q}_0 - \frac{1}{\theta}r = 0 \tag{6.52}$$

Considering a free energy density defined as $\psi = \psi(\boldsymbol{F}, \theta)$, the entropy is defined as the negative derivative with respect to the temperature as:

$$\eta = -\frac{\partial\psi}{\partial\theta} \tag{6.53}$$

In the same way, the total time derivative of entropy density is obtained as:

$$\dot{\eta} = -\frac{\partial^2\psi}{\partial\theta^2}\dot{\theta} \tag{6.54}$$

Then, the final form of the heat equation can be formulated as (without considering thermal-mechanical coupling):

$$\rho_0 C\dot{\theta} + \boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{q}_0 - r = 0 \tag{6.55}$$

where $C = -\theta\frac{\partial^2\psi}{\partial\theta^2}$ is the heat capacity at constant volume.

6.2.1 Boundary conditions

The Neumann and Dirichlet boundary conditions, $\bar{q}$ on $\partial\Omega_n$ and $\bar{\theta}$ on $\partial\Omega_d$, respectively, are imposed in the reference configuration by:

$$\begin{cases} \boldsymbol{q}_0 \cdot \boldsymbol{N} = -\bar{q},\ \forall \boldsymbol{X} \in \partial\Omega_n \\ \theta = \bar{\theta},\ \forall \boldsymbol{X} \in \partial\Omega_d \end{cases} \tag{6.56}$$

where $\boldsymbol{N}$ is the normal to the boundary in the reference configuration.
Additionally, other BC such as surface convection and radiation are implemented:

$$\begin{cases} \bar{q}_c = h(\theta - \theta^0),\ \forall \boldsymbol{X} \in \partial\Omega_n \\ \bar{q}_r = A\left(\theta^4 - \theta^{0^4}\right),\ \forall \boldsymbol{X} \in \partial\Omega_n \end{cases} \tag{6.57}$$

where h is the film coefficient (this has been considered temperature-independent), A is the radiation constant (emissivity times the Stefan-Boltzmann constant) and θ^0 is the sink temperature. It is emphasised that, as $\boldsymbol{N}$ is the outward normal to the surface, the sign in the case of convection and radiation is positive.

6.2.2 Weak form of heat equation

The weak form of the heat equation can be derived from Eq. (6.96) by multiplying by the scalar-valued test function $\delta\theta$, and integrating over the reference volume Ω_o:

$$\int_{\Omega_0} \rho_0 C\dot{\theta}\delta\theta \,\mathrm{d}V + \int_{\Omega_0} \boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{q}_0 \delta\theta \,\mathrm{d}V - \int_{\Omega_0} r\delta\theta \,\mathrm{d}V = 0 \tag{6.58}$$

Integrating by parts and using Green's theorem, the above expression can be rewritten as

$$\int_{\Omega_0} \rho_0 C\dot{\theta}\delta\theta \,\mathrm{d}V + \int_{\partial\Omega_0} (\boldsymbol{q}_0 \cdot \boldsymbol{N})\delta\theta \,\mathrm{d}S - \int_{\Omega_0} \boldsymbol{q}_0 \boldsymbol{\nabla}_{\boldsymbol{X}} \delta\theta \,\mathrm{d}V - \int_{\Omega_0} r\delta\theta \,\mathrm{d}V = 0 \tag{6.59}$$

6.2.3 Spatial discretisation

Regarding the spatial discretisation, the test functions $\delta\theta$ and trial functions θ are discretised as:

$$\begin{aligned} \delta\theta^e &= \sum_{a=1}^{n_{el}} N_a \delta\theta_a^e \\ \theta^e &= \sum_{a=1}^{n_{el}} N_a \theta_a^e \end{aligned} \tag{6.60}$$

where n_{el} is the number of nodes per element and N_a are the shape functions.

Therefore, combining Eq. (6.100) with (6.101), we can reach the following expression for the heat residual:

$$\sum_e \int_{\Omega_0^e} \rho_0 C N_b^e \dot{\theta}\delta\theta^b \,\mathrm{d}V - \sum_e \int_{\Omega_n^e} \bar{q} N_b^e \delta\theta^b \,\mathrm{d}S - \sum_e \int_{\Omega_0^e} \boldsymbol{q}_0 \boldsymbol{\nabla}_{\boldsymbol{X}} N_b^e \delta\theta^b \,\mathrm{d}V - \sum_e \int_{\Omega_0^e} r N_b^e \delta\theta^b \,\mathrm{d}V = 0 \tag{6.61}$$

In addition, the heat flux vector is assumed to be given by Fourier's law:

$$\boldsymbol{q}_0 = -\boldsymbol{k}_0 \nabla_X \theta \tag{6.62}$$

where $\boldsymbol{k}_0 = J\boldsymbol{F}^{-1}\boldsymbol{k}\boldsymbol{F}^{-T}$ is the material's conductivity. If $\boldsymbol{k}$ is an isotropic tensor, i.e. the material is thermally isotropic, $\boldsymbol{k} = k\boldsymbol{I}$ and conductivity tensor is expressed alternatively as:

$$\boldsymbol{k}_0 = Jk\boldsymbol{C}^{-1} \tag{6.63}$$

Finally, Eq.(6.61) is expressed as:

$$\sum_e \int_{\Omega_0^e} \rho_0 C N_b^e \dot{\theta} \delta\theta^b \,\mathrm{d}V + \sum_e \int_{\Omega_0^e} \nabla_X \left[\boldsymbol{k}_0 \nabla_X \theta\right] N_b^e \delta\theta^b \,\mathrm{d}V = \sum_e \int_{\Omega_n^e} \bar{q} N_b^e \delta\theta^b \,\mathrm{d}S + \sum_e \int_{\Omega_0^e} r N_b^e \delta\theta^b \,\mathrm{d}V \tag{6.64}$$

Since the above equation must be valid for all admissible $\delta\theta^b$, and using the approximation of (7.18), the finite element problem generalised to the integration points q now reads: Find θ such that, for all b (then replaced by a),

$$F_b^{\text{int}} - F_b^{\text{ext}} = 0 \tag{6.65}$$

where the external and internal thermal forces are defined for each node a as:

$$F_a^{\text{int}} = \sum_q w_q \rho_0 C N_a(\boldsymbol{\xi}_q) \dot{\theta}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) + \sum_q w_q N_{a,i}(\boldsymbol{\xi}_q) k_{0ij} \theta_{,j} J_0(\boldsymbol{\xi}_q) \tag{6.66}$$

$$F_a^{\text{ext}} = \sum_q w_q r N_a(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) + \sum_p w_p \bar{q} N_a(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{6.67}$$

where $\boldsymbol{\xi}_q$ is the coordinate vector of Gauss point q of element e in the isoparametric coordinate system, w_q is its corresponding weight, and J_0 is the Jacobian between the isoparametric coordinate system and the reference configuration.

6.2.4 Temporal discretisation - Implicit Dynamic and Static solvers

For the temporal discretisation we use an implicit Euler backward integration scheme, defining the temporal rate of the temperature as:

$$\dot{\theta}^{n+1} = \frac{\theta^{n+1} - \theta^n}{\Delta t} \tag{6.68}$$

where Δt is the time increment used in the integration process.

For the implicit formulation, the unknown distribution of the field θ can be solved by an iterative Newton-Raphson algorithm as:

$$Res_a^\theta = F_a^{\text{int}} - F_a^{\text{ext}} \tag{6.69}$$

$$Res_a^\theta + \sum_b K_{ab}^{\theta\theta} \Delta\theta_b + \sum_b \sum_k K_{akb}^{\theta u} \Delta u_{kb} = 0 \tag{6.70}$$

The stiffness matrices are computed as:

$$K_{ab}^{\theta\theta} = \frac{\partial Res_a^\theta}{\partial \theta_b} = \sum_q w_q \rho_0 \left(\frac{\partial C}{\partial \theta} \dot{\theta}(\boldsymbol{\xi}_q) + C \frac{\partial \dot{\theta}}{\partial \theta}(\boldsymbol{\xi}_q) \right) N_a(\boldsymbol{\xi}_q) N_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) + \sum_q w_q N_{a,i}(\boldsymbol{\xi}_q) k_{0_{ij}} N_{b,j}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) + \sum_q w_q N_{a,i} \frac{\partial k_{0ij}}{\partial \theta} \theta_{,j} N_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) - \sum_p w_p \frac{\partial \bar{q}_c}{\partial \theta} N_a(\boldsymbol{\xi}_q) N_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) - \sum_p w_p \frac{\partial \bar{q}_r}{\partial \theta} N_a(\boldsymbol{\xi}_q) N_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{6.71}$$

$$K_{akb}^{\theta u} = \frac{\partial Res_a^\theta}{\partial x_{kb}} = \sum_q w_q N_{a,i}(\boldsymbol{\xi}_q) \frac{\partial k_{0_{ij}}}{\partial x_{kb}} \theta_{,j}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) - \sum_p w_p \frac{\partial \bar{q}}{\partial x_{kb}} N_a(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{6.72}$$

$$K_{iab}^{u\theta} = \frac{\partial Res_{ia}^u}{\partial \theta_b} = 0 \tag{6.73}$$

where $C(\theta) = C_0 + C_1(\theta - \theta_{ref1})$ and $k(\theta) = k_0 + k_1(\theta - \theta_{ref2})$ are considered linear dependents on temperature.
See section Eq. 5.19 for the computation of K_{iakb}^{uu}. The global stiffness matrix is defined as:

$$\boldsymbol{K}^{\text{total}} = \begin{pmatrix} K_{iakb}^{uu} & K_{iab}^{u\theta} \\ K_{akb}^{\theta u} & K_{ab}^{\theta\theta} \end{pmatrix} \tag{6.74}$$

The term $\dfrac{\partial \dot{\theta}}{\partial \theta}$ can be derived as:

$$\frac{\partial \dot{\theta}}{\partial \theta} = \frac{1}{\Delta t} \tag{6.75}$$

The term $\dfrac{\partial \bar{q}_c}{\partial \theta}$ can be derived as:

$$\frac{\partial \bar{q}_c}{\partial \theta} = h \tag{6.76}$$

The term $\dfrac{\partial \bar{q}_r}{\partial \theta}$ can be derived as:

$$\frac{\partial \bar{q}_r}{\partial \theta} = 4A\theta^3 \tag{6.77}$$

The terms $\dfrac{\partial k_{0_{iJ}}}{\partial x_{kb}}$, being $k_{0iJ} = JkC_{IJ}^{-1} = JkF_{iQ}^{-1}F_{jQ}^{-1}$ can be derived as:

$$\frac{\partial k_{0iJ}}{\partial x_{kb}} = \frac{\partial k_{0iJ}}{\partial F_{sT}} \frac{\partial F_{sT}}{\partial x_{kb}} = \left[kJ(F_{sT})^{-T} F_{iQ}^{-1} F_{jQ}^{-1} - kJ(F_{is}^{-1} F_{TQ}^{-1} F_{JQ}^{-1} + F_{iQ}^{-1} F_{Js}^{-1} F_{TQ}^{-1}) \right] N_{b,T}(\boldsymbol{\xi}_q) \delta_{ks} \tag{6.78}$$

Finally, the terms $\dfrac{\partial \bar{q}}{\partial x_{kb}}$, being $Q_{0I} = JF_{iJ}^{-1} q_j$ can be derived as:

$$\frac{\partial Q_{0I}}{\partial x_{kb}} = \frac{\partial Q_{0I}}{\partial F_{sT}} \frac{\partial F_{sT}}{\partial x_{kb}} = \left[J(F_{sT})^{-T} F_{iJ}^{-1} - JF_{iS}^{-1} F_{tJ}^{-1} \right] q_j N_{b,T}(\boldsymbol{\xi}_q) \delta_{ks} \tag{6.79}$$

6.2.5 Temporal discretisation - Explicit solver

For the temporal discretisation an explicit Euler backward integration scheme is used, defining the temporal rate of the temperature as:

$$\dot{\theta}^n = \frac{\theta^{n+1} - \theta^n}{\Delta t} \tag{6.80}$$

where Δt is the time increment used in the integration process.

For the explicit formulation, the unknown distribution of the field θ can be solved directly with Eq. (6.103) with:

$$F_b^{\text{int}} = F_b^{\text{int},n+1} - F_b^{\text{int},n} \tag{6.81}$$

since $F_b^{\text{int},n}$ and $F_b^{\text{int},n+1}$ can be computed with the variables of the previous time step as following:

$$F_a^{\text{int},n} = \sum_q w_q \rho_0 C N_a(\boldsymbol{\xi}_q) \theta^n(\boldsymbol{\xi}_q) \frac{1}{\Delta t} J_0(\boldsymbol{\xi}_q) - \sum_q w_q N_{a,i}(\boldsymbol{\xi}_q) k_{0ij} \theta_{,j} J_0(\boldsymbol{\xi}_q) \tag{6.82}$$

$$F_a^{\text{int},n+1} = \sum_q w_q \rho_0 C N_a(\boldsymbol{\xi}_q) \theta^{n+1}(\boldsymbol{\xi}_q) \frac{1}{\Delta t} J_0(\boldsymbol{\xi}_q) \tag{6.83}$$

The term $F_a^{\text{int},n+1}$ can be written as following:

$$\begin{aligned} F_a^{\text{int},n+1} &= \sum_q w_q \rho_0 C N_a(\boldsymbol{\xi}_q) \left(\sum_b N_b(\boldsymbol{\xi}_q) \theta_b^{n+1} \right) \frac{1}{\Delta t} J_0(\boldsymbol{\xi}_q) \\ &= \sum_b \frac{1}{\Delta t} M_{ab}^{\text{extra}} \theta_b^{n+1} \end{aligned} \tag{6.84}$$

where the extra mass matrix $\boldsymbol{M}^{\text{extra}}$ is defined in the same way than in (7.11) and can be lumped with the same reasons in $\boldsymbol{M}_{\text{lump}}^{\text{extra}}$:

$$M_{ab}^{\text{extra}} = \sum_q w_q \rho_0 C N_a(\boldsymbol{\xi}_q) N_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{6.85}$$

Eq. (6.102) is evaluated at time step n to compute θ^{n+1} at each node a according to the Eq. (6.117) :

$$\theta_a^{n+1} = \Delta t \frac{Res_a^\theta}{M_{\text{lump},a}^{\text{extra}}} \tag{6.86}$$

$$Res_a^\theta = F_a^{\text{int},n} + F_a^{\text{ext}} \tag{6.87}$$

This scheme is conditionally stable, i.e., Δt needs to be smaller than a critical time step Δt_c, determined as:

$$\Delta t_c = \frac{{l_e}^2}{2\alpha} \tag{6.88}$$

where l_e is its characteristic size and $\alpha = \frac{k}{\rho C}$.

6.3 FULLY COUPLED THERMAL-STRESS ANALYSIS

This chapter provides the formulation of thermal coupled to mechanics and its implementation step by step for Implicit Static, Dynamic and Explicit solvers.

Here the equation of heat conduction that is obtained from the balances of energy (First law of thermodynamics) and entropy (Second law of thermodynamics) are specified, which are formulated in the reference configuration as:

$$\rho_0 \dot{U} = \boldsymbol{P} : \dot{\boldsymbol{F}} - \boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{q}_0 + r \tag{6.89}$$

$$\rho_0 \dot{\eta} + \frac{1}{\theta} \boldsymbol{P} : \dot{\boldsymbol{F}} - \frac{1}{\theta} \rho_0 \dot{U} - \frac{1}{\theta^2} \boldsymbol{q}_0 \cdot \boldsymbol{\nabla}_{\boldsymbol{X}} \theta \geq 0 \tag{6.90}$$

where θ is the temperature, η is the entropy, $\boldsymbol{\nabla}_{\boldsymbol{X}}$ is the material gradient with respect to the reference configuration, $\boldsymbol{q}_0$ is the heat flux vector, U is the specific internal energy and r is the heat source. Due to the direction of heat flux vector $\boldsymbol{q}_0$ is always contrary to temperature gradient $\boldsymbol{\nabla}_{\boldsymbol{X}} \theta$, then the last term of the second law always satisfies:

$$-\boldsymbol{q}_0 \cdot \boldsymbol{\nabla}_{\boldsymbol{X}} \theta \geq 0 \tag{6.91}$$

and therefore can be neglected. Then, combining Eq. (6.89) and (6.90), we obtain the equation of heat conduction as:

$$\rho_0 \dot{\eta} + \frac{1}{\theta} \boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{q}_0 - \frac{1}{\theta} r \geq 0 \tag{6.92}$$

In the case of conservative process, Eq. (6.92) can be expressed as an equality:

$$\rho_0 \dot{\eta} + \frac{1}{\theta} \boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{q}_0 - \frac{1}{\theta} r = 0 \tag{6.93}$$

Considering a free energy density defined as $\psi = \psi(\boldsymbol{F}, \theta)$, the entropy is defined as the negative derivative with respect to the temperature as:

$$\eta = -\frac{\partial \psi}{\partial \theta} \tag{6.94}$$

In the same way, the total time derivative of entropy density is obtained as:

$$\dot{\eta} = -\frac{\partial^2 \psi}{\partial \theta^2} \dot{\theta} - \frac{\partial^2 \psi}{\partial \boldsymbol{F} \partial \theta} : \dot{\boldsymbol{F}} \tag{6.95}$$

Finally, the final form of the heat equation can be formulated as:

$$\rho_0 C \dot{\theta} + \theta \boldsymbol{a} : \dot{\boldsymbol{F}} + \boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{q}_0 - r = 0 \tag{6.96}$$

where $C = -\theta \frac{\partial^2 \psi}{\partial \theta^2}$ is the heat capacity at constant volume and $\boldsymbol{a} = -\frac{\partial^2 \psi}{\partial \boldsymbol{F} \partial \theta}$ (ψ defined in J/m^3) is a generalised coupling tensor.

6.3.1 Boundary conditions

The Neumann and Dirichlet boundary conditions, $\bar{\boldsymbol{q}}$ on $\partial\Omega_n$ and $\bar{\theta}$ on $\partial\Omega_d$, are imposed in the reference configuration by:

$$\begin{cases} \boldsymbol{q}_0 \cdot \boldsymbol{N} = -\bar{q},\ \forall \boldsymbol{X} \in \partial\Omega_n \\ \theta = \bar{\theta},\ \forall \boldsymbol{X} \in \partial\Omega_d \end{cases} \tag{6.97}$$

where $\boldsymbol{N}$ is the normal to the boundary in the reference configuration. Additionally, other BC such as surface convection and radiation are implemented:

$$\begin{cases} \bar{q}_c = h(\theta - \theta^0),\ \forall \boldsymbol{X} \in \partial\Omega_n \\ \bar{q}_r = A\left(\theta^4 - \theta^{04}\right),\ \forall \boldsymbol{X} \in \partial\Omega_n \end{cases} \tag{6.98}$$

where h is the film coefficient (this is considered temperature-independent), A is the radiation constant (emissivity times the Stefan-Boltzmann constant) and θ^0 is the sink temperature. As $\boldsymbol{N}$ is the outward normal to the surface, the sign in the case of convection and radiation is positive.

