

3MAH : un ensemble de bibliothèques pour analyser le comportement complexe de matériaux hétérogènes

E. Prulière¹, Y. Chemisky², M. R. Guevara³

¹ I2M, Arts et Métiers, etienne.pruliere@ensam.eu

² TIMC, Université Grenoble Alpes, yves.chemisky@univ-grenoble-alpes.fr

³ I2M, Université de Bordeaux, manuel.guevara-garban@u-bordeaux.fr

Résumé —

Nous proposons un ensemble de bibliothèques open-source permettant d'analyser le comportement de matériaux hétérogènes, en intégrant trois principaux outils de simulation complémentaires : une bibliothèque de génération de microstructures virtuelles (microgen), une bibliothèque de simulation par éléments finis (fedoo) et une bibliothèque de comportement de matériaux (simcoon). Une attention particulière est portée à la simulation automatisée d'un grand nombre de cas d'étude (pour la génération de bases de données d'apprentissage, par exemple), et à la portée pédagogique de ces outils (utilisation systématique du langage python).

Mots clés — simulation numérique, open-source, pédagogie interactive

1 Introduction

La simulation numérique est une activité devenue inévitable lors de la conception et tout au long de la vie d'un produit. Les logiciels industriels du marché intègrent des méthodes numériques et des modèles mécaniques avancés. Il y a un double enjeu : (i) pédagogique concernant la maîtrise de ces outils et concepts. La formation des futurs ingénieurs et experts en simulation doit nécessairement inclure des bases solides à la fois sur les concepts fondamentaux en mécanique et les méthodes numériques associées. (ii) scientifique dans la mesure où les chercheurs doivent disposer d'outils ouverts leur permettant de tester la pertinence de nouveaux modèles et méthodes innovantes.

L'essor récent des outils interactifs (langage python, notebooks interactifs) est une opportunité pour faciliter ce processus d'apprentissage, et permet de faciliter l'implémentation de nouvelles fonctionnalités.

L'objectif de 3MAH est de proposer :

- un ensemble de bibliothèques ouvertes pour traiter de manière intégrée l'analyse du comportement de structures, de la définition du domaine et des cas d'étude à l'analyse des résultats.
- des interfaces pédagogiques (notebook interactifs, fonctions à différents niveaux) permettant d'appréhender à la fois la présentation de concepts fondamentaux et la méthodologie de simulation numérique.
- un code open-source permettant de facilement implémenter/tester des nouveaux modèles et/ou algorithmes dans des programmes de recherche en mécanique numérique et fondamentale.
- un outil ayant un bon compromis performances/souplesse d'utilisation/facilité de développement.

Cet objectif ambitieux est réalisé à l'aide d'une API (Application Programming Interface, ou bibliothèque) à plusieurs niveaux : (i) Un socle de fonctions qui permet d'analyser/tester des outils de la mécanique des milieux continus de manière séparée et de réaliser des simulations au niveau du matériau (bibliothèque simcoon) (ii) un solveur éléments finis pour des simulations sur structure et sur VER de matériaux non linéaires hétérogènes (bibliothèque fedoo) (iii) Un ensemble de scripts permettant de générer et mailler de manière automatisée des microstructures numériques (bibliothèque microgen).

2 Simcoon

La bibliothèque Simcoon a pour objectif d'étudier le comportement de matériaux hétérogènes. En ce sens, elle regroupe des fonctions et des logiciels permettant :

- des méthodes multiéchelles : micromécaniques (Mori-Tanaka, schéma autocohérent) et homogénéisation périodique analytique pour une direction. Ces schémas sont adaptés à des simulations de comportements non-linéaires et anisotropes.
- des méthodes permettant de développer des lois de comportement en grandes transformations : Coordonnées matérielles entrainées (base naturelle), repères corotationnels objectifs (repère orthonormés), intégration de la déformation cumulées tensorielles (exacte dans un repère corotationnel dit "logarithmique" entre autres)
- La construction de lois de comportement basée sur la thermodynamique des processus irréversibles. Une méthodologie permettant d'implémenter des lois de comportement non-linéaire des matériaux intégrant des mécanismes dissipatifs pouvant apparaître simultanément est utilisée. A partir de la description thermodynamique du matériau, une forme analytique du système d'EDP à résoudre est proposée (conditions de Kuhn-Tucker et équations d'évolution). Les opérateurs tangents thermomécaniques sont décrits ainsi que les quantités énergétiques (puissance mécaniques et thermiques). Ces formes sont adaptées à une analyse multiéchelle des matériaux hétérogènes.
- des méthodes d'identification de paramètres : heuristiques et basés sur le gradient. Ces méthodes sont adaptées à une grande variété d'essais mécaniques, peuvent être utilisées pour une simulation du comportement du matériau mais également de structures avec fedoo.

La majeure partie de la librairie Simcoon est développée en C++, avec une interface python développée à l'aide de Pybind11 [4] permettant d'exposer les fonctions essentielles en Python et de rendre l'ensemble compatible avec les autres librairies de l'ensemble 3MAH. Les opérations entre objets C++ et objets Python sont optimisés en utilisant directement l'adresse mémoire des objets pour éviter les opérations de copie.

2.1 Exemples de fonctions élémentaires

Un exemple de fonction élémentaire est présenté ci-dessous. Elle permet d'obtenir la matrice de rigidité d'un matériau élastique isotrope ($L = \text{sim.L_iso}(E, \nu, \text{'Enu'})$, en notation de Voigt) sous la forme d'un numpy array 6x6. Une seconde fonction permet d'analyser les symétries d'une matrice de rigidité quelconque ($d = \text{sim.check_symetries}(L)$), et renvoyant sous forme d'un dictionnaire le type de symétrie identifiée 'umat_type' (isotrope, orthotrope, ...) et la liste des paramètres matériaux 'props' correspondant. L