6.3.2 Weak form of heat equation

The weak form of the heat equation can be derived from Eq. (6.96) by multiplying by the scalar-valued test function $\delta\theta$ and integrating over the reference volume Ω_o:

$$\int_{\Omega_0} \rho_0 C\dot{\theta}\delta\theta \,\mathrm{d}V + \int_{\Omega_0} \theta\boldsymbol{a} : \dot{\boldsymbol{F}}\delta\theta \,\mathrm{d}V + \int_{\Omega_0} \boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{q}_0\delta\theta \,\mathrm{d}V - \int_{\Omega_0} r\delta\theta \,\mathrm{d}V = 0 \tag{6.99}$$

Integrating by parts and using Green's theorem, the above expression can be rewritten as

$$\int_{\Omega_0} \rho_0 C\dot{\theta}\delta\theta \,\mathrm{d}V + \int_{\Omega_0} \theta\boldsymbol{a} : \dot{\boldsymbol{F}}\delta\theta \,\mathrm{d}V + \int_{\partial\Omega_0} (\boldsymbol{q}_0 \cdot \boldsymbol{N})\delta\theta \,\mathrm{d}S - \int_{\Omega_0} \boldsymbol{q}_0\boldsymbol{\nabla}_{\boldsymbol{X}}\delta\theta \,\mathrm{d}V - \int_{\Omega_0} r\delta\theta \,\mathrm{d}V = 0 \tag{6.100}$$

6.3.3 Spatial discretisation

Regarding the spatial discretisation, the test functions $\delta\theta$ and trial functions θ are discretised as:

$$\begin{aligned} \delta\theta^e &= \sum_{a=1}^{n_{el}} N_a \delta\theta_a^e \\ \theta^e &= \sum_{a=1}^{n_{el}} N_a \theta_a^e \end{aligned} \tag{6.101}$$

where n_{el} is the number of nodes per element and N_a are the shape functions.

Therefore, combining Eq. (6.100) with (6.101), we can reach the following expression for the thermal residual:

$$\begin{aligned} \sum_e \int_{\Omega_0^e} \rho_0 C N_b^e \dot{\theta}\delta\theta^b \,\mathrm{d}V + \sum_e \int_{\Omega_0^e} N_b^e \theta\boldsymbol{a} : \dot{\boldsymbol{F}}\delta\theta^b \,\mathrm{d}V - \sum_e \int_{\Omega_n^e} \bar{q} N_b^e \delta\theta^b \,\mathrm{d}S \\ - \sum_e \int_{\Omega_0^e} \boldsymbol{q}_0\boldsymbol{\nabla}_{\boldsymbol{X}} N_b^e \delta\theta^b \,\mathrm{d}V - \sum_e \int_{\Omega_0^e} r N_b^e \delta\theta^b \,\mathrm{d}V = 0 \end{aligned} \tag{6.102}$$

Since the above equation must be valid for all admissible $\delta\theta^b$ and using the approximation of (7.18), the finite element problem generalised to the integration points q now reads: Find θ such that, for all b (then replaced by a),

$$F_b^{\text{int}} = 0 \tag{6.103}$$

where the external and internal thermal forces are defined for each node a as:

$$\begin{aligned} F_a^{\text{int}} = \sum_q w_q \rho_0 C N_a(\boldsymbol{\xi}_q)\dot{\theta}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) + \sum_q w_q N_a(\boldsymbol{\xi}_q)\theta(\boldsymbol{\xi}_q) a_{iJ}\dot{F}_{iJ} J_0(\boldsymbol{\xi}_q) \\ - \sum_q w_q q_{0i} N_{a,i}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \end{aligned} \tag{6.104}$$

$$F_a^{\text{ext}} = \sum_p w_p \bar{q} N_a(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) + \sum_q w_q r N_a(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{6.105}$$

where $\boldsymbol{\xi}_q$ is the coordinate vector of Gauss point q of element e in the isoparametric coordinate system, w_q is its corresponding weight, and J_0 is the Jacobian between the isoparametric coordinate system and the reference configuration.

6.3.4 Temporal discretisation - Implicit Dynamic and Static solvers

For the temporal discretisation an implicit Euler backward integration scheme is used, defining the temporal rate of the temperature as:

$$\dot{\theta}^{n+1} = \frac{\theta^{n+1} - \theta^n}{\Delta t} \tag{6.106}$$

where Δt is the time increment used in the integration process.

For the implicit formulation, the unknown distribution of the field θ can be solved by an iterative Newton-Raphson algorithm as:

$$Res_a^\theta = F_a^{\text{int}} - F_a^{\text{ext}} \tag{6.107}$$

$$Res_a^\theta + \sum_b K_{ab}^{\theta\theta}\Delta\theta_b + \sum_b \sum_k K_{akb}^{\theta u}\Delta u_{kb} = 0 \tag{6.108}$$

The stiffness matrices are computed as:

$$\begin{aligned} K_{ab}^{\theta\theta} = \frac{\partial Res_a^\theta}{\partial \theta_b} = \sum_q w_q \rho_0 \left(\frac{\partial C}{\partial \theta}\dot{\theta}(\boldsymbol{\xi}_q) + C\frac{\partial \dot{\theta}}{\partial \theta}(\boldsymbol{\xi}_q) \right) N_a(\boldsymbol{\xi}_q) N_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \\ + \sum_q w_q N_a(\boldsymbol{\xi}_q) N_b(\boldsymbol{\xi}_q) \left(a_{iJ} + \theta\frac{\partial a_{iJ}}{\partial \theta} \right) \dot{F}_{iJ} J_0(\boldsymbol{\xi}_q) - \sum_q w_q N_{a,i}(\boldsymbol{\xi}_q) \frac{\partial q_{0_i}}{\partial \theta} J_0(\boldsymbol{\xi}_q) \\ - \sum_p w_p \frac{\partial \bar{q}_c}{\partial \theta} N_a(\boldsymbol{\xi}_q) N_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) - \sum_p w_p \frac{\partial \bar{q}_r}{\partial \theta} N_a(\boldsymbol{\xi}_q) N_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \end{aligned} \tag{6.109}$$

$$K^{\theta u}_{akb} = \frac{\partial Res^{\theta}_{a}}{\partial x_{kb}} = \sum_{q} w_q N_a(\boldsymbol{\xi}_q)\theta \left(\dot{F}_{iJ}\frac{\partial a_{iJ}}{\partial x_{kb}} + a_{iJ}\frac{\partial \dot{F}_{iJ}}{\partial x_{kb}} \right) J_0(\boldsymbol{\xi}_q) - \sum_{q} w_q N_{a,_i}(\boldsymbol{\xi}_q)\frac{\partial q_{0_i}}{\partial x_{kb}} J_0(\boldsymbol{\xi}_q) \tag{6.110}$$

$$K^{u\theta}_{iab} = \frac{\partial Res^{u}_{ia}}{\partial \theta_b} = \sum_{q} w_q \frac{\partial P_{iJ}}{\partial \theta} N_{a,J}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{6.111}$$

See section Eq. 5.19 for the computation of K^{uu}_{iakb}. The global stiffness matrix is defined as:

$$\boldsymbol{K}^{\text{total}} = \begin{pmatrix} K^{uu}_{iakb} & K^{u\theta}_{iab} \\ K^{\theta u}_{akb} & K^{\theta\theta}_{ab} \end{pmatrix} \tag{6.112}$$

More details about the derivation of the terms $\frac{\partial C}{\partial \theta}$, $\frac{\partial \dot{\theta}}{\partial \theta}$,$\frac{\partial a_{ij}}{\partial \theta}$,$\frac{\partial q_{0i}}{\partial \theta}$, $\frac{\partial a_{ij}}{\partial x_{kb}}$, $\frac{\partial \dot{F}_{ij}}{\partial x_{kb}}$ and $\frac{\partial P_{iJ}}{\partial \theta}$ can be found in *Appendix*.

6.3.5 Temporal discretisation - Explicit solver

For the temporal discretisation an explicit Euler backward integration scheme is used, defining the temporal rate of the temperature as:

$$\dot{\theta}^n = \frac{\theta^{n+1} - \theta^n}{\Delta t} \tag{6.113}$$

where Δt is the time increment used in the integration process.

For the explicit formulation, the unknown distribution of the field θ can be solved directly by solving Eq. (6.103) with:

$$F^{\text{int}}_{b} = F^{\text{int},n+1}_{b} - F^{\text{int},n}_{b} \tag{6.114}$$

since $F^{\text{int},n}_{b}$ and $F^{\text{int},n+1}_{b}$ can be computed with the variables of the previous time step as following:

$$F^{\text{int},n}_{a} = \sum_{q} w_q \rho_0 C N_a(\boldsymbol{\xi}_q)\theta^n(\boldsymbol{\xi}_q)\frac{1}{\Delta t} J_0(\boldsymbol{\xi}_q) - \sum_{q} w_q N_a(\boldsymbol{\xi}_q)\theta(\boldsymbol{\xi}_q) a_{iJ}\dot{F}_{iJ} J_0(\boldsymbol{\xi}_q) + \sum_{q} w_q q_{0i} N_{a,_i}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{6.115}$$

$$F^{\text{int},n+1}_{a} = \sum_{q} w_q \rho_0 C N_a(\boldsymbol{\xi}_q)\theta^{n+1}(\boldsymbol{\xi}_q)\frac{1}{\Delta t} J_0(\boldsymbol{\xi}_q) \tag{6.116}$$

The term $F^{\text{int},n+1}_{a}$ can be written as following:

$$F^{\text{int},n+1}_{a} = \sum_{q} w_q \rho_0 C N_a(\boldsymbol{\xi}_q) \left(\sum_{b} N_b(\boldsymbol{\xi}_q)\theta^{n+1}_{b} \right) \frac{1}{\Delta t} J_0(\boldsymbol{\xi}_q) = \sum_{b} \frac{1}{\Delta t} M^{\text{extra}}_{ab} \theta^{n+1}_{b} \tag{6.117}$$

where the extra mass matrix $\boldsymbol{M}^{\text{extra}}$ is defined in the same way than in (7.11) and can be lumped with the same reasons in $\boldsymbol{M}^{\text{extra}}_{\text{lump}}$:

$$M^{\text{extra}}_{ab} = \sum_q w_q \rho_0 C N_a(\boldsymbol{\xi}_q) N_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{6.118}$$

Eq. (6.102) is evaluated at time step n to compute Φ^{n+1} at each node a according to the Eq. (6.117) :

$$\Phi^{n+1}_a = \Delta t \frac{Res^{\Phi}_a}{M^{\text{extra}}_{\text{lump},a}} \tag{6.119}$$

$$Res^{\Phi}_a = F^{\text{int},n}_a + F^{\text{ext}}_a \tag{6.120}$$

where here Res^{Φ}_a is physically not a residual.

This scheme is conditionally stable, i.e., Δt needs to be smaller than a critical time step Δt_c, determined as:

$$\Delta t_c = \frac{{l_e}^2}{2\alpha} \tag{6.121}$$

where l_e is its characteristic size and $\alpha = \frac{k}{\rho C}$.

6.3.6 Appendix

This appendix provides the derivation in index notation of some components of the stiffness matrices for the different thermomechanical constitutive models described below:

6.3.6.1 Derivation of $\dfrac{\partial C}{\partial \theta}$

A linear dependence of C has been implemented: Two different dependencies of C have been formulated:

$$C(\theta) = C_0 + C_1(\theta - \theta_{ref1}) \tag{6.122}$$

The term $\dfrac{\partial C}{\partial \theta}$ in Eq.(6.109) can be derived from Eq.(6.122) as:

$$\frac{\partial C}{\partial \theta} = C_1 \tag{6.123}$$

In case of temperature-independent heat capacity ($\tau_1 = 0$), the term in Eq.(6.109) is:

$$\frac{\partial C}{\partial \theta} = 0 \tag{6.124}$$

6.3.6.2 Derivation of $\dfrac{\partial \dot{\theta}}{\partial \theta}$

The term $\dfrac{\partial \dot{\theta}}{\partial \theta}$ in Eq.(6.109) can be derived from Eq.(6.106) as:

$$\frac{\partial \dot{\theta}}{\partial \theta} = \frac{1}{\Delta t} \tag{6.125}$$

6.3.6.3 Derivation of $\frac{\partial q_{0i}}{\partial \theta}$

Assuming the heat flux vector is given by Fourier's law:

$$\boldsymbol{q}_0 = -\boldsymbol{k}_0(F,\theta)\nabla_X \theta \tag{6.126}$$

where the conductivity is defined as $\boldsymbol{k}_0 = J\boldsymbol{F}^{-1}\boldsymbol{k}\boldsymbol{F}^{-T}$. In this formulation, k is taken to kbe linearly dependent on temperature:

$$k(\theta) = k_0 + k_1(\theta - \theta_{ref2}) \tag{6.127}$$

The term $\frac{\partial q_{0i}}{\partial \theta}$ of Eq. (6.109) can be computed as:

$$\frac{\partial q_{0i}}{\partial \theta} = -k_{0ij}N_{b,j}(\boldsymbol{\xi}_q) - \frac{\partial k_{0ij}}{\partial \theta}\theta_{,j}N_b(\boldsymbol{\xi}_q) \tag{6.128}$$

6.3.6.4 Derivation of $\frac{\partial q_{0_i}}{\partial x_{kb}}$

The term $\frac{\partial q_{0i}}{\partial x_{kb}}$, being $k_{0iJ} = JkC^{-1}_{IJ} = JkF^{-1}_{iQ}F^{-1}_{jQ}$, in Eq.(6.110) as:

$$\frac{\partial q_{0i}}{\partial x_{kb}} = \frac{\partial q_{0i}}{\partial F_{sT}}\frac{\partial F_{sT}}{\partial x_{kb}} = \left[kJ(F_{sT})^{-T}F^{-1}_{iQ}F^{-1}_{jQ} - kJ(F^{-1}_{is}F^{-1}_{TQ}F^{-1}_{JQ} + F^{-1}_{iQ}F^{-1}_{Js}F^{-1}_{TQ})\right]\theta_{,J}N_{b,T}(\boldsymbol{\xi}_q)\delta_{ks} \tag{6.129}$$

6.3.6.5 Derivation of $\frac{\partial \bar{q}_c}{\partial \theta}$

The term $\frac{\partial \bar{q}_c}{\partial \theta}$ can be derived as:

$$\frac{\partial \bar{q}_c}{\partial \theta} = h \tag{6.130}$$

6.3.6.6 Derivation of $\frac{\partial \bar{q}_r}{\partial \theta}$

The term $\frac{\partial \bar{q}_r}{\partial \theta}$ can be derived as:

$$\frac{\partial \bar{q}_r}{\partial \theta} = 4A\theta^3 \tag{6.131}$$

6.3.7 St Venant-Kirchhoff thermomechanical constitutive model (Temperature-independent material parameters)

This model is implemented as *St-Venant-Kirchhoff-ThermoMechanics*.

Assuming small deformation($\boldsymbol{P} \approx \boldsymbol{S} \approx \boldsymbol{\sigma}$, $\boldsymbol{E} \approx \boldsymbol{e} \approx \boldsymbol{\epsilon}$, $\rho \approx \rho_0$, $\nabla_{\vec{\boldsymbol{X}}} \approx \nabla_{\vec{\boldsymbol{x}}} \approx \nabla$) and small temperature changes, the problem separates the free energy density into:

$$\psi(\theta,\boldsymbol{\epsilon}) = \psi^m(\theta,\boldsymbol{\epsilon}) + \psi^c(\theta,\boldsymbol{\epsilon}) + \psi^t(\theta) \tag{6.132}$$

where $\psi^m(\theta,\boldsymbol{\epsilon})$ is a mechanical, $\psi^c(\theta,\boldsymbol{\epsilon})$ a coupling and $\psi^t(\theta)$ a thermal part.

For the particular constitutive model of St Venant-Kirchhof, $\psi^m(\theta, \boldsymbol{\epsilon})$ is expressed as:

$$\psi^m(\theta, \boldsymbol{\epsilon}) = \frac{1}{2}\lambda(\theta)\left[tr(\boldsymbol{E})\right]^2 + \mu(\theta)tr(\boldsymbol{E}^2) \tag{6.133}$$

Moreover, for small deformations, $\psi^c(\theta, \boldsymbol{\epsilon})$ can be defined as:

$$\psi^c(\theta, \boldsymbol{\epsilon}) = \mathbb{C}(\theta)\boldsymbol{\alpha}(\theta)\left(\theta - \theta_{ini}\right) : \boldsymbol{E} \tag{6.134}$$

where $\mathbb{C}(\theta)$ is the elastic stiffness tensor, $\boldsymbol{\alpha}(\theta)$is the thermal expansion tensor, which is equal to $\boldsymbol{\alpha}(\theta) = \alpha(\theta)\boldsymbol{I}$ for isotropic heat expansion, and θ_{ini} is the initial temperature at which the material is defined as undeformed in the absence of mechanical loads.
The stress follows as

$$S_{IJ} = \lambda(\theta)tr(E_{IJ})\delta_{IJ} + 2\mu(\theta)E_{IJ} - 3k(\theta)\alpha(\theta)[\theta - \theta_{ini}]\delta_{IJ} \tag{6.135}$$

For the sake of convenience, the thermomechanical coupling tensor is defined as $m_{IJ} = 3k(\theta)\alpha(\theta)\delta_{IJ}$.
Considering the elastic stiffness tensor and heat expansion as temperature independent, the same expression for large deformations is obtained. The coupling tensor a_{iJ} is defined as:

$$\begin{aligned} a_{iJ} &= -\frac{\partial^2\psi}{\partial F_{iJ}\partial\theta} = -\frac{\partial^2\psi}{\partial E_{PQ}\partial\theta}\frac{\partial E_{PQ}}{\partial F_{iJ}} = m_{PQ}\frac{1}{2}\left[\frac{\partial\left(F^TF\right)}{\partial F}\right]_{PQiJ} = m_{PQ}\frac{1}{2}\left[\frac{\partial\left(F_{rP}F_{rQ}\right)}{\partial F_{iJ}}\right] \\ &= m_{PQ}\frac{1}{2}\left[\delta_{ri}\delta_{PJ}F_{rQ} + F_{rP}\delta_{ri}\delta_{QJ}\right] = m_{PQ}\frac{1}{2}\left[\delta_{PJ}F_{iQ} + F_{iP}\delta_{QJ}\right] \end{aligned} \tag{6.136}$$

$^*\boldsymbol{e} \approx \boldsymbol{E} = \frac{1}{2}\left(\boldsymbol{C} - \boldsymbol{I}\right) = \frac{1}{2}\left(\boldsymbol{F}^T\boldsymbol{F} - \boldsymbol{I}\right)$
Once the stress tensor and generalised coupling have been defined, the rest of derivations in index notation of the components of the stiffness matrix are presented below. All these components that have been implemented in the corresponded constitutive model (marked in blue). However, some terms will common to all models and have been kept in "***Thermomechanics.cpp***" (marked in black).

6.3.7.1 Derivation of $\frac{\partial a_{iJ}}{\partial\theta}$

The term $\frac{\partial a_{iJ}}{\partial\theta}$ in Eq.(6.109) can be derived from Eq.(6.155) as:

$$\frac{\partial a_{iJ}}{\partial\theta} = \mathbf{0} \tag{6.137}$$

6.3.7.2 Derivation of $\frac{\partial a_{iJ}}{\partial x_{kb}}$

The term $\frac{\partial a_{iJ}}{\partial x_{kb}}$ in Eq.(6.110) can be derived from Eq.(6.155) as:

$$\begin{aligned} \frac{\partial a_{iJ}}{\partial x_{kb}} &= \frac{\partial a_{iJ}}{\partial F_{sT}}\frac{\partial F_{sT}}{\partial x_{kb}} = \frac{\partial\left[m_{PQ}\frac{1}{2}\left[\delta_{PJ}F_{iQ} + F_{iP}\delta_{QJ}\right]\right]}{\partial F_{sT}}N_{b,T}(\boldsymbol{\xi}_q)\delta_{ks} \\ &= m_{PQ}\frac{1}{2}\left[\delta_{PJ}\delta_{is}\delta_{QT} + \delta_{is}\delta_{PT}\delta_{QJ}\right]N_{b,T}(\boldsymbol{\xi}_q)\delta_{ks} \end{aligned} \tag{6.138}$$

6.3.7.3 Derivation of $\frac{\partial \dot{F}_{iJ}}{\partial x_{kb}}$

The term $\frac{\partial \dot{F}_{iJ}}{\partial x_{kb}}$ in Eq.(6.110) can be derived as:

$$\frac{\partial \dot{F}_{iJ}}{\partial x_{kb}} = \frac{\partial \dot{F}_{iJ}}{\partial F_{sT}} \frac{\partial F_{sT}}{\partial x_{kb}} = \frac{1}{\Delta t} \delta_{is} \delta_{JT} N_{b,T}(\boldsymbol{\xi}_q) \delta_{ks} = \frac{1}{\Delta t} \delta_{ik} N_{b,J} \tag{6.139}$$

6.3.7.4 Derivation of $\frac{\partial P_{iJ}}{\partial \theta}$

The term $\frac{\partial P_{iJ}}{\partial \theta}$ in Eq.(6.111) can be derived from Eq.(6.135) as:

$$\frac{\partial P_{iJ}}{\partial \theta} = -F_{iK} m_{KJ} N_b(\boldsymbol{\xi}_q) \tag{6.140}$$

6.3.8 St-Venant-Kirchhoff thermomechanical constitutive model (Temperature-dependent material parameters)

This model is implemented as **3DPrinting-St-Venant-Kirchhoff-Thermoelasticity**.

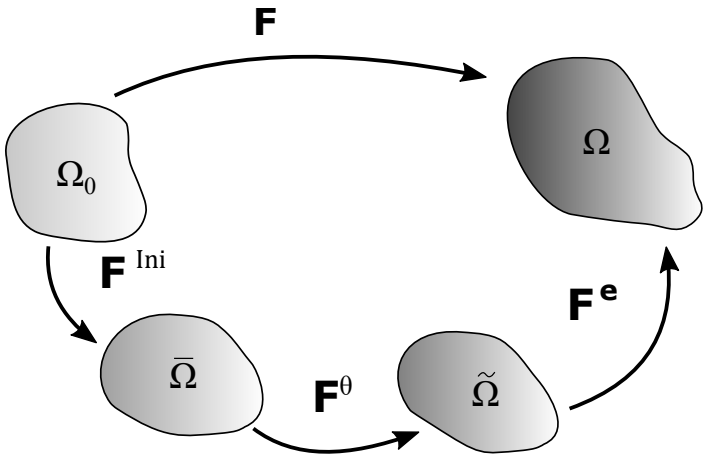


Figure 6.1: Kinematic

Considering thermoelasticity in finite deformation, the deformation gradient is split using the multiplicative decomposition such as $F_{iJ} = F^e{}_{i\tilde{k}} F^\theta{}_{\tilde{k}\bar{q}} F^{Ini}_{\bar{q}J}$, where $F^\theta_{\tilde{k}\bar{q}} = \upsilon^\theta \delta_{\tilde{k}\bar{q}}$and $F^{Ini}_{\bar{q}J}$ is the initial deformation defined as the deformation when an element turns to powder to liquid. In the same way, the Jacobian can be decomposed as $J = J^e J^\theta J^{Ini}$.

Considering elastic and thermal contributions, both Green strain tensor can be defined as:

$$\begin{aligned} \boldsymbol{E} &= \frac{1}{2}\left(\boldsymbol{F}^T\boldsymbol{F} - \boldsymbol{I}\right) \rightarrow E_{IJ} = \frac{1}{2}\left(F_{kI}F_{kJ} - \delta_{IJ}\right) \\ \boldsymbol{E}^e &= \frac{1}{2}\left(\boldsymbol{F}^{eT}\boldsymbol{F}^e - \boldsymbol{I}\right) \rightarrow E^e{}_{\tilde{s}\tilde{t}} = \frac{1}{2}\left(F^e{}_{k\tilde{s}} F^e{}_{k\tilde{t}} - \delta_{\tilde{s}\tilde{t}}\right) \end{aligned} \tag{6.141}$$

The elastic Green strain tensor can be equivalently rewritten in terms of the total deformation gradient as:

$$\begin{aligned}\boldsymbol{E}^e &= \frac{1}{2}\left(\boldsymbol{F}^{eT}\boldsymbol{F}^e - \boldsymbol{I}\right) = \frac{1}{2}\left(\frac{1}{\upsilon^{\theta^2}}\boldsymbol{F}^{Ini-T}\boldsymbol{F}^T\boldsymbol{F}\boldsymbol{F}^{Ini-1} - \boldsymbol{I}\right)\\ E^e{}_{\tilde{s}\tilde{t}} &= \frac{1}{2}\left(F^e{}_{k\tilde{s}}F^e{}_{k\tilde{t}} - \delta_{\tilde{s}\tilde{t}}\right) = \frac{1}{2}\left(\frac{1}{\upsilon^{\theta^2}}F^{Ini-1}_{I\tilde{s}}F_{kI}F_{kJ}F^{Ini-1}_{J\tilde{t}} - \delta_{\tilde{s}\tilde{t}}\right)\end{aligned} \tag{6.142}$$

The rate of total Green strain is given by:

$$\dot{\boldsymbol{E}} = \frac{1}{2}\left(\dot{\boldsymbol{F}}^T\boldsymbol{F} + \boldsymbol{F}^T\dot{\boldsymbol{F}}\right) \rightarrow \dot{E}_{IJ} = \frac{1}{2}\left(\dot{F}_{kI}F_{kJ} + F_{kI}\dot{F}_{kJ}\right) \tag{6.143}$$

and using the multiplicative decomposition, the elastic Green strain tensor can be written as:

$$\begin{aligned}\dot{\boldsymbol{E}}^e &= \frac{1}{2}\left(\frac{d}{dt}\left[\frac{1}{\upsilon^{\theta^2}}\right]\boldsymbol{F}^{Ini-T}\boldsymbol{F}^T\boldsymbol{F}\boldsymbol{F}^{Ini-1} + \frac{1}{\upsilon^{\theta^2}}\boldsymbol{F}^{Ini-T}\dot{\boldsymbol{F}}^T\boldsymbol{F}\boldsymbol{F}^{Ini-1} + \frac{1}{\upsilon^{\theta^2}}\boldsymbol{F}^{Ini-T}\boldsymbol{F}^T\dot{\boldsymbol{F}}\boldsymbol{F}^{Ini-1}\right)\\ &= \frac{1}{\upsilon^{\theta^2}}\boldsymbol{F}^{Ini-T}\left(-\frac{1}{\upsilon^\theta}\frac{d\upsilon^\theta}{d\theta}\dot{\theta}\boldsymbol{F}^T\boldsymbol{F} + \frac{1}{2}\dot{\boldsymbol{F}}^T\boldsymbol{F} + \frac{1}{2}\boldsymbol{F}^T\dot{\boldsymbol{F}}\right)\boldsymbol{F}^{Ini-1}\\ &= \frac{1}{\upsilon^{\theta^2}}\boldsymbol{F}^{Ini-T}\left(\dot{E} - \frac{1}{\upsilon^\theta}\frac{d\upsilon^\theta}{d\theta}\dot{\theta}\left(2\boldsymbol{E} + \boldsymbol{I}\right)\right)\boldsymbol{F}^{Ini-1}\\ \dot{E}^e{}_{\tilde{s}\tilde{t}} &= \frac{1}{\upsilon^{\theta^2}}F^{Ini-1}_{I\tilde{s}}\left[\dot{E}_{IJ} - \frac{1}{\upsilon^\theta}\frac{d\upsilon^\theta}{d\theta}\left(2E_{IJ} + \delta_{IJ}\right)\dot{\theta}\right]F^{Ini-1}_{J\tilde{t}}\end{aligned} \tag{6.144}$$

The free energy density, considering the multiplicative decomposition, can be separated into:

$$\psi = \psi^e(E^e_{\tilde{s}\tilde{t}}, \theta) + \psi^\theta(\theta) \tag{6.145}$$

The rate of free energy density is therefore expressed as:

$$\begin{aligned}\dot{\psi} &= \frac{\partial\psi^e}{\partial\boldsymbol{E}^e} : \dot{\boldsymbol{E}}^e + \frac{\partial\psi^e}{\partial\theta}\dot{\theta} + \frac{\partial\psi^\theta}{\partial\theta}\dot{\theta} = \frac{1}{\upsilon^{\theta^2}}\boldsymbol{F}^{Ini-1}\frac{\partial\psi^e}{\partial\boldsymbol{E}^e}\boldsymbol{F}^{Ini-T} : \dot{\boldsymbol{E}}\\ &\quad - \left[\frac{1}{\upsilon^\theta}\frac{d\upsilon^\theta}{d\theta}\boldsymbol{F}^{Ini-1}\frac{\partial\psi^e}{\partial\boldsymbol{E}^e}\boldsymbol{F}^{Ini-T} : \left(\boldsymbol{I} + 2\boldsymbol{E}\right) - \frac{\partial\psi^e}{\partial\theta} - \frac{\partial\psi^\theta}{\partial\theta}\right]\dot{\theta}\\ \dot{\psi} &= \frac{\partial\psi^e}{\partial E^e_{\tilde{s}\tilde{t}}} : \dot{E}^e_{\tilde{s}\tilde{t}} + \frac{\partial\psi^e}{\partial\theta}\dot{\theta} + \frac{\partial\psi^\theta}{\partial\theta}\dot{\theta} = \frac{1}{\upsilon^{\theta^2}}F^{Ini-1}_{\tilde{s}I}\frac{\partial\psi^e}{\partial E^e_{\tilde{s}\tilde{t}}}F^{Ini-1}_{\tilde{t}J} : \dot{E}_{IJ}\\ &\quad - \left[\frac{1}{\upsilon^\theta}\frac{d\upsilon^\theta}{d\theta}F^{Ini-1}_{\tilde{s}I}\frac{\partial\psi^e}{\partial E^e_{\tilde{s}\tilde{t}}}F^{Ini-1}_{\tilde{t}J} : \left(2E_{IJ} + \delta_{IJ}\right) - \frac{\partial\psi^e}{\partial\theta} - \frac{\partial\psi^\theta}{\partial\theta}\right]\dot{\theta}\end{aligned} \tag{6.146}$$

Through the Clausius-Plank inequality, the stress tensor is expressed as:

$$\begin{aligned}\boldsymbol{S} &= \frac{\rho_0}{\upsilon^{\theta^2}}\boldsymbol{F}^{Ini-1}\frac{\partial\psi^e}{\partial\boldsymbol{E}^e}\boldsymbol{F}^{Ini-T}\\ S_{IJ} &= \frac{\rho_0}{\upsilon^{\theta^2}}F^{Ini-1}_{I\tilde{s}}\frac{\partial\psi^e}{\partial E^e_{\tilde{s}\tilde{t}}}F^{Ini-1}_{J\tilde{t}}\end{aligned} \tag{6.147}$$

Considering, $\rho_0 = \rho_{ine}J_{ine} = \rho_{ine}\upsilon^{\theta^3}J^{Ini}$, then:

$$\begin{aligned}\boldsymbol{S} &= \rho_{ine}\upsilon^\theta J^{Ini}\boldsymbol{F}^{Ini-1}\frac{\partial\psi^e}{\partial\boldsymbol{E}^e}\boldsymbol{F}^{Ini-T}\\ S_{IJ} &= \rho_{ine}\upsilon^\theta J^{Ini}F^{Ini-1}_{I\tilde{s}}\frac{\partial\psi^e}{\partial E^e_{\tilde{s}\tilde{t}}}F^{Ini-1}_{J\tilde{t}}\end{aligned} \tag{6.148}$$

For the particular constitutive model of St Venant-Kirchhof, $\psi^e(\boldsymbol{E}^e, \theta)$ is expressed as:

$$\psi^e(E^e_{\tilde{s}\tilde{t}}, \theta) = \lambda(\theta) E^e_{\tilde{s}\tilde{s}} + 2\mu(\theta) E^e_{\tilde{s}\tilde{t}} \tag{6.149}$$

Rewriting the definition of elastic Green strain tensor

$$\begin{aligned} \boldsymbol{E}^e &= \frac{1}{\upsilon^{\theta^2}} \boldsymbol{F}^{Ini^{-T}} \left(\frac{1}{2} \boldsymbol{F}^T \boldsymbol{F} - \frac{1}{2} \upsilon^{\theta^2} \boldsymbol{F}^{Ini^T} \boldsymbol{F}^{Ini} \right) \boldsymbol{F}^{Ini^{-1}} \\ &= \frac{1}{\upsilon^{\theta^2}} \boldsymbol{F}^{Ini^{-T}} \left(\boldsymbol{E} - \frac{1}{2} \left(\upsilon^{\theta^2} \boldsymbol{F}^{Ini^T} \boldsymbol{F}^{Ini} - \boldsymbol{I} \right) \right) \boldsymbol{F}^{Ini^{-1}} \end{aligned} \tag{6.150}$$

the trace is defined as

$$\begin{aligned} tr(\boldsymbol{E}^e) &= \frac{1}{\upsilon^{\theta^2}} \left[tr(\boldsymbol{F}^{Ini^{-T}} \boldsymbol{E} \boldsymbol{F}^{Ini^{-1}}) - tr(\frac{1}{2} \upsilon^{\theta^2}) + tr(\frac{1}{2} \boldsymbol{F}^{Ini^{-T}} \boldsymbol{F}^{Ini^{-1}}) \right] \\ &= \frac{1}{\upsilon^{\theta^2}} \left[tr(\boldsymbol{F}^{Ini^{-T}} \boldsymbol{E} \boldsymbol{F}^{Ini^{-1}}) - \frac{3}{2} \upsilon^{\theta^2} + tr(\frac{1}{2} \boldsymbol{F}^{Ini^{-T}} \boldsymbol{F}^{Ini^{-1}}) \right] \end{aligned} \tag{6.151}$$

Considering the previous expression, Eqs. (6.148) and (6.149), the stress is defined as:

$$\begin{aligned} \boldsymbol{S}^e = \frac{1}{\upsilon^{\theta^2}} \Big[& \lambda(\theta) \left(tr(\boldsymbol{F}^{Ini^{-T}} \boldsymbol{E} \boldsymbol{F}^{Ini^{-1}}) - \frac{3}{2} \upsilon^{\theta^2} + tr(\frac{1}{2} \boldsymbol{F}^{Ini^{-T}} \boldsymbol{F}^{Ini^{-1}}) \right) \boldsymbol{I} \\ & + 2\mu(\theta) \left(\boldsymbol{F}^{Ini^{-T}} \boldsymbol{E} \boldsymbol{F}^{Ini^{-1}} - \frac{1}{2} \left(\upsilon^{\theta^2} \boldsymbol{I} - \boldsymbol{F}^{Ini^{-T}} \boldsymbol{F}^{Ini^{-1}} \right) \right) \Big] \end{aligned} \tag{6.152}$$

where $3k(\theta) = 3\lambda(\theta) + 2\mu(\theta)$. Then assuming $\upsilon^\theta \approx 1 + \varepsilon^{th}$, Eq. (6.154) is expressed as:

$$\begin{aligned} \boldsymbol{S} = \frac{1}{\upsilon^\theta} J^{Ini} \boldsymbol{F}^{Ini^{-1}} \Big[& \lambda(\theta) \left(tr(\boldsymbol{F}^{Ini^{-T}} \boldsymbol{E} \boldsymbol{F}^{Ini^{-1}}) + \frac{1}{2} tr(\boldsymbol{F}^{Ini^{-T}} \boldsymbol{F}^{Ini^{-1}}) \right) \boldsymbol{I} \\ & + 2\mu(\theta) \left(\boldsymbol{F}^{Ini^{-T}} \boldsymbol{E} \boldsymbol{F}^{Ini^{-1}} + \frac{1}{2} \boldsymbol{F}^{Ini^{-T}} \boldsymbol{F}^{Ini^{-1}} \right) - \frac{3}{2} k(\theta) \upsilon^{\theta^2} \boldsymbol{I} \Big] \boldsymbol{F}^{Ini^{-T}} \\ S_{IJ} = \frac{1}{\upsilon^\theta} J^{Ini} F^{Ini^{-1}}_{I\tilde{s}} \Big[& \lambda(\theta) \left(F^{Ini^{-1}}_{A\tilde{u}} E_{AB} F^{Ini^{-1}}_{B\tilde{u}} + \frac{1}{2} F^{Ini^{-1}}_{A\tilde{u}} F^{Ini^{-1}}_{A\tilde{u}} \right) \delta_{\tilde{s}\tilde{t}} \\ & + 2\mu(\theta) \left(F^{Ini^{-1}}_{A\tilde{s}} E_{AB} F^{Ini^{-1}}_{B\tilde{t}} + \frac{1}{2} F^{Ini^{-1}}_{A\tilde{s}} F^{Ini^{-1}}_{A\tilde{t}} \right) - \frac{3}{2} k(\theta) \upsilon^{\theta^2} \delta_{\tilde{s}\tilde{t}} \Big] F^{Ini^{-1}}_{J\tilde{t}} \end{aligned} \tag{6.153}$$

Assuming that ε^{th} is small enough that $|\varepsilon^{th}| < 1$, then Eq. (6.153) can be written as:

$$\begin{aligned} S_{IJ} = J^{Ini} F^{Ini^{-1}}_{I\tilde{s}} \Big[& \left(1 - \varepsilon^{th} \right) \lambda(\theta) \left(F^{Ini^{-1}}_{A\tilde{u}} E_{AB} F^{Ini^{-1}}_{B\tilde{u}} + \frac{1}{2} F^{Ini^{-1}}_{A\tilde{u}} F^{Ini^{-1}}_{A\tilde{u}} \right) \delta_{\tilde{s}\tilde{t}} \\ & + 2 \left(1 - \varepsilon^{th} \right) \mu(\theta) \left(F^{Ini^{-1}}_{A\tilde{s}} E_{AB} F^{Ini^{-1}}_{B\tilde{t}} + \frac{1}{2} F^{Ini^{-1}}_{A\tilde{s}} F^{Ini^{-1}}_{A\tilde{t}} \right) - \frac{3}{2} k(\theta) \upsilon^\theta \delta_{\tilde{s}\tilde{t}} \Big] F^{Ini^{-1}}_{J\tilde{t}} \end{aligned} \tag{6.154}$$

Therefore the generalised coupling tensor a_{ij} is defined as:

$$
\begin{aligned}
a_{iJ} &= -\frac{\partial^2\psi}{\partial F_{iJ}\partial\theta} = -\frac{\partial^2\psi}{\partial E_{PQ}\partial\theta}\frac{\partial E_{PQ}}{\partial F_{iJ}} \\
&= -J^{Ini}F^{Ini-1}_{P\tilde{s}}\Bigg[\left(1-\varepsilon^{th}\right)\Bigg[\frac{\partial\lambda(\theta)}{\partial\theta}\left(F^{Ini-1}_{A\tilde{u}}E_{AB}F^{Ini-1}_{B\tilde{u}}+\frac{1}{2}F^{Ini-1}_{A\tilde{u}}F^{Ini-1}_{A\tilde{u}}\right)\delta_{\tilde{s}\tilde{t}} \\
&+2\frac{\partial\mu(\theta)}{\partial\theta}\left(F^{Ini-1}_{A\tilde{s}}E_{AB}F^{Ini-1}_{B\tilde{t}}+\frac{1}{2}F^{Ini-1}_{A\tilde{s}}F^{Ini-1}_{A\tilde{t}}\right)\Bigg] \\
&-\frac{\partial\varepsilon^{th}}{\partial\theta}\Bigg[\lambda(\theta)\left(F^{Ini-1}_{A\tilde{u}}E_{AB}F^{Ini-1}_{B\tilde{u}}+\frac{1}{2}F^{Ini-1}_{A\tilde{u}}F^{Ini-1}_{A\tilde{u}}\right)\delta_{\tilde{s}\tilde{t}} \\
&+2\mu(\theta)\left(F^{Ini-1}_{A\tilde{s}}E_{AB}F^{Ini-1}_{B\tilde{t}}+\frac{1}{2}F^{Ini-1}_{A\tilde{s}}F^{Ini-1}_{A\tilde{t}}\right)\Bigg] \\
&-\frac{3}{2}\upsilon^{\theta}\frac{\partial k(\theta)}{\partial\theta}\delta_{\tilde{s}\tilde{t}}-\frac{3}{2}\frac{\partial\upsilon^{\theta}}{\partial\theta}k(\theta)\delta_{\tilde{s}\tilde{t}}\Bigg]F^{Ini-1}_{Q\tilde{t}}\frac{1}{2}\left[\frac{\partial\left(F_{rP}F_{rQ}-\delta_{PQ}\right)}{\partial F_{iJ}}\right] \\
&= X_{PQ}\frac{1}{2}\left[\delta_{ri}\delta_{PJ}F_{rQ}+F_{rP}\delta_{ri}\delta_{QJ}\right] = X_{PQ}\frac{1}{2}\left[\delta_{PJ}F_{iQ}+\delta_{QJ}F_{iP}\right]
\end{aligned}
\tag{6.155}
$$

Once the stress tensor and generalised coupling have been defined, the rest of derivations in index notation of the components of the stiffness matrix are presented below. All these components have been implemented in the corresponded constitutive model (marked in blue). However, some terms are common to all models and have been kept in "***Thermomechanics.cpp***" (marked in black).

6.3.8.1 Derivation of $\frac{\partial a_{ij}}{\partial\theta}$

The term $\frac{\partial a_{iJ}}{\partial\theta}$ in Eq.(6.109) can be derived from Eq.(6.155) as:

$$
\begin{aligned}
\frac{\partial a_{iJ}}{\partial\theta} &= -J^{Ini}F^{Ini-1}_{P\tilde{s}}\Bigg[\left(1-\varepsilon^{th}\right)\Bigg[\frac{\partial^2\lambda(\theta)}{\partial\theta^2}\left(F^{Ini-1}_{A\tilde{u}}E_{AB}F^{Ini-1}_{B\tilde{u}}\right)+\frac{1}{2}F^{Ini-1}_{A\tilde{u}}F^{Ini-1}_{A\tilde{u}}\Bigg)\delta_{\tilde{s}\tilde{t}} \\
&+2\frac{\partial^2\mu(\theta)}{\partial\theta^2}\left(F^{Ini-1}_{A\tilde{s}}E_{AB}F^{Ini-1}_{B\tilde{t}}+\frac{1}{2}F^{Ini-1}_{A\tilde{s}}F^{Ini-1}_{A\tilde{t}}\right)\Bigg] \\
&-2\frac{\partial\varepsilon^{th}}{\partial\theta}\Bigg[\frac{\partial\lambda(\theta)}{\partial\theta}\left(F^{Ini-1}_{A\tilde{u}}E_{AB}F^{Ini-1}_{B\tilde{u}}+\frac{1}{2}F^{Ini-1}_{A\tilde{u}}F^{Ini-1}_{A\tilde{u}}\right)\delta_{\tilde{s}\tilde{t}} \\
&+2\frac{\partial\mu(\theta)}{\partial\theta}\left(F^{Ini-1}_{A\tilde{s}}E_{AB}F^{Ini-1}_{B\tilde{t}}+\frac{1}{2}F^{Ini-1}_{A\tilde{s}}F^{Ini-1}_{A\tilde{t}}\right)\Bigg] \\
&-\frac{\partial^2\varepsilon^{th}}{\partial\theta^2}\Bigg[\lambda(\theta)\left(F^{Ini-1}_{A\tilde{u}}E_{AB}F^{Ini-1}_{B\tilde{u}}+\frac{1}{2}F^{Ini-1}_{A\tilde{u}}F^{Ini-1}_{A\tilde{t}}\right)\delta_{\tilde{s}\tilde{t}} \\
&+2\mu(\theta)\left(F^{Ini-1}_{A\tilde{s}}E_{AB}F^{Ini-1}_{B\tilde{t}}+\frac{1}{2}F^{Ini-1}_{A\tilde{s}}F^{Ini-1}_{A\tilde{t}}\right)\Bigg] \\
&-\frac{3}{2}\upsilon^{\theta}\frac{\partial^2 k(\theta)}{\partial\theta^2}\delta_{\tilde{s}\tilde{t}}-\frac{3}{2}\frac{\partial^2\upsilon^{\theta}}{\partial\theta^2}k(\theta)\delta_{\tilde{s}\tilde{t}}-3\frac{\partial\upsilon^{\theta}}{\partial\theta}\frac{\partial k(\theta)}{\partial\theta}\delta_{\tilde{s}\tilde{t}}\Bigg]F^{Ini-1}_{Q\tilde{t}}\frac{1}{2}\left[\delta_{PJ}F_{iQ}+\delta_{QJ}F_{iP}\right]
\end{aligned}
\tag{6.156}
$$

6.3.8.2 Derivation of $\frac{\partial a_{ij}}{\partial x_{kb}}$

The term $\frac{\partial a_{iJ}}{\partial x_{kb}}$ in Eq.(6.110) can be derived from Eq.(6.155) as:

$$\frac{\partial a_{iJ}}{\partial x_{kb}} = \frac{\partial a_{iJ}}{\partial F_{mN}}\frac{\partial F_{mN}}{\partial x_{kb}} = \Bigg[-J^{Ini}F^{Ini-1}_{P\tilde{s}}\Bigg[\left(1-\varepsilon^{th}\right)\Bigg[\frac{\partial\lambda(\theta)}{\partial\theta}\frac{1}{2}F^{Ini-1}_{A\tilde{u}}\left[\delta_{AN}F_{mB}+\delta_{BN}F_{mA}\right]F^{Ini-1}_{B\tilde{u}}\delta_{\tilde{s}\tilde{t}}$$
$$+\frac{\partial\mu(\theta)}{\partial\theta}F^{Ini-1}_{A\tilde{s}}\left[\delta_{AN}F_{mB}+\delta_{BN}F_{mA}\right]F^{Ini-1}_{B\tilde{t}}\Bigg]$$
$$-\frac{\partial\varepsilon^{th}}{\partial\theta}\Bigg[\lambda(\theta)\frac{1}{2}F^{Ini-1}_{A\tilde{u}}\left[\delta_{AN}F_{mB}+\delta_{BN}F_{mA}\right]F^{Ini-1}_{B\tilde{u}}\delta_{\tilde{s}\tilde{t}}+\mu(\theta)F^{Ini-1}_{A\tilde{s}}\left[\delta_{AN}F_{mB}+\delta_{BN}F_{mA}\right]F^{Ini-1}_{B\tilde{t}}\Bigg]\Bigg]$$
$$F^{Ini-1}_{Q\tilde{t}}\frac{1}{2}\left[\delta_{PJ}F_{iQ}+\delta_{QJ}F_{iP}\right]$$
$$+X_{PQ}\frac{1}{2}\left[\delta_{PJ}\delta_{im}\delta_{QN}+\delta_{QJ}\delta_{im}\delta_{PN}\right]\Bigg]N_{b,N}(\boldsymbol{\xi}_q)\delta_{km} \tag{6.157}$$

6.3.8.3 Derivation of $\frac{\partial \dot{F}_{iJ}}{\partial x_{kb}}$

The term $\frac{\partial \dot{F}_{iJ}}{\partial x_{kb}}$ in Eq.(6.110) can be derived as:

$$\frac{\partial \dot{F}_{iJ}}{\partial x_{kb}} = \frac{\partial \dot{F}_{iJ}}{\partial F_{mN}}\frac{\partial F_{mN}}{\partial x_{kb}} = \frac{1}{\Delta t}\delta_{im}\delta_{JN}N_{b,N}(\boldsymbol{\xi}_q)\delta_{km} \tag{6.158}$$

6.3.8.4 Derivation of $\frac{\partial P_{ij}}{\partial \theta}$

The term $\frac{\partial P_{ij}}{\partial \theta}$ in Eq.(6.111) can be derived from Eq.(6.135) as:

$$\frac{\partial P_{iJ}}{\partial \theta} = -F_{iK}X_{KJ}N_b(\boldsymbol{\xi}_q) \tag{6.159}$$

Tangent modulus

$$\mathbb{C}_{PJLQ} = \frac{\partial S_{PJ}}{\partial E_{LQ}} = J^{Ini}F^{Ini-1}_{P\tilde{s}}(1-\varepsilon^{th})\Bigg[\lambda(\theta)\frac{\partial F^{Ini-1}_{A\tilde{u}}E_{AB}F^{Ini-1}_{B\tilde{u}}}{\partial E_{LQ}}\delta_{\tilde{s}\tilde{t}}$$
$$+2\mu(\theta)F^{Ini-1}_{A\tilde{s}}\frac{\partial\left[\frac{1}{2}(E_{AB}+E_{BA})\right]}{\partial E_{LQ}}F^{Ini-1}_{B\tilde{t}}\Bigg]F^{Ini-1}_{J\tilde{t}}$$
$$= J^{Ini}F^{Ini-1}_{P\tilde{s}}(1-\varepsilon^{th})\left[\lambda(\theta)F^{Ini-1}_{A\tilde{u}}\delta_{AL}\delta_{QB}F^{Ini-1}_{B\tilde{u}}\delta_{\tilde{s}\tilde{t}}+\mu(\theta)F^{Ini-1}_{A\tilde{s}}\left(\delta_{AL}\delta_{BQ}+\delta_{BL}\delta_{AQ}\right)F^{Ini-1}_{B\tilde{t}}\right]F^{Ini-1}_{J\tilde{t}} \tag{6.160}$$

$$\mathbb{K}_{iJkL} = \frac{\partial^2\psi(\boldsymbol{F})}{\partial F_{iJ}\partial F_{kL}} = \mathbb{C}_{PJLQ}F_{kQ}F_{iP}+S_{LJ}\delta_{ik} \tag{6.161}$$

Summary of simple operations and transformations

$$
\begin{aligned}
E_{ij} &= \frac{1}{2}\left(F_{pi}F_{pj} - \delta_{ij}\right) \\
\frac{\partial Tr(\boldsymbol{E})}{\partial F_{kl}} &= \frac{\partial}{\partial F_{kl}}E_{ii} = \frac{\partial}{\partial F_{kl}}\frac{1}{2}\left(F_{pi}F_{pi} - \delta_{ii}\right) = \delta_{pk}\delta_{il}F_{pi} = \delta_{il}F_{ki} = F_{kl} \\
\frac{\partial E_{ij}}{\partial F_{kl}} &= \frac{1}{2}\left(\delta_{pk}\delta_{il}F_{pj} + F_{pi}\delta_{pk}\delta_{jl}\right) = \frac{1}{2}\left(\delta_{il}F_{kj} + \delta_{jl}F_{ki}\right)
\end{aligned}
$$

CHAPTER 7

STOCHASTIC METHOD

The goal of the Stochastic Finite Element Method (SFEM) is to seamlessly propagate uncertainties from the input level to the output level. The input level encompasses constitutive parameters, geometry, forces and initial conditions. To use the SFEM, the author must specify a keyword *Stochastic in the input file.
This chapter provides the mathematical and numerical details of the stochastic finite element method using a Saint Venant Kirchhoff model with random constitutive properties.

7.1 Stochastic Saint Venant Kirchhoff formulation

In this section the formulation of the stochastic finite element method is given. The main feature of this method is to consider that the displacement has both a spatial and a stochastic component. As a result, the displacement is written as follows:

$$\boldsymbol{u}^h(\boldsymbol{x},\boldsymbol{\theta}) = \sum_{i=1}^{n} N_i(\boldsymbol{x}) \sum_{\epsilon=1}^{\zeta} \Psi_\epsilon(\boldsymbol{\theta}) u_{i\epsilon} \tag{7.1}$$

where the functions Ψs are shape function of the stochastic space. The symbol θ Consequently at a given node, both spatial and stochastic components of the displacement have to be determined. The shape functions are depend of the probabilistic distribution of the inputs. If the inputs is a Gaussian distribution, the functions Ψ's should be Hermite polynomials to ensure optimal convergence. From now on, all indices related to randomness will be noted with Greek letters. H is the space generated by the functions Ψs. Suppose that every quantity can be written on this space, at the cost of accuracy (discussed in the appendix). As an example, the deformation gradient tensor has the following expression:

$$\boldsymbol{F} = \boldsymbol{F}_\epsilon \Psi_\epsilon = \left(\boldsymbol{I}\delta_{\epsilon 1} + \boldsymbol{\nabla}_0 \boldsymbol{u}_\epsilon\right) \Psi_\epsilon \tag{7.2}$$

where the matrix $\boldsymbol{F}_\epsilon$ is the stochastic component of $\boldsymbol{F}$ associated with the polynomial Ψ_ϵ. The entry at the intersection of the line i and the column J of the matrix $\boldsymbol{F}_\epsilon$ is noted $\boldsymbol{F}_{\epsilon iJ}$ The second Piola-Kirchhoff $\boldsymbol{S}$, now a random quantity, is linked to the Green Lagrangian strain tensor $\boldsymbol{E}$ by

the stochastic stress-strain constitutive relative relationship for the Saint Venant Kirchhoff model:

$$\boldsymbol{S}(\theta) = \lambda(\theta) tr(\boldsymbol{E})\boldsymbol{I} + 2\mu(\theta)\boldsymbol{E}. \tag{7.3}$$

7.2 Numerical integration

7.2.1 Balance of linear momentum with respect to the reference configuration

The equation of the balance of linear momentum with respect to the reference (or undeformed) configuration can be written as

$$\operatorname{Div} \boldsymbol{P} + \rho_0 \boldsymbol{b} = \rho_0 \ddot{\boldsymbol{x}}\,, \quad \forall \boldsymbol{X} \in \Omega_0 \text{ almost surely.} \tag{7.4}$$

Note that the previous equation is the extension of Equation 5.2 to the stochastic dimension specified by the term "almost surely".
Alternative stress measures typically used in the finite deformation framework include Cauchy stress (or true stress) $\boldsymbol{\sigma} = J^{-1}\boldsymbol{P}\cdot\boldsymbol{F}^T$, Kirchhoff stress $\boldsymbol{\tau} = J\boldsymbol{\sigma} = \boldsymbol{P}\cdot\boldsymbol{F}^T$, and the second Piola-Kirchhoff stress $\boldsymbol{S} = \boldsymbol{F}^{-1}\cdot\boldsymbol{P}$, where $J = \det \boldsymbol{F}$ is the Jacobian, the volumetric change relative to the original configuration. The extension of these operations on matrices are detailed in the appendix.

7.2.2 Balance of angular momentum with respect to the reference configuration

The balance of angular momentum is generally not imposed directly through the weak form. Instead, the constitutive model is chosen such that it holds at all times, i.e. that $\boldsymbol{\sigma}$, $\boldsymbol{S}$ or $\boldsymbol{P}\cdot\boldsymbol{F}^T$ is symmetric.

7.2.3 Boundary conditions

The Neumann and Dirichlet boundary conditions, $\bar{\boldsymbol{T}}$ on $\partial\Omega_n$ and $\bar{\boldsymbol{u}}$ on $\partial\Omega_d$, respectively, are imposed in the reference configuration by:

$$\begin{cases} \boldsymbol{P}\cdot\boldsymbol{N} = \bar{\boldsymbol{T}},\ \forall \boldsymbol{X} \in \partial\Omega_n \text{ almost surely} \\ \boldsymbol{u} = \bar{\boldsymbol{u}},\ \forall \boldsymbol{X} \in \partial\Omega_d \text{ almost surely} \end{cases} \tag{7.5}$$

where $\boldsymbol{N}$ is the normal to the boundary in the reference configuration, and where the subscript "0" (indicating the reference configuration) of $\partial\Omega_n$ and $\partial\Omega_d$ is dropped for clarity, see Figure 5.1.

7.2.4 Weak form of the balance of momentum

The weak form of the balance of linear momentum (see Equation (7.4)) can be formulated as follows: for all arbitrary admissible virtual displacement $\boldsymbol{\eta}$, with $\boldsymbol{\eta}(\boldsymbol{X}) = \boldsymbol{0}$ almost surely, for all $\boldsymbol{X} \in \partial\Omega_d$,

$$\int_\Theta \int_{\Omega_0} \operatorname{Div} \boldsymbol{P}\cdot\boldsymbol{\eta}\,\mathrm{d}V\mathrm{d}\theta + \int_\Theta \int_{\Omega_0} \rho_0 \boldsymbol{b}\cdot\boldsymbol{\eta}\,\mathrm{d}V\mathrm{d}\theta = \int_\Theta \int_{\Omega_0} \rho_0 \ddot{\boldsymbol{x}}\cdot\boldsymbol{\eta}\,\mathrm{d}V\mathrm{d}\theta \tag{7.6}$$

Equation (7.6) is the weak form, since it requires the condition of balance of linear momentum to be satisfied only as as an integral over the entire domain, whereas Equation (7.4) is the strong form (or canonical form) because it fulfills the conditions locally at each point $\boldsymbol{X} \in \Omega_0$.
Integrating by parts and using Green's theorem, the above expression can be rewritten as

$$\int_\Theta \int_{\partial\Omega_0} (\boldsymbol{P}\cdot\boldsymbol{N})\cdot\boldsymbol{\eta}\,\mathrm{d}S\mathrm{d}\theta + \int_\Theta \int_{\Omega_0} \rho_0 \boldsymbol{b}\cdot\boldsymbol{\eta}\,\mathrm{d}V\mathrm{d}\theta = \int_\Theta \int_{\Omega_0} \boldsymbol{P} : \operatorname{Grad}\boldsymbol{\eta}\,\mathrm{d}V\mathrm{d}\theta + \int_\Theta \int_{\Omega_0} \rho_0 \ddot{\boldsymbol{x}}\cdot\boldsymbol{\eta}\,\mathrm{d}V\mathrm{d}\theta \tag{7.7}$$

which, noting that $\boldsymbol{\eta} = \mathbf{0}$ almost surely for all $\boldsymbol{X} \in \partial\Omega_d$ and using Equation (7.5), leads to

$$\int_\Theta \int_{\partial\Omega_n} \bar{\boldsymbol{T}} \cdot \boldsymbol{\eta} \,\mathrm{d}S\mathrm{d}\theta + \int_\Theta \int_{\Omega_0} \rho_0 \boldsymbol{b} \cdot \boldsymbol{\eta} \,\mathrm{d}V\mathrm{d}\theta = \int_\Theta \int_{\Omega_0} \boldsymbol{P} : \mathrm{Grad}\,\boldsymbol{\eta} \,\mathrm{d}V\mathrm{d}\theta + \int_\Theta \int_{\Omega_0} \rho_0 \ddot{\boldsymbol{x}} \cdot \boldsymbol{\eta} \,\mathrm{d}V\mathrm{d}\theta \quad (7.8)$$

Note that the first Piola-Kirchhoff stress $\boldsymbol{P}$ is a function of the deformation gradient $\boldsymbol{F}$ and so a function of the displacement $\boldsymbol{u}$. The relation between $\boldsymbol{P}$ and $\boldsymbol{F}$ is defined by the material constitutive law $\boldsymbol{P} = \tilde{\boldsymbol{P}}(\boldsymbol{F})$.

7.3 Numerical integration

It should be emphasised that the displacements (real and virtual) and the stress are now random quantities. Consequently the displacement at a node a is $\boldsymbol{x}^a = \boldsymbol{x}^a_\epsilon \Psi_\epsilon$. The same decomposition applies for the virtual displacement, volumic mass and the stresses. The stochastic shape functions Ψ_ϵ are space and node independent.

Stochastic Finite element problem: Find all $\boldsymbol{x}^a \in \mathbb{R}^3 \otimes \Theta$ such that for all admissible virtual displacement $\boldsymbol{\eta}^b \in \mathbb{R}^3 \otimes \Theta$:

$$\begin{aligned}\sum_e \int_\Theta \int_{\Omega_0^e} \rho_{0\kappa} N_a^e N_b^e \ddot{\boldsymbol{x}}^a_\epsilon \cdot \boldsymbol{\eta}^b_\iota \Psi_\kappa \Psi_\epsilon \Psi_\iota \,\mathrm{d}V\mathrm{d}\theta + \sum_e \int_\Theta \int_{\Omega_0^e} \left(\tilde{\boldsymbol{P}} \left(\sum_{a \in \Omega_0^e} N_a^e(\boldsymbol{\xi})\, \boldsymbol{x}^a_\epsilon \right) \cdot \boldsymbol{\nabla}_0 N_b^e \right) \cdot \boldsymbol{\eta}^b_\iota \Psi_\iota \,\mathrm{d}V\mathrm{d}\theta \\ = \sum_e \int_\Theta \int_{\partial\Omega_n^e} N_b^e \bar{\boldsymbol{T}} \cdot \boldsymbol{\eta}^b_\iota \Psi_\iota \,\mathrm{d}S\mathrm{d}\theta + \sum_e \int_\Theta \int_{\Omega_0^e} \rho_0 N_b^e \boldsymbol{b} \cdot \boldsymbol{\eta}^b_\iota \Psi_\iota \,\mathrm{d}V\mathrm{d}\theta\end{aligned} \quad (7.9)$$

Since the above equation must be valid for all admissible $\boldsymbol{\eta}^b$, the finite element problem now reads: Find $\boldsymbol{x}^a$ such that, for all b,

$$\sum_e \boldsymbol{M}^e_{\epsilon b \iota a} \ddot{\boldsymbol{x}}^a_\epsilon + \sum_e \boldsymbol{f}^{e\,\mathrm{int}}_{b\iota}(\boldsymbol{x}^a_\epsilon) = \sum_e \boldsymbol{f}^{e\,\mathrm{ext}}_{b\iota} \quad (7.10)$$

where the mass matrix $\boldsymbol{M}^e$, the element internal force vector $\boldsymbol{f}^{e\,\mathrm{int}}_b(\boldsymbol{x}^a)$ and external force vector $\boldsymbol{f}^{e\,\mathrm{ext}}_b$ are

$$\boldsymbol{M}^e_{\epsilon b \iota a} = \int_\Theta \int_{\Omega_0^e} \rho_{0\kappa} N_a^e N_b^e \Psi_\kappa \Psi_\epsilon \Psi_\iota \,\mathrm{d}V\mathrm{d}\theta \quad (7.11)$$

$$\boldsymbol{f}^{e\,\mathrm{int}}_{\iota b}(\boldsymbol{x}^a) = \int_\Theta \int_{\Omega_0^e} \tilde{\boldsymbol{P}} \left(\sum_{a \in \Omega_0^e} N_a^e(\boldsymbol{\xi})\, \boldsymbol{x}^a \right) \cdot \boldsymbol{\nabla}_0 N_b^e \Psi_\iota \,\mathrm{d}V \quad (7.12)$$

$$\boldsymbol{f}^{e\,\mathrm{ext}}_{\iota b} = \int_\Theta \int_{\partial\Omega_n^e} N_b^e \bar{\boldsymbol{T}} \Psi_\iota \,\mathrm{d}S\mathrm{d}\theta + \int_\Theta \int_{\Omega_0^e} \rho_0 N_b^e \boldsymbol{b} \Psi_\iota \,\mathrm{d}V\mathrm{d}\theta. \quad (7.13)$$

Expending $\tilde{\boldsymbol{P}}$ as $\tilde{\boldsymbol{P}}_\tau \Psi_\tau$, $\bar{\boldsymbol{T}}$ and $\boldsymbol{b}$ in the same manner, the internal and external forces become:

$$\boldsymbol{f}^{e\,\mathrm{int}}_{\iota b}(\boldsymbol{x}^a) = \int_\Theta \int_{\Omega_0^e} \tilde{\boldsymbol{P}}_\tau \cdot \boldsymbol{\nabla}_0 N_b^e \Psi_\iota \Psi_\tau \,\mathrm{d}V\mathrm{d}\theta \quad (7.14)$$

$$\boldsymbol{f}^{e\,\mathrm{ext}}_{\iota b} = \int_\Theta \int_{\partial\Omega_n^e} N_b^e \bar{\boldsymbol{T}}_\tau \Psi_\iota \Psi_\tau \,\mathrm{d}S\mathrm{d}\theta + \int_\Theta \int_{\Omega_0^e} \rho_{0\kappa} N_b^e \boldsymbol{b}_\tau \Psi_\kappa \Psi_\iota \Psi_\tau \,\mathrm{d}V\mathrm{d}\theta. \quad (7.15)$$

For simplification purposes, the quantity $\int_\Theta f(\theta)g(\theta)\mathrm{d}\theta$ is denoted $< f, g >$ where f and g are two functions. Furthermore the term $< \Psi_\iota, \Psi_\tau >$ is non-zero only in the case where $\iota = \tau$. Consequently, we can further simplify the expressions of the forces as follow:

$$\boldsymbol{f}^{e\,\mathrm{int}}_{\iota b}(\boldsymbol{x}^a) = \int_{\Omega^e_0} \tilde{\boldsymbol{P}}_\iota \cdot \boldsymbol{\nabla}_0 N^e_b < \Psi_\iota, \Psi_\iota > \mathrm{d}V \tag{7.16}$$

$$\boldsymbol{f}^{e\,\mathrm{ext}}_{\iota b} = \int_{\partial\Omega^e_n} N^e_b \bar{\boldsymbol{T}}_\tau < \Psi_\iota, \Psi_\iota > \mathrm{d}S + \int_\Theta \int_{\Omega^e_0} \rho_{0\kappa} N^e_b \boldsymbol{b}_\iota < \Psi_\kappa \Psi_\iota, \Psi_\tau > \mathrm{d}V. \tag{7.17}$$

These element tensors are finally calculated using Gauss quadrature approximation in the isoparametric coordinate system. This means that each element e can be integrated upon with a function $f(\boldsymbol{x})$ by assuming that:

$$\int_{\Omega^e_0} f(\boldsymbol{\varphi}(\boldsymbol{X}))\,\mathrm{d}V = \sum_{q\in\Omega^e_0} w_q f(\boldsymbol{\varphi}(\boldsymbol{\Phi}(\boldsymbol{\xi}_q)))J_0(\boldsymbol{\xi}_q) \tag{7.18}$$

where $\boldsymbol{\xi}_q$ is the coordinate vector of Gauss point q of element e in the isoparametric coordinate system, w_q is its corresponding weight, and J_0 is the Jacobian between the isoparametric coordinate system and the reference configuration. This quadrature only concerns the physical space as all integrals on the stochastic space can be computed without any numerical approximations. Noting that each mass matrix element $\boldsymbol{M}^e_{\epsilon b\iota a}$ relates $\ddot{\boldsymbol{x}}^a$ to the other force vectors in b coordinate-by-coordinate, the matrix can be extended to the nodal coordinates by $\boldsymbol{M}^e_{\iota a\epsilon b} = \delta_{ik}\boldsymbol{M}^e_{\iota ia\epsilon kb}$ where δ_{ik} is the Kronecker symbol. Swapping a and b, in indicial notation:

$$\begin{aligned}
M^e_{\iota ia\epsilon kb} &= \sum_q w_q \rho_{0\kappa}\delta_{ik} N^e_a(\boldsymbol{\xi}_q) N^e_b(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) < \Psi_\kappa\Psi_\epsilon, \Psi_\iota > \\
f^{e\,\mathrm{int}}_{\iota ia} &= \sum_q w_q P_{\iota iJ}(\boldsymbol{\xi}_q) N^e_{a,J}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) < \Psi_\iota, \Psi_\iota > \\
f^{e\,\mathrm{ext}}_{\iota ia} &= \sum_p w_p T_{\iota i}(\boldsymbol{\xi}_p) N^e_a(\boldsymbol{\xi}_p) J_0(\boldsymbol{\xi}_p) < \Psi_\iota, \Psi_\iota > + \\
&\quad \sum_q w_q \rho_{0\kappa} b_{\iota i}(\boldsymbol{\xi}_q) N^e_a(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) < \Psi_\kappa\Psi_\epsilon, \Psi\iota >
\end{aligned} \tag{7.19}$$

The notation with subscripts p (first term of Equation (5.17)) are for the surface elements belonging to $\partial\Omega^e_{dh}$ mapped onto the corresponding faces of the volume elements.
As the internal force vector $\boldsymbol{f}^{e\,\mathrm{int}}$ is not necessarily linear, a Newton-Raphson procedure can be used in conjunction with the time discretisation scheme (see next section). In this case, the element stiffness matrix $\boldsymbol{K}^e$ is needed:

$$Res^e_{\iota ia} = f^{e\,\mathrm{int}}_{\iota ia} - f^{e\,\mathrm{ext}}_{\iota ia} \tag{7.20}$$

$$K^e_{\iota ia\epsilon kb} = \frac{\partial Res^e_{\iota ia}}{\partial x_{\epsilon kb}} = \frac{\partial f^{e\,\mathrm{int}}_{\iota ia}}{\partial x_{\epsilon kb}} = \sum_q w_q \frac{\partial P_{\iota iJ}}{\partial F_{oqM}} \frac{\partial F_{oqM}}{\partial x_{\epsilon kb}}(\boldsymbol{\xi}_q) N^e_{a,J}(\boldsymbol{\xi}_q) J_0(\boldsymbol{\xi}_q) \tag{7.21}$$

where Res is the remainder (or residual). The tangent moduli are provided by the constitutive law at each time step. More details about the derivation of the terms $\frac{\partial P_{\iota\iota J}}{\partial F_{qoM}}$ and $\frac{\partial F_{qoM}}{\partial x_{k\epsilon b}}$ can be

found in the Appendix. Here, the subscript 'u' refers to the stochastic component of the quantity. Consequently the increment of stochastic displacement $\boldsymbol{\Delta x_{n+1}}$ must verify:

$$\sum_{u=1}^{N_\epsilon} \boldsymbol{K}^{u,eq}(\boldsymbol{x}_{n+1}^{k,u}) \sum_{r=1}^{N_\epsilon} \boldsymbol{\Delta x_{n+1}^r} \Psi_u \Psi_r = \sum_{s=1}^{N_\epsilon} (\boldsymbol{f}_{n+1}^{ext,s} - \boldsymbol{f}^{int,s}(\boldsymbol{x}_{n+1}^{k,s})) \Psi_s. \tag{7.22}$$

If it is supposed that the size of the deterministic problem is N, then this system is composed of N equations. However, N times N_ϵ unknowns have yet to be determined as Δx_{n+1}^r is composed of N unknowns for each value of r between 1 and N_ϵ. In order to close the system, a Galerkin projection of Eq. (7.22) is used.The symbol $< ., . >$ denotes this projection. Then (7.22) becomes :

$$\sum_{u=1}^{N_\epsilon} \boldsymbol{K}^{u,eq}(\boldsymbol{x}_{n+1}^{k,u}) \sum_{r=1}^{N_\epsilon} \boldsymbol{\Delta x_{n+1}^r} < \Psi_u \Psi_r, \psi_s >= \sum_{s=1}^{N_\epsilon} (\boldsymbol{f}_{n+1}^{ext,s} - \boldsymbol{f}^{int,s}(\boldsymbol{x}_{n+1}^{k,s})) < \Psi_s, \Psi_s > . \tag{7.23}$$

The subscript t can take all integer values between 1 and N_ϵ. Assuming the vector Δx_{n+1} gathers all the stochastic components of the increment of displacement $[\Delta x_{n+1}^1,...,\Delta x_{n+1}^{N_\epsilon}]$, Eq. 7.23 can be rewritten as :

$$\begin{pmatrix} \sum_{u=1}^{N_\epsilon} C_{u11} K^{u,eq} & \cdots & \sum_{u=1}^{N_\epsilon} C_{u1N_\epsilon} K^{u,eq} \\ \vdots & \ddots & \vdots \\ \sum_{u=1}^{N_\epsilon} C_{uN_\epsilon 1} K^{u,eq} & \cdots & \sum_{u=1}^{N_\epsilon} C_{uN_\epsilon N_\epsilon} K^{u,eq} \end{pmatrix} \begin{pmatrix} \Delta x_{n+1}^1 \\ \vdots \\ \Delta x_{n+1}^{N_\epsilon} \end{pmatrix} = \begin{pmatrix} \sum_{u=1}^{N_\epsilon} (\boldsymbol{f}_{n+1}^{ext,u} - \boldsymbol{f}^{int,u}(\boldsymbol{x}_{n+1}^{k,u})) < \Psi_u, \Psi_1 > \\ \vdots \\ \sum_{u=1}^{N_\epsilon} (\boldsymbol{f}_{n+1}^{ext,u} - \boldsymbol{f}^{int,u}(\boldsymbol{x}_{n+1}^{k,u})) < \Psi_u, \Psi_{N_\epsilon} > \end{pmatrix}, K\Delta x = F. \tag{7.24}$$

The computation of the different elements of the stiffness matrices and the force vector requires the implementation of the corresponding algebra. Its theory is developed in the appendix.

7.4 Implicit dynamic and explicit

In the case the inertial force are taken into account, (7.25) becomes:

$$\sum_{u=1}^{N_\epsilon} \boldsymbol{M^u} \Psi_u \cdot \{\sum_{r=1}^{N_\epsilon} \ddot{\boldsymbol{x}}_{n+1}^{k,r} + \Delta \ddot{\boldsymbol{x}}_{n+1}^{r}\} \Psi_r + \sum_{s=1}^{N_\epsilon} \boldsymbol{f}^{int,s}(\boldsymbol{x}_{n+1}^{k}) \Psi_s + \frac{\partial \boldsymbol{f}^{int}}{\partial \boldsymbol{x}}(\boldsymbol{x}_{n+1}^{k}) \sum_{r=1}^{N_\epsilon} \Delta \boldsymbol{x}_{n+1}^{r} \Psi_r = \sum_{s=1}^{N_\epsilon} \boldsymbol{f}_{n+1}^{ext} \Psi_s \tag{7.25}$$

where $\boldsymbol{K} = \frac{\partial \boldsymbol{f}^{int}}{\partial \boldsymbol{x}}$ is the stiffness matrix. This stiffness matrix is also expanded as $\boldsymbol{K} = \sum_{u=1}^{N_\epsilon} \boldsymbol{K}^u \Psi_u$. This equation has to be divided into other equation using to a Galerkin projection.

$$\begin{aligned} \sum_{u=1}^{N_\epsilon} \boldsymbol{M^u} \cdot \{\sum_{r=1}^{N_\epsilon} \ddot{\boldsymbol{x}}_{n+1}^{k,r} + \Delta \ddot{\boldsymbol{x}}_{n+1}^{k,r}\} < \Psi_r \Psi_u, \Psi_s > + \sum_{s=1}^{N_\epsilon} \boldsymbol{f}^{int,s}(\boldsymbol{x}_{n+1}^{k}) < \Psi_s, \Psi_s > + \\ \sum_{u=1}^{N_\epsilon} \boldsymbol{K}^u(\boldsymbol{x}_{n+1}^{k}) \sum_{r=1}^{N_\epsilon} \Delta \boldsymbol{x}_{n+1}^{k,r} < \Psi_r \Psi_u, \Psi_s >= \sum_{s=1}^{N_\epsilon} \boldsymbol{f}_{n+1}^{ext,s} < \Psi_s, \Psi_s > \end{aligned} \tag{7.26}$$

Assuming that $\Delta \ddot{\boldsymbol{x}}_{n+1}^r$ and $\Delta \boldsymbol{x}_{n+1}^r$ are proportional to each other, it is possible to solve for $\Delta \boldsymbol{x}_{n+1}^r$ by rewriting (7.26) as:

$$\boldsymbol{x}_{n+1}^{k+1} = \boldsymbol{x}_{n+1}^{k} - [\boldsymbol{K}^{eq}(\boldsymbol{x}_{n+1}^{k})]^{-1} \cdot \boldsymbol{r}^k \tag{7.27}$$

where $\boldsymbol{K}^{eq}$ is defined by:

$$\boldsymbol{K}^{eq}(\boldsymbol{x}_{n+1}^k) = \frac{1}{\beta \Delta t^2}\boldsymbol{M} + \boldsymbol{K}(\boldsymbol{x}_{n+1}^k) \tag{7.28}$$

and $\boldsymbol{r}^k$ is the residual:

$$\boldsymbol{r}^k = \boldsymbol{M} \cdot \ddot{\boldsymbol{x}}_{n+1}^k + \boldsymbol{f}^{int}(\boldsymbol{x}_{n+1}^k) - \boldsymbol{f}_{n+1}^{ext} \tag{7.29}$$

The expression of the matrix $\boldsymbol{K}$ is specified in (7.34). The expression of the matrix M follows a similar pattern:

$$K = \begin{pmatrix} \sum_{u=1}^{N_\epsilon} C_{u11} M^u & \cdots & \sum_{u=1}^{N_\epsilon} C_{u1N_\epsilon} M^u \\ \vdots & \ddots & \vdots \\ \sum_{u=1}^{N_\epsilon} C_{uN_\epsilon 1} M^u & \cdots & \sum_{u=1}^{N_\epsilon} C_{uN_\epsilon N_\epsilon} M^u \end{pmatrix} \tag{7.30}$$

In the case where the mass is supposed to be deterministic, i.e, M^u is null when u is bigger than 1, then the mass matrix has a diagonal shape:

$$M = \begin{pmatrix} C_{111} M^1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & C_{1N_\epsilon N_\epsilon} M^1 \end{pmatrix} \tag{7.31}$$

In the explicit framework, no remainder is used. Equation (7.32) is rewritten as:

$$\sum_{u=1}^{N_\epsilon} M^u \psi_u \cdot \sum_{r=1}^{N_\epsilon} \ddot{\boldsymbol{x}}_{n+1}^r \psi_r = (\sum_{s=1}^{N_\epsilon} \boldsymbol{f}_{n+1}^{ext,s} - \sum_{s=1}^{N_\epsilon} \boldsymbol{f}_{n+1}^{int,s}) \Psi_s \tag{7.32}$$

Here $\ddot{x_{n+1}}$ is derived thanks to Equation (5.41) using $\beta = 0$. This vector gathers all deterministic stochastic counterparts of the incremental acceleration vector. It is then possible to solve for the updated displacement:

$$\begin{pmatrix} C_{111} M^1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & C_{1N_\epsilon N_\epsilon} M^1 \end{pmatrix} \begin{pmatrix} \Delta x_{n+1}^1 \\ \vdots \\ \Delta x_{n+1}^{N_\epsilon} \end{pmatrix} = \begin{pmatrix} \sum_{u=1}^{N_\epsilon} (\boldsymbol{f}_{n+1}^{ext,u} - \boldsymbol{f}^{int,u}(\boldsymbol{x}_{n+1}^{k,u})) < \Psi_u, \Psi_1 > \\ \vdots \\ \sum_{u=1}^{N_\epsilon} (\boldsymbol{f}_{n+1}^{ext,u} - \boldsymbol{f}^{int,u}(\boldsymbol{x}_{n+1}^{k,u})) < \Psi_u, \Psi_{N_\epsilon} > \end{pmatrix}, \tag{7.33}$$

Noting that $< \Psi_u, \Psi_i >= C_{1ui}$, the system can be further simplified:

$$\begin{pmatrix} M^1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & M^1 \end{pmatrix} \begin{pmatrix} \Delta x_{n+1}^1 \\ \vdots \\ \Delta x_{n+1}^{N_\epsilon} \end{pmatrix} = \begin{pmatrix} \sum_{u=1}^{N_\epsilon} (\boldsymbol{f}_{n+1}^{ext,u} - \boldsymbol{f}^{int,u}(\boldsymbol{x}_{n+1}^{k,u})) \\ \vdots \\ \sum_{u=1}^{N_\epsilon} (\boldsymbol{f}_{n+1}^{ext,u} - \boldsymbol{f}^{int,u}(\boldsymbol{x}_{n+1}^{k,u})) \end{pmatrix}, \tag{7.34}$$

A lumped scheme can be used to avoid the expensive cost that the inversion of a matrix require. Consequently, acceleration are solved DoF's by DoF's.

7.5 Appendix

The appendix details the theory of SFEM and its application on *MuPhiSim*.

7.5.1 Approximation of the solution

The displacement is approximated on both the physical and the stochastic space. Two different kinds of shape functions are available on *MuPhiSim*:
- The first type of approximation involves the polynomial chaos. This approximation is the most used in the solid mechanics community. The functions Ψ are smooth polynomials. The expression of the polynomial depends on the probabilistic distribution of the randomnesses introduced in the problem. Table 7.1 shows the guidance to chose the appropriate polynomial chaos.
- The second type approximation involves Haar polynomials. This approximation is rarely used in solid mechanics. These polynomials are piecewise continuous which allows one to capture singularities. As opposed to polynomial chaos, Haar polynomials have the same expression regardless of the distribution of the uncertainties.

Input probabilistic distribution	Polynomial chaos
Gaussian	Hermite
Uniform	Legendre
Gamma	Laguerre
Beta	Jacobi

Table 7.1: Correspondence of the type of polynomial chaos and the underlying probabilistic distribution of the input

7.5.1.1 Polynomial chaos expansion

The principle of the PCE, introduced for first time in 1938 by Weiner, is to create a metamodel that determines the evolution of uncertainty in a dynamical system. This metamodel is dependent of the probabilistic distribution of input parameters. According to the Cameron-Martin theorem , such an expansion converges in the L_2 sense for a stochastic process with finite variance depending on Gaussian random variables. Xiu extended these results to various probabilistic distribution of the inputs. Calling $\boldsymbol{\epsilon} = [\varepsilon_1, ..., \varepsilon_M]$ the vector that fully describes the uncertainty of the inputs. The univariate polynomials $\psi_{\lambda_{i,j}}$, where $\lambda_{i,j}$ indicates that the degree of the polynomial associated with the variable ε_j is i, must satisfy this integral:

$$\int_{\Omega_j} \psi_{\lambda_{i,j}} \psi_{\lambda_{k,j}} f(\varepsilon_j) d\varepsilon_j = 0 \text{ if } k \neq i, \tag{7.35}$$

where the function $f(\varepsilon_j)$ is the probability distribution of the random variable ε_j. For the sake of simplicity, the random variables are supposed to be uncorrelated so that the joint probabilistic distribution is the product of each probabilistic distribution functions. The polynomials ψ's are determined recursively. Uniqueness of these polynomials is imposed by setting the coefficient of the highest degree to 1. Dropping the first index of λ for the sake of simplicity, the multivariate polynomials Ψ of the PCE are defined as follows:

$$\Psi = \{\prod_{j=1}^{N} \psi_{\lambda_j}(\epsilon_j) : \sum_{k=0}^{N} \lambda_k \leq n\}, \tag{7.36}$$

where n indicates the degree of the expansion. The number of multivariate polynomials pertaining a PCE expansion of order n is $\binom{n+N}{n}$. These expansions are suitable in the case where the quantity of interest exhibits a smooth dependence on the input parameters. However, to avoid Gibbs phenomenon, other expansions are considered. To overcome this limitation of the PCE expansion, the random inputs and the solution can be decomposed using Haar expansion [**?**].

7.5.1.2 Haar expansion

Given a random variable ϵ, we note p(x) the probability that $\epsilon < x$, where x is a real number. The function p(x) is assumed to be a continuous strictly increasing function of x. Based on these assumptions, there exists only one real x such that p(x) = y, where $0 \leq y \leq 1$. We note F = p^{-1}(y) and we introduce the Haar function ψ^h:

$$\psi^h(y) = \begin{cases} 1, & \text{if } 0 \leq y \leq 0.5 \\ -1, & \text{if } 0.5 \leq y \leq 1 \\ 0, & \text{otherwise.} \end{cases}$$

The Haar function is the mother wavelet that generates the wavelet family:

$$\psi_{j,k}(y) = 2^{\frac{j}{2}} \psi^h(2^j y - k), \tag{7.37}$$

where j is scale index and k is a space index. W is the space gathering the functions $\psi_{j,k}$, where $j = 0, ..., n$ and $k = 0, ..., 2^j - 1$ with n a user-defined index indicating the resolution of the expansion. Any random process $X(\epsilon)$ with finite variance can be approximated arbitrary well by the following expression:

$$X = \bar{X} + \sum_{j=0}^{n} \sum_{k=0}^{2^j - 1} X_{j,k} \psi_{j,k}. \tag{7.38}$$

The values of $X_{j,k}$ are determined by a scalar projection of X on the functions $X_{j,k}$:

$$X_{j,k} = \int_0^1 X\left(F(y)\right) \psi_{j,k}(y) dy. \tag{7.39}$$

For the sake of simplicity, the two indices j and k are concatenated using an integer $\lambda = 2^k + j$. Note ∇ the set of integers λ, $\nabla \equiv \{\lambda, j = 0, .., n, k = 2^j - 1\}$. Noting $W_0 = 1$, and $\nabla_0 \cup 0$, Equation (7.37) can be rewritten as:

$$X = \sum_{\lambda \subset \nabla_0} X_\lambda \psi_\lambda. \tag{7.40}$$

Considering now a multi-dimensional random process, with multi-dimensional index $\boldsymbol{\lambda} = [\lambda_1, ..., \lambda_N]$ the univariate sequence W can be modified to multivariate one noted W^N:

$$W^N = \{\prod_{k=1}^{N} W_{\lambda_k}(\epsilon_k) : \sum_{k=1}^{N} |\lambda_k| \leq n\}, \tag{7.41}$$

where n indicates the level of resolution. The Haar expansions, intrinsically discontinuous, needs a high level of resolution to capture continuous distributions. To overcome this issue, it is possible to use Wavelet expansion.

7.5.1.3 Wavelet expansion

The wavelet expansion uses both continuous and discontinuous functions. Firstly, a space V^{N_0}, containing polynomials of order equal or smaller than N_0 is constructed. The cardinal of that space is $N_0 + 1$. Typically, the scaled Legendre polynomials $[\phi_0, ..., \phi_{N_0+1}]$ are used as a basis of V^{N_0}. Then discontinuous orthonormal functions $[\psi_0, ..., \psi_{N_0}]$ are built as follows :

(i) Two sets of polynomials $p_i(y)$ and $q_i(y)$ are defined:

$$\begin{cases} p_i(y) = y^i \\ q_i(y) = \begin{cases} p_i(y), & \text{if } 0 \leq y \leq 0.5 \\ -p_i(y), & \text{if } 0.5 \leq y \leq 1. \end{cases} \end{cases}$$

(ii) Each polynomial $q_i(y)$ is orthogonalised with respect to all the function $p_i(y)$ to obtain a new set of polynomials $[\tilde{q}_0(y), ..., \tilde{q}_{N_0+1}(y)]$;

(iii) Using a Gram-Schmidt scheme, the set of polynomials $[\tilde{q}_0(y), ..., \tilde{q}_{N_0+1}(y)]$ are orthogonalised with respect to each other to give a set of polynomials $[r_0(y), ..., r_{N_0+1}(y)]$;

(iv) The set of polynomials $[r_0(y), ..., r_{N_0+1}(y)]$ is normalised to obtain the functions $[\psi_0(y), ..., \psi_{N_0+1}(y)]$.

Recall that by construction, the functions ψ_i and ϕ_j are orthogonal. Define the functions ψ_{il}^k as:

$$\psi_{ik}^j = 2^{\frac{j}{2}} \psi_i(2^j y - k). \tag{7.42}$$

Similarly as Equation (7.38), a given stochastic process can be expanded using Wavelet expansion:

$$X(y) = \sum_{i=0}^{N_0} X_i \phi_i(y) + \sum_{i=0}^{N_0} \sum_{j=0}^{n} \sum_{k=0}^{2^j-1} X_{i+(N_0+1)(2^j+k)} \psi_{il}^k(y). \tag{7.43}$$

The coordinates X_λ, where lambda is comprised between 0 and $2^{n+1}(N_0 + 1) - 1$, are obtained by projection of X along the functions ψ and ϕ. More information can be found about the underlying mathematical theory of these expansions in the following references .

7.5.2 Algebra

Most basic operation such as multiplication or division implemented in most of the programming languages cannot be trivially transposed when one wants to do some the same operations with random variables. Each deterministic quantity becomes a random variable that belongs to a space $\mathrm{H} = [\Psi_1, ..., \Psi_\zeta]$. When two random variables are multiplied, it is likely that the product will not belong on the space H. Thus, this product has to be projected back to H. Let us write a random variable A in the space H as $[a_1, ..., a_\zeta]$ and B as $[b_1, ..., b_\zeta]$. The result of the product $\mathrm{A} \hat{\cdot} \mathrm{B}$ is noted $\mathrm{C} = [c_1, ..., c_\zeta]$. This operation can be written as:

$$\left(\sum_{\alpha=1}^{\zeta} a_\alpha \Psi_\alpha \right) \left(\sum_{\beta=1}^{\zeta} b_\beta \Psi_\beta \right) = \sum_{\gamma=1}^{\zeta} c_\gamma \Psi_\gamma. \tag{7.44}$$

The coefficients of C follows by doing a Galerkin projection of (7.44):

$$\sum_{\alpha=1}^{\zeta}\sum_{\beta=1}^{\zeta} a_\alpha b_\beta < \Psi_\alpha \Psi_\beta, \Psi_\gamma >= c_\gamma < \Psi_\gamma, \Psi_\gamma > . \tag{7.45}$$

If we note $C_{\alpha\beta\gamma} = \frac{<\Psi_\alpha\Psi_\beta,\Psi_\gamma>}{<\Psi_\gamma,\Psi_\gamma>}$, Eq. 7.45 becomes:

$$\sum_{\alpha=1}^{\zeta}\sum_{\beta=1}^{\zeta} C_{\alpha\beta\gamma} a_\alpha b_\beta = c_\gamma. \tag{7.46}$$

Thus, the multiplication of two random variables requires the introduction of the third-order tensor $C_{\alpha\beta\gamma}$. The division of two random variables calls for more expensive computations. Indeed, a Galerkin projection is not suitable here. The division of A and B showed that A $\hat{\div}$ B can be transformed as A = D $\hat{\cdot}$ B where the coordinates of D in H have to be determined. These unknowns are found by solving a linear system FD = A $F_{\beta\gamma} = \sum_{\alpha=0}^{\zeta} b_\alpha D_{\alpha\beta\gamma}$.

7.5.3 Derivation of $\frac{\partial F_{oqM}}{\partial x_{\epsilon kb}}$

The term $\frac{\partial F_{oqM}}{\partial x_{\epsilon kb}}$ can be derived as:

$$\frac{\partial F_{oqM}}{\partial x_{\epsilon kb}} = \delta_{qk} N_{b,M} \delta_{\epsilon o} \tag{7.47}$$

7.5.4 Derivation of $\frac{\partial P_{\iota iJ}}{\partial F_{oqM}}$

The term $P_{\iota iJ}$ can be written as:

$$P_{\iota iJ} = C_{\alpha\beta\iota} F_{\alpha iK} S_{\beta KJ} \tag{7.48}$$

Consequently, the term $\frac{\partial P_{\iota iJ}}{\partial F_{oqM}}$ can be written as follows:

$$\frac{\partial P_{iJ}}{\partial F_{oqM}} = C_{\alpha\beta\iota}(\frac{\partial F_{\alpha iK}}{\partial F_{oqM}} S_{\beta KJ} + F_{\alpha iK}\frac{\partial S_{\beta KJ}}{\partial F_{oqM}}) \tag{7.49}$$

7.5.5 Derivation of $\frac{\partial F_{\alpha iK}}{\partial F_{oqM}}$

The term $\frac{\partial F_{\alpha iK}}{\partial F_{oqM}}$ can be written as:

$$\frac{\partial F_{\alpha iK}}{\partial F_{oqM}} = \delta_{iq}\delta_{\alpha o}\delta_{KM} \tag{7.50}$$

7.5.6 Derivation of $\frac{\partial S_{\beta KJ}}{\partial F_{oqM}}$

The term $S_{\beta KJ}$ can be written as follows:

$$S_{\beta KJ} = C_{\gamma\zeta\beta}(2\mu_\gamma E_{\zeta KJ} + \lambda_\gamma E_{\zeta XX}\delta_{KJ}) \tag{7.51}$$

The term $\frac{S_{\beta KJ}}{\partial F_{oqM}}$ can be written as:

$$\frac{\partial S_{\beta KJ}}{\partial F_{oqM}} = \frac{\partial S_{\beta KJ}}{\partial E_{\psi RS}} \frac{\partial E_{\psi RS}}{\partial F_{oqM}} \tag{7.52}$$

Using the symmetry of the spatial indices of the tensor $\boldsymbol{E}$, the term $\frac{\partial S_{\beta KJ}}{\partial E_{\psi RS}}$ can be written as:

$$\frac{\partial S_{\beta KJ}}{\partial E_{\psi RS}} = C_{\gamma\zeta\beta}(\mu_\gamma \delta_{\zeta\psi}(\delta_{KR}\delta_{JS} + \delta_{SK}\delta_{JR}) + \lambda_\gamma \delta_{\zeta\psi}\delta_{RS}\delta_{KJ}) \tag{7.53}$$

This equation can further be simplified as:

$$\frac{\partial S_{\beta KJ}}{\partial E_{\psi RS}} = C_{\gamma\psi\beta}(\mu_\gamma (\delta_{KR}\delta_{JS} + \delta_{SK}\delta_{JR}) + \lambda_\gamma \delta_{RS}\delta_{KJ}) \tag{7.54}$$

The last term to determine is $\frac{\partial E_{\psi RS}}{\partial F_{oqM}}$. The expression of $E_{\psi RS}$ is:

$$E_{\psi RS} = \frac{1}{2} C_{\chi\varphi\psi}(F_{\chi nR}F_{\varphi nS} - \delta_{RS}\delta_{1\psi}) \tag{7.55}$$

Consequently, the term $\frac{\partial E_{\psi RS}}{\partial F_{oqM}}$ can be written as:

$$\frac{\partial E_{\psi RS}}{\partial F_{oqM}} = \frac{1}{2} C_{\chi\varphi\psi}(\delta_{nq}\delta_{\chi o}\delta_{RM}F_{\varphi nS} + \delta_{nq}\delta_{\varphi o}\delta_{SM}F_{\chi nR}) \tag{7.56}$$

This formula can be simplified as:

$$\frac{\partial E_{\psi RS}}{\partial F_{oqM}} = \frac{1}{2}(C_{o\varphi\psi}\delta_{RM}F_{\varphi qS} + C_{\chi o\psi}\delta_{SM}F_{\chi qR}) \tag{7.57}$$

As a result, the term $\frac{\partial S_{\beta KJ}}{\partial F_{oqM}}$ can be expressed as:

$$\frac{\partial S_{\beta KJ}}{\partial F_{oqM}} = \frac{1}{2} C_{\gamma\psi\beta}\left(\mu_\gamma\left(\delta_{KR}\delta_{JS} + \delta_{SK}\delta_{JR}\right) + \lambda_\gamma \delta_{RS}\delta_{KJ}\right)\left(C_{o\varphi\psi}\delta_{RM}F_{\varphi qS} + C_{\chi o\psi}\delta_{SM}F_{\chi qR}\right) \tag{7.58}$$

7.5.7 Derivation of $\frac{\partial P_{\iota iJ}}{\partial F_{oqM}} \frac{\partial F_{oqM}}{\partial x_{\epsilon kb}}$

Using Equation (7.47), the term $\frac{\partial P_{\iota iJ}}{\partial F_{oqM}} \frac{\partial F_{oqM}}{\partial x_{\epsilon kb}}$ can be written as:

$$\frac{\partial P_{\iota iJ}}{\partial F_{oqM}} \frac{\partial F_{oqM}}{\partial x_{\epsilon kb}} = \frac{\partial P_{\iota iJ}}{\partial F_{oqM}} \delta_{qk} N_{b,M} \delta_{\epsilon o} = \frac{\partial P_{\iota iJ}}{\partial F_{\epsilon kM}} N_{b,M} \tag{7.59}$$

This equation can be further expanded as:

$$\begin{gathered}\frac{\partial P_{\iota iJ}}{\partial F_{\epsilon kM}} N_{b,M} = C_{\alpha\beta\iota}(\frac{\partial F_{\alpha iK}}{\partial F_{\epsilon kM}} S_{\beta KJ} + F_{\alpha iK}\frac{\partial S_{\beta KJ}}{\partial F_{\epsilon kM}})N_{b,M} = \\ C_{\alpha\beta\iota}\left(\delta_{ik}\delta_{\alpha\epsilon}S_{\beta MJ} + F_{\alpha iK}\frac{1}{2}C_{\gamma\psi\beta}(\mu_\gamma\left(\delta_{KR}\delta_{JS}+\delta_{SK}\delta_{JR}\right)+\lambda_\gamma\delta_{RS}\delta_{KJ})\left(C_{\epsilon\varphi\psi}\delta_{RM}F_{\varphi kS}+C_{\chi\epsilon\psi}\delta_{SM}F_{\chi kR}\right)\right)N_{b,M} = \\ C_{\alpha\beta\iota}(\delta_{ik}\delta_{\alpha\epsilon}S_{\beta MJ} + F_{\alpha iK}\frac{1}{2}C_{\gamma\psi\beta}\mu_\gamma\delta_{KR}\delta_{JS}C_{\epsilon\varphi\psi}\delta_{RM}F_{\varphi kS}+ \\ F_{\alpha iK}\frac{1}{2}C_{\gamma\psi\beta}\mu_\gamma\delta_{KR}\delta_{JS}C_{\chi\epsilon\psi}\delta_{SM}F_{\chi kR}+ \\ F_{\alpha iK}\frac{1}{2}C_{\gamma\psi\beta}\mu_\gamma\delta_{SK}\delta_{JR}C_{\epsilon\varphi\psi}\delta_{RM}F_{\varphi kS}+ \\ F_{\alpha iK}\frac{1}{2}C_{\gamma\psi\beta}\mu_\gamma\delta_{SK}\delta_{JR}C_{\chi\epsilon\psi}\delta_{SM}F_{\chi kR} + F_{\alpha iK}\frac{1}{2}C_{\gamma\psi\beta}\lambda_\gamma\delta_{RS}\delta_{KJ}C_{\epsilon\varphi\psi}\delta_{RM}F_{\varphi kS}+ \\ F_{\alpha iK}\frac{1}{2}C_{\gamma\psi\beta}\lambda_\gamma\delta_{RS}\delta_{KJ}C_{\chi\epsilon\psi}\delta_{SM}F_{\chi kR})\end{gathered} \tag{7.60}$$

Simplifying the Kronecker indices, the expression (7.60) becomes:

$$\begin{gathered}\frac{\partial P_{\iota iJ}}{\partial F_{\epsilon kM}} N_{b,M} = C_{\alpha\beta\iota}(\frac{\partial F_{\alpha iK}}{\partial F_{\epsilon kM}} S_{\beta KJ} + F_{\alpha iK}\frac{\partial S_{\beta KJ}}{\partial F_{\epsilon kM}})N_{b,M} = \\ C_{\alpha\beta\iota}\left(\delta_{ik}\delta_{\alpha\epsilon}S_{\beta MJ} + F_{\alpha iK}\frac{1}{2}C_{\gamma\psi\beta}(\mu_\gamma\left(\delta_{KR}\delta_{JS}+\delta_{SK}\delta_{JR}\right)+\lambda_\gamma\delta_{RS}\delta_{KJ})\left(C_{\epsilon\varphi\psi}\delta_{RM}F_{\varphi kS}+C_{\chi\epsilon\psi}\delta_{SM}F_{\chi kR}\right)\right)N_{b,M} = \\ C_{\alpha\beta\iota}(\delta_{ik}\delta_{\alpha\epsilon}S_{\beta MJ} + F_{\alpha iM}\frac{1}{2}C_{\gamma\psi\beta}\mu_\gamma C_{\epsilon\varphi\psi}F_{\varphi kJ}+ \\ F_{\alpha iK}\frac{1}{2}C_{\gamma\psi\beta}\mu_\gamma\delta_{JM}C_{\chi\epsilon\psi}F_{\chi kK}+ \\ F_{\alpha iK}\frac{1}{2}C_{\gamma\psi\beta}\mu_\gamma\delta_{JM}C_{\epsilon\varphi\psi}F_{\varphi kK}+ \\ F_{\alpha iM}\frac{1}{2}C_{\gamma\psi\beta}\mu_\gamma C_{\chi\epsilon\psi}F_{\chi kJ} + F_{\alpha iJ}\frac{1}{2}C_{\gamma\psi\beta}\lambda_\gamma C_{\epsilon\varphi\psi}F_{\varphi kM}+ \\ F_{\alpha iJ}\frac{1}{2}C_{\gamma\psi\beta}\lambda_\gamma C_{\chi\epsilon\psi}F_{\chi kM})\end{gathered} \tag{7.61}$$

Using the symmetry of the third-order tensor C, the expression (7.60) becomes:

$$\begin{gathered}\frac{\partial P_{\iota iJ}}{\partial F_{\epsilon kM}} N_{b,M} = C_{\alpha\beta\iota}(\frac{\partial F_{\alpha iK}}{\partial F_{\epsilon kM}} S_{\beta KJ} + F_{\alpha iK}\frac{\partial S_{\beta KJ}}{\partial F_{\epsilon kM}})N_{b,M} = \\ C_{\alpha\beta\iota}(\delta_{ik}\delta_{\alpha\epsilon}S_{\beta MJ} + F_{\alpha iM}C_{\gamma\psi\beta}\mu_\gamma C_{\epsilon\varphi\psi}F_{\varphi kJ}+ \\ F_{\alpha iK}C_{\gamma\psi\beta}\mu_\gamma\delta_{JM}C_{\epsilon\varphi\psi}F_{\varphi kK}+ \\ F_{\alpha iJ}C_{\gamma\psi\beta}\lambda_\gamma C_{\epsilon\varphi\psi}F_{\varphi kM})\end{gathered} \tag{7.62}$$

CHAPTER 8

FSI simulations

8.1 Installation

To run FSI (fluid-structure interaction) simulations, the packages preCICE, OpenFOAM adapter (both provided by preCICE code developers), and OpenFOAM must be installed, see https://precice.org/quickstart.html. If runing on Ubuntu, the version should be at least 20.04 for full functionality.

To compile MuPhiSim with FSI, MuPhiSim needs to be configured with

```
make -j np TYPE=SEQUENTIAL/PARALLEL PRECICE_DIR=path/to/precice-include
```

where np is the number of processors used for compilation.

Running an FSI simulation with MuPhiSim requires three entities:

- A preCICE configuration file, usually named `precice-config.xml`.
- An OpenFOAM configuration folder
- An input file for MuPhiSim

It is noted that each problem (fluid and solid) can be run independently without any interaction. The communication between to these problems is carried out by preCICE whose options given in the configuration file.

8.2 OpenFOAM

OpenFOAM is an open-source computational fluid dynamics software [11]. It can solve complex fluid flows involving chemical reactions, heat transfer, turbulence, acoustics, electromagnetics, and solid mechanics.

To run an OpenFOAM simulation, several input files should be configured, located across three directories (`0/`, `constant/` and `system/`). The outputs are generated in .foam format but can easily be converted to VTK format using the shell command **foamToVTK**.

8.3 preCICE

preCICE (precise Code Interaction Coupling Environment) is a C++ open-source library for partitioned multi-physics simulations in which it couples existing solvers or programs that solve a system of equations for only a part of the complete simulation [12]. This approach is opposed to a monolithic one, in which the same software has to simulate the complete physics involved in the whole domain. The main advantage of the partitioned approach is that it can combine different combinations of components that are best suited to use for each subdomain. It also allows existing components to reused and thus to reduce the time to solution. PreCICE is a library: each solver used needs to call preCICE. Hence, it follows a minimally invasive approach to the existing coupled codes and allows a great flexibility thanks to a high-level API, (Application Programming Interface). Thus, even complex simulations can have an acceptable computational calculation time.

A black-box coupling approach is followed by preCICE. Therefore, preCICE considers the coupled solvers as systems which can only be viewed in terms of their inputs and outputs. Thus, only minimal information about these black boxes is available and preCICE does not have any knowledge of their inner workings. The advantages of such an approach are manifold. Firstly, coupling a new code becomes easy as only little information needs to be provided. Secondly, it is simple to exchange participants in a simulation, i.e. use a solver instead of an other.

In practical terms, this means that the coupling algorithm will only use the input and output values of a coupled solver, not their derivatives. For instance, preCICE will read the displacement data of a structural mechanics FEM code, but neither its Jacobian matrix nor the shape function.

As shown in Figure 13.1, preCICE can couple several solvers, depending on the type of simulation desired.
Linking OpenFOAM to preCICE is simple as preCICE includes a ready-to-use adapter. This adapter is a plug-in for OpenFOAM [15], i.e. a downloadable folder that makes use of the OpenFOAM functionalities.

8.4 Linking OpenFOAM and MuPhiSim using preCICE

8.4.1 Framework

To couple the *MuPhiSim*,the preCICE API has to be implemented. It requires the participant's name, the preCICE configuration file's name and the rank and number of solver processes using preCICE. Before entering the time loop, some FSI configuration settings have to be made: retrieve the vector **boundary_nodes** from the input file which contains the coordinates of the vertices of the coupled surface and supply them to preCICE to define the coupling mesh.

Once the preCICE interface is set up, the **initialize** function is called. This sets up data structures of preCICE, establishes communication channels, and returns the maximum timestep size the solver should use next.

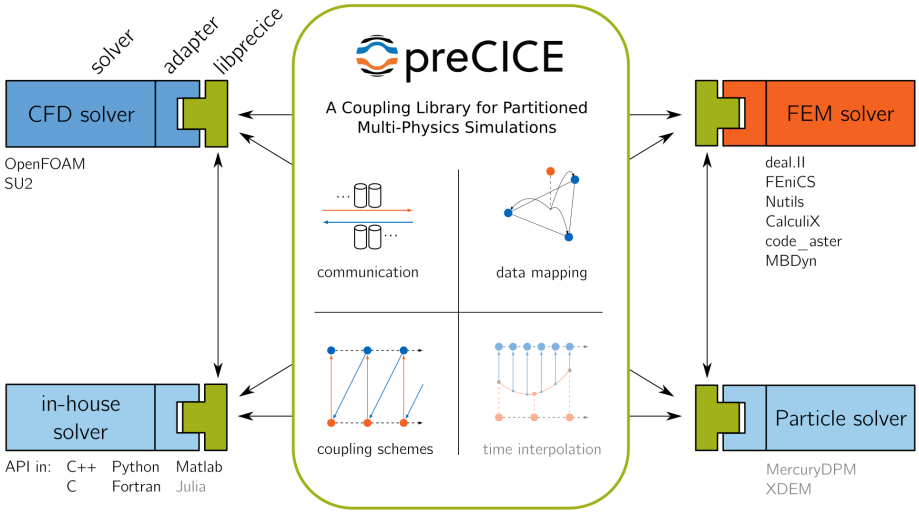


Figure 8.1: Diagram representing the working principle of preCICE [14]
preCICE can couple different solvers to simulate complex multi-physics problems. For instance, the software can couple a CFD solver (CFD stands for Computational Fluid Dynamics), like OpenFOAM, and a Finite Element Method (FEM) solver, like FEniCS or CalculiX.

The implicit dynamic solver subcycles until a tolerance threshold for the norm of the residual is reached. Thus, it is not necessary to write data to or read data from preCICE at every iteration. To avoid unnecessary calls, preCICE offers two functions, **isReadDataAvailable** and **isWriteDataRequired**. When these functions are true, MuPhiSim can access coupling data.

Moreover, preCICE itself allows the user to choose between both implicit and explicit coupling - see below. For implicit coupling, MuPhiSim should be able to go back in time when requested by preCICE, as long as a convergence threshold is not reached. To that end, iteration checkpoints are placed inside the solver time loop. An iteration checkpoint contains all the information necessary to reload a previous state. preCICE tells MuPhiSim when to write and read these checkpoints using **isActionRequired** and **markActionFulfilled**. These two methods take a string argument to reference the action. For implicit coupling, **actionReadIterationCheckpoint** and **actionWriteIterationCheckpoint** are used. In the first coupling iteration of each time window (i.e. the time step of the coupling simulation), preCICE tells MuPhiSim to write a checkpoint.

As one can see in Figure 8.2, the forces were set in the MuPhiSim code after calling the function **NeumannBCManagement()** as this function resets the forces to zero before calculating them. To limit the number of lines changed in the MuPhiSim code, forces are set from the preCICE data after this function.

Once convergence is achieved for the MuPhiSim implicit solver after one or more iterations of the Newmark loop, **advance** must be called to advance the coupling. As an argument, the solver's last timestep size is given and the function returns the next maximum timestep size that MuPhiSim can use. Coupling schemes, communication and acceleration are hidden behind the library call **advance** [12,13]. Moreover, in case of an implicit coupling scheme, the actual coupling

convergence measure is computed in advance.

Furthermore, preCICE instructs MuPhiSim to read a checkpoint for every coupling iteration in which the coupling has not converged. If **isActionRequired(actionReadIterationCheckpoint)** is true, the previous state saved is reloaded. An advantage of preCICE is that even with this adapted code, the explicit coupling for the FSI simulation still works. No code modification is required. In case of explicit coupling, both actions reading and writing iteration checkpoints return false. At the end of a simulation, the **finalize** function closes communication channels and releases the preCICE data structures.

Figure 8.2 summarizes these additions (represented in red). The initial structure of the solver has been simplified so as not to overload the figure."w" stands for **actionWriteIterationCheckpoint** and "r" for **actionReadIterationCheckpoint**.

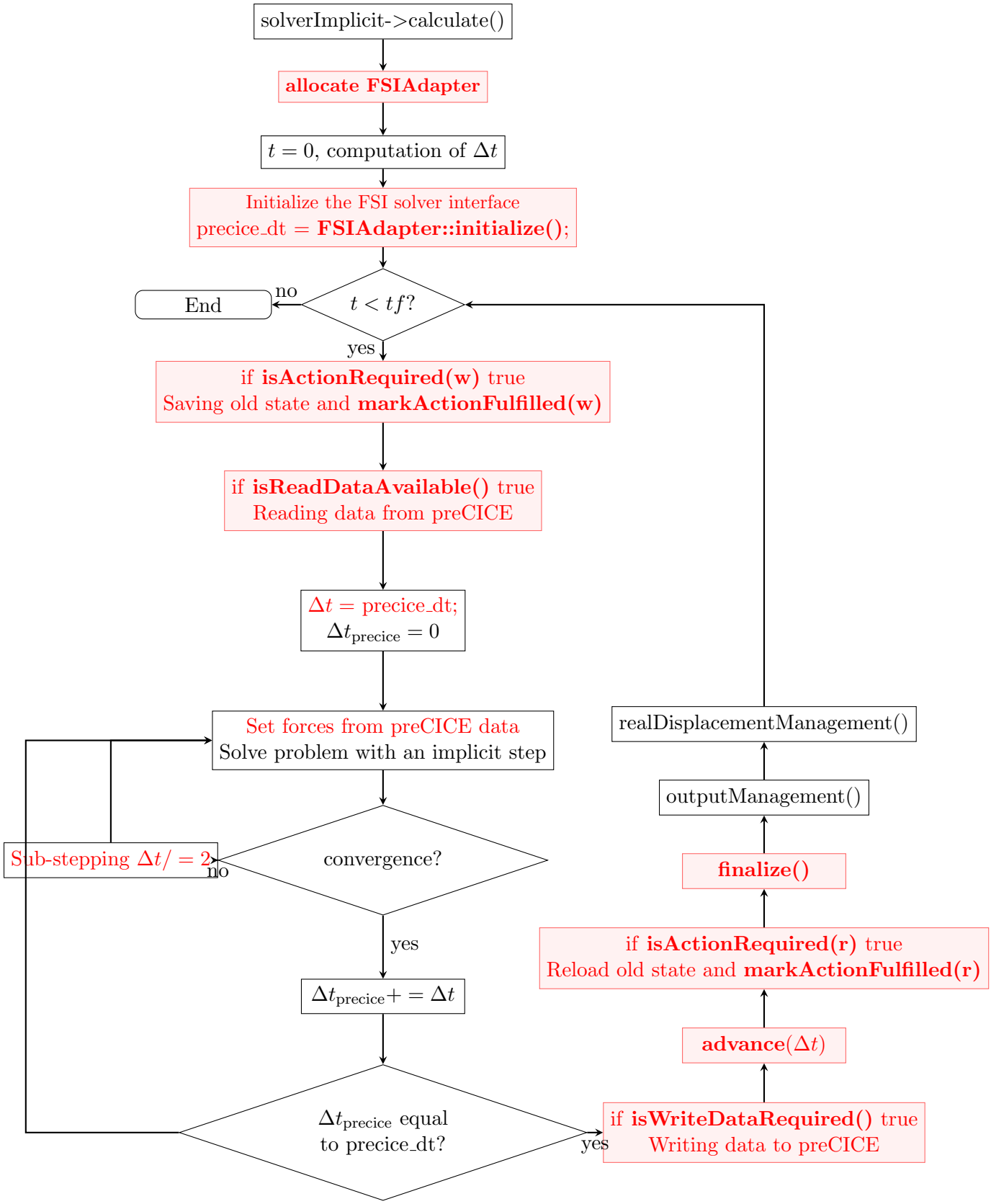


Figure 8.2: Simplified structure of MuPhiSim Implicit Dynamic Solver with the addition of FSI options

If an FSI simulation is needed by the user, they should add to the input file the keyword ***FSI** followed by the list of nodes at the coupling interface by means of the corresponding label in the full list of nodes.

8.4.2 OpenFOAM adapter

For linking OpenFOAM and preCICE, an open-source adapter has been developed and can be downloaded from the preCICE website. However, to use it properly, some steps must be followed :

- Write a `system/preciceDict`
- Set compatible boundary conditions
- Activate the adapter in `system/controlDict`.

The tree of the folder corresponding to the fluid part is shown in Figure 8.3.

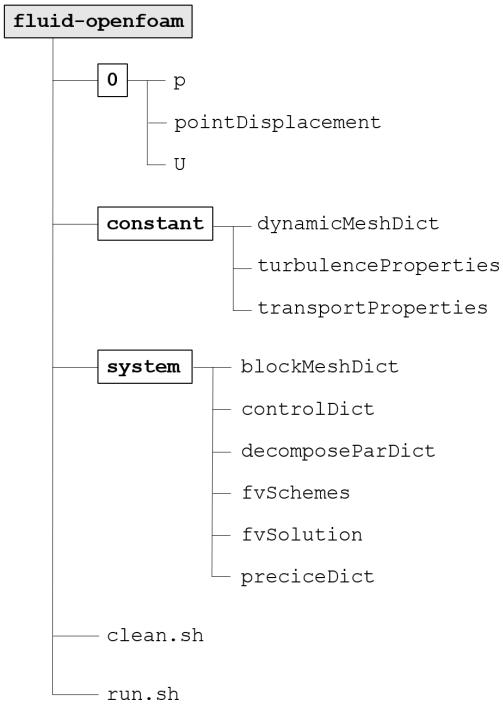


Figure 8.3: Tree of the `fluid-openfoam` folder
All the files participate in the configuration of the OpenFOAM adapter except the files clean.sh and run.sh which respectively cleans the folder and runs the simulation for the fluid.

`preciceDict` is an OpenFOAM dictionary and the adapter's configuration file. A list of the names of the OpenFOAM boundary patches that are participating in the coupled simulation is specified in this file. These need also to be defined in the files included in the `0/` directory.

The boundary conditions for the interface are set in the `0/` folder and the type of solver is specified in `constant/dynamicMeshDict`.

To load the adapter, the `system/controlDict` configuration file must direct the solver to use the **preciceAdapterFunctionObject** function object. This function is part of the **libpreciceAdapterFunctionObject.so** shared library, available with the OpenFOAM adapter.

Please refer to **https://precice.org/adapter-openfoam-config.html** to get all the configuration details.

8.4.3 The role of preCICE

Once the links between preCICE and the two solvers are established, all that remains is to configure the coupling algorithm so that the FSI simulation can be done.

8.4.3.1 Configuration file

Before launching a coupling simulation, preCICE must be configured at runtime via an .xml file named `precice-config.xml`. This file defines all coupling simulation choices made by the user.

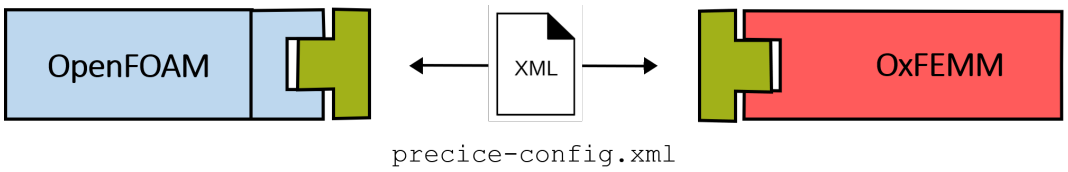


Figure 8.4: Diagram of the coupled solvers
`precice-config.xml` *configurates preCICE, which is represented by the green blocks.*

The easiest way to write a configuration file is to use those given in the examples in either in the **quickstart** or **perpendicular_flap** folders or in the preCICE tutorials.

First, the dimensions of the coupling simulation is set : `<solver-interface dimensions="2">`. This value agrees with the physical dimension of the simulation, i.e. the number of coordinates forces and displacements vertex have. However, OpenFOAM only supports 3D simulation and MuPhiSim is used in 2D. Hence, the OpenFOAM adapter maps from 3D to 2D as the `precice-config.xml` sets the preCICE dimension to 2D. As already mentioned, we recommend simulating mainly bidirectional problems as communication with preCICE is particularly expensive for volume coupled problems.

Then, the preCICE configuration file is structured in the following five sections :

- `<data .../>`: defines the coupling data values the participants exchange. For the example case, these are displacements and forces. Once these fields are defined, MuPhiSim and OpenFOAM use the preCICE API to access them.
- `<mesh .../>`: defines the interface coupling meshes by giving them an ID transmitted to the solvers via the preCICE API.
- `<participant .../>`: the solvers that participate in the coupled simulation need a participant definition in the `xml` file. Each participant can either provide the mesh by defining the coordinates or receive the mesh coordinates from another participant who defines them.

As the coordinates of the boundary nodes are defined in the MuPhiSim input file, it seems natural that the fluid participant should use the solid mesh. Thus, a data mapping is defined between both participants - see next section.

- `<m2n .../>`: defines a communication channel for the two participants to exchange data.
- `<coupling-scheme .../>`: where the two participants exchange data, the simulation time, the coupling scheme wanted, the coupling time window length (i.e. the coupling time step) are defined.

preCICE provides a tool to easily understand the `precic-config.xml` file with a simple shell command. This tool interprets the given configuration file and visualizes it as a graph, represented in Figure 8.5. See at **https://precice.org/tooling-config-visualization.html** for further information.

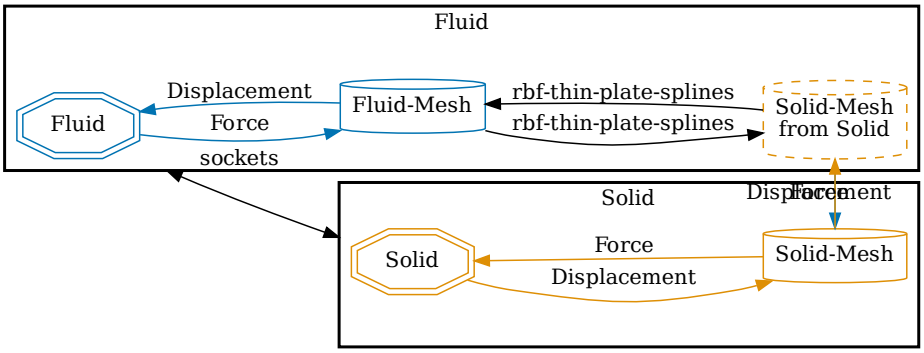


Figure 8.5: Graph showing the configuration of preCICE for the example studied
This schematic view of the `precic-config.xml` *file helps to understand how works the FSI simulation.*

8.4.3.2 Mapping configuration

The FSI simulation requires the communication of data from the coupling surface of the deformable body to the coupling surface of the fluid. Thus, coupling the two participants at a common interface generally means mapping data between surface meshes which do not match. To deal with this issue, preCICE provides a choice of data mapping methods to map coupling data from one mesh to the other [12,13]. Each coupled variable can be mapped in either consistent or conservative form.

As part of its black box approach, preCICE only uses nodal values, values at mesh vertices. Consider two meshes, a fine and a coarse grid. If the value at coarse nodes is the same as the value at the corresponding fine node, the type of the mapping is consistent. A consistent mapping exactly reproduces constant functions at the coupling surface. It is usually applied to values such as densities and fluxes. In the case studied, the displacement data is mapped in a consistent form.

Conservative mappings preserve integral values. The value at a coarse node is computed as a sum of the corresponding fine nodes, such that the total coupling value on the coarse and fine mesh is

the same. This type of mapping is applied to forces in the example case.

preCICE provides three mapping methods :

- Nearest-Neighbor mapping is a first-order method that only requires vertex position information and works locally. As represented in Figure 8.6, a data point in the target mesh is assigned the value of the data point in the source mesh that is the closest to its position.

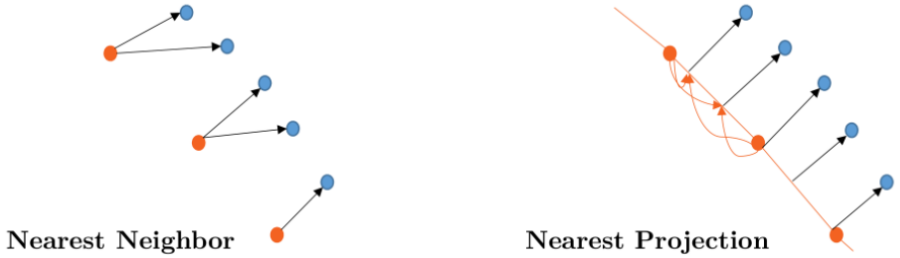


Figure 8.6: Nearest-Neighbor and Nearest-Projection mappings
These two mapping configuration are different ways to exchange date between a source and a target mesh.

- Nearest-Projection mapping is a second-order method based on projections of data points of the target mesh to mesh elements of the source mesh and a linear interpolation scheme in these elements. This method is relatively fast and numerically superior to nearest-neighbor. However, using this method requires mesh connectivity information to be defined.
- Radial Basis Function (RBF) mapping is a method that requires no topological information and works well on general non-matching meshes. This mapping uses radially-symmetric basis functions centered on each vertex of the source mesh. Many choices for the basis functions are implemented in preCICE and are listed in [13].

For the coupling simulation studied, the radial basis function has been used with a `thin-plate-splines` type. Indeed, as nearest-projection, radial basis function mapping behaves as a second-order method, but without the need to define mesh connectivity information. Moreover, basis function with global support are easier to configure as only a few parameters need to be set.

8.4.3.3 Communication

Communication between participants works point-to-point [12,13]. preCICE analyses the mesh decomposition of both participants and constructs only local communication channels when needed. Therefore, mesh connectivity is often not necessary: a cloud of vertices as coupling mesh is sufficient for most of the numerical methods that preCICE offers.

8.4.3.4 Coupling schemes: explicit/implicit and serial/parallel

One of the main assets of preCICE is its adaptability to different schemes. Indeed, preCICE offers a variety of coupling schemes : either serial or parallel and either explicit or implicit.

The user can chose whether to run a serial or a parallel coupling scheme. With a serial scheme, the participants run after one another while with a parallel one the participants run simultaneously. Nothing needs to be changed in the solver code to switch from a serial to a parallel coupling scheme thanks to the high-level API of preCICE.

It is also possible to choose between an explicit and an implicit scheme [13]. Denote x as the forces data and y the displacements data so that

$$\begin{cases} S_1 : x \to y \\ S_2 : y \to x \end{cases} \tag{8.1}$$

where S_1 and S_2 are respectively the mappings of MuPhiSim and OpenFOAM routines.

With an explicit scheme, both participants are only executed once per time window. Used with the serial scheme, the system to solve at every time step is the following :

$$\begin{cases} S_1^n(x^n) = y^{n+1} \\ S_2^n(y^{n+1}) = x^{n+1} \end{cases} \tag{8.2}$$

However, this implies a staggered execution of the two solvers. The parallel scheme avoids this by using old time step values x^n and y^n as an input for both solvers :

$$\begin{cases} S_1^n(x^n) = y^{n+1} \\ S_2^n(y^n) = x^{n+1} \end{cases} \tag{8.3}$$

Implicit coupling schemes iterate over a coupling equation until convergence. This means that the participants are coupled iteratively, repeating each coupling time window until both solvers have converged to the same values. With a serial scheme, the following system is solved for each iteration i :

$$\begin{cases} S_1^n(x_i^{n+1}) = y_{i+1}^{n+1} \\ S_2^n(y_{i+1}^{n+1}) = x_{i+1}^{n+1} \end{cases} \tag{8.4}$$

whereas a parallel scheme computes the new values for a given iteration x^{n+1} and y^{n+1} simultaneously :

$$\begin{cases} S_1^n(x_i^{n+1}) = y_{i+1}^{n+1} \\ S_2^n(y_i^{n+1}) = x_{i+1}^{n+1} \end{cases} \tag{8.5}$$

The iterations of implicit coupling schemes can be reduced by using acceleration techniques that allow the data exchanged to be modified by preCICE. Three different types of acceleration are provided: constant (constant under-relaxation), aitken (adaptive under-relaxation), and various quasi-Newton variants (IQN-ILS aka. Anderson acceleration, IQN-IMVJ aka. generalized Broyden) [12,13]. The acceleration modifies coupling data in advance() to stabilize values by using a linear combination of values from previous iterations. For the case studied, the Anderson acceleration is used as the Quasi-Newton acceleration is recommended for strong interactions.
IMPORTANT : Once the `precic-config.xml` file has been edited, the path to the preCICE configuration file in the FSI configuration settings must be specified in the solverImplicit.cpp file !

8.5 Example

Consider a deformable body in a fluid flow inside a channel. The diagram of this example can be seen in Figure 13.3. The flap is fixed at a rotation point, constituting Dirichlet boundary

conditions. The structure has a density of $\rho_s = 1.0.10^4 kg.m^{-3}$, the Poisson ratio $\nu = 0.3$ and a Young's modulus of $1.0.10^6 kg.m^{-1}.s^{-2}$. The simulation time was set at 2.5 seconds. This is a bidirectional surface coupling. Indeed, communication with preCICE is particularly expensive for volume coupled problems [13].

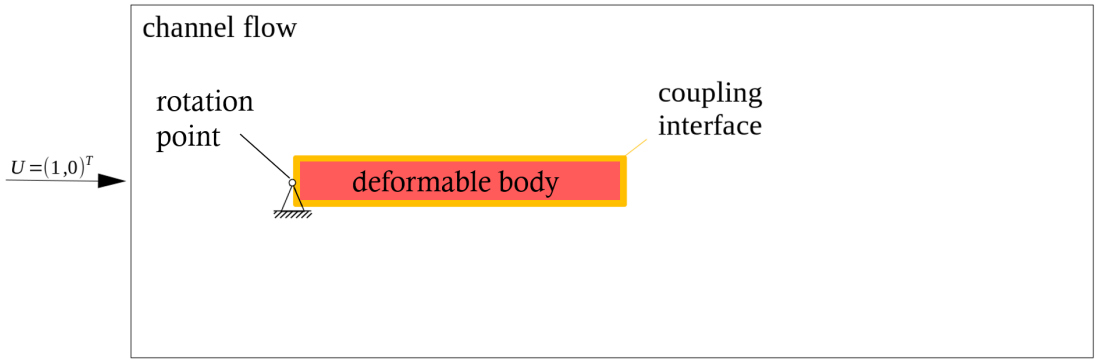


Figure 8.7: Diagram of the case study
The simple example studied is a deformable body in a fluid flow inside a channel. The flap is fixed at a rotation point.

For this example, only FEM nodes and elements are defined. The material keyword used is ***HyperElastic, St-Venant-Kirchhoff**. This is equivalent to linear elastic for small deformation. The type of solver used was **IMPLICIT** (dynamic).

Returning to the example, the angle of attack of the flow must be sufficient to set the solid in motion. This is ensured by the point of rotation being set in the upper left corner of the flap. Boundary conditions were defined in the input file before running the simulation.

preCICE follows a peer-to-peer approach: in order to run a simulation, the user have to start the coupled solvers individually in two different terminals.
Paraview allows the fluid and solid results to be viewed either together or separately. Results can be shown in Figure 8.8 and 8.9.

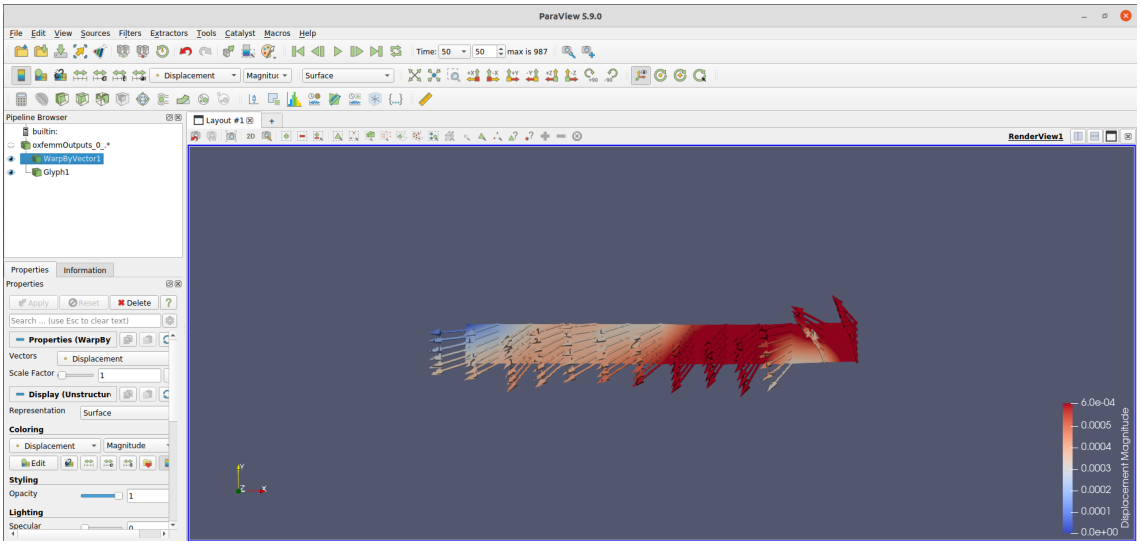


Figure 8.8: Result of the FSI simulation for the deformable solid (screenshot of the Paraview window)
This screenshot represents the 50^{th} *MuPhiSim output file, corresponding to a time simulation of* $0.0619377\,s$

Paraview provides to the user several filters to visualize the results properly. Here, The "Glyph" and "WrapByVector" filters have been used. The former displays the displacements with arrows and the latter enables to deform the coupling data.

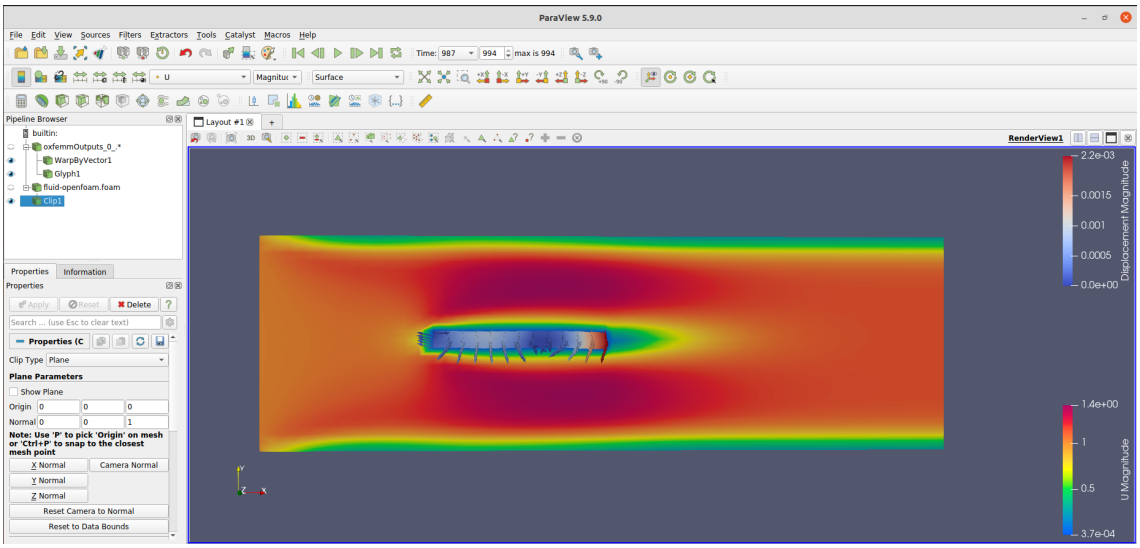


Figure 8.9: Result of the FSI simulation (screenshot of the Paraview window)
This screenshot corresponds to a time simulation of $0.15\,s$

APPENDIX A

ACTIVATION/DEACTIVATION OF ELEMENTS

This chapter includes the element progressive activation/deactivation capabilities in MuPhiSim. Typical examples include analysing fracture and additive manufacturing.
To use activation/deactivation option, the input file must to include the keyword ***ACTIVATION** just after the line that specific the analysis type. All the elements will be defined as active by default. The activation status occurs at the beginning of a load step. The different criteria for activation/deactivation must be defined in the constitutive model (the criteria can be defined in the mechanical or extraDof models; however, this status will affect all the DOF, not just the ones used in that specific model). Note that elements cannot be partially activated. This means that the status of the element can change only from inactive to fully active. That provides a stepped change in the element status. Following this premise, boundary conditions have to be defined as instantaneous.
Unlike other FEM software, where the stiffness matrix and load vectors size is always preserved, in order to decrease the computational cost, the stiffness matrix, residual and deltaU vectors change according with the active DOF (not including Dirichlet DOF) in each step.

APPENDIX B

Tangent modulus of the Neo-Hookean constitutive model

Considering a material that follows a Neo-Hookean constitutive law, the expression of the first Piola-Kirchhoff stress is :

$$P = \frac{\mu}{J^{\frac{2}{3}}}(F - \frac{1}{3}trace(B)F^{-T}) + K_1 J(J-1)F^{-T} \tag{B.1}$$

The expression of the tangent modulus by taking the derivative of P with respect to F is derived by:

$$\begin{aligned}\frac{\partial P}{\partial F} = \mu(F \otimes \frac{\partial(J^{\frac{-2}{ndim}})}{\partial F} + J^{\frac{2}{ndim}}\frac{\partial F}{\partial F} - \frac{1}{ndim}(F^{-T} \otimes \frac{\partial trace(B)}{\partial F} + \\ trace(B)\frac{\partial F^{-T}}{\partial F}) + K_1(F^{-T} \otimes \frac{\partial J(J-1)}{\partial F} + J(J-1)\frac{\partial F^{-T}}{\partial F}).\end{aligned} \tag{B.2}$$

After all calculation are done, one obtains:

$$\begin{aligned}C_{iJkL} = \frac{\mu}{J^{\frac{2}{ndim}}}(\delta_{iK}\delta_{Jl} - \frac{2}{ndim}F^{-1}_{Lk}F_{iJ} + \frac{1}{ndim}traceB(F^{-1}_{Li}F^{-1}_{Jk}) + \\ \frac{-1}{ndim}(\frac{-2}{ndim}F^{-1}_{Lk}trace(B) + F_{kL}F^{-1}_{Ji})) + K_1 J(J-1)(-F^{-1}_{Li}F^{-1}_{Jk}) + (2J^2 - J)(F^{-1}_{Lk}F^{-1}_{Ji}).\end{aligned} \tag{B.3}$$

Bibliography

[1] Arroyo, M. and Ortiz, M. (2006). Local *maximum-entropy* approximation schemes: a seamless bridge between finite elements and meshfree methods. *International Journal for Numerical Methods in Engineering*, 65:2167–2202.

[2] Belytschko, T., Liu, W., and Moran, B. (2000). *Nonlinear Finite Elements for Continua and Structures*. John Wiley & Sons, Ltd.

[3] Dunavant, D. A. (1985). High degree efficient symmetrical Gaussian quadrature rules for the triangle. 21:1129–1148.

[4] Garcia-Gonzalez, D. and Jerusalem, A. (2019). Energy based mechano-electrophysiological model of cns damage at the tissue scale. *Journal of the Mechanics and Physics of Solids*, 125:22 – 37.

[5] Géradin, M. and Rixen, D. (1997). *Mechanical Vibrations: Theory and Application to Structural Dynamics*. Wiley.

[6] Nguyen, V., Rabczuk, T., Bordas, S., and Duflot, M. (2008). Meshless methods: a review and computer implementation aspects. *Mathematics and Computers in Simulation*, 79:763–813.

[7] Novascone, S., Spencer, B., Hales, J., and Williamson, R. (2015). Evaluation of coupling approaches for thermomechanical simulations. *Nuclear Engineering and Design*, 295:910 – 921.

[8] Sukumar, N. and Wright, R. (2007). Overview and construction of meshfree basis functions: from moving least squares to entropy approximants. *International Journal for Numerical Methods in Engineering*, 70:181–205.

[9] Ullah, Z. and Augarde, C. (2013). Finite deformation elasto-plastic modelling using an adaptive meshless method. *Computers and Structures*, 118:39–52.

[10] Zienkiewicz, O., Taylor, R., and Fox, D. (2014). The finite element method for solid and structural mechanics. Butterworth-Heinemann, Oxford, seventh edition edition.

[11] "About OpenFOAM", *OpenFOAM*, https://www.openfoam.com/. Accessed Aug. 2021.

[12] "The preCICE documentation", *preCICE - The Coupling Library*, https://precice.org/docs.html. Accessed Aug. 2021.

[13] Hans-Joachim Bungartz, Florian Lindner, Bernhard Gatzhammer, Miriam Mehl, Klaudius Scheufele, Alexander Shukaev, Benjamin Uekermann, *preCICE – A fully parallel library for multi-physics surface coupling*, Computers & Fluids, Volume 141, 2016, Pages 250-258, ISSN 0045-7930,
https://doi.org/10.1016/j.compfluid.2016.04.003.

[14] David Schneider, "Material when writing about preCICE", *precice.github.io/material/*, https://github.com/precice/precice.github.io/tree/master/material. Accessed Aug. 2021.

[15] Gerasimos Chourdakis, *A general OpenFOAM adapter for the coupling library preCICE*, Master's thesis, Department of Informatics, Technical University of Munich, 2017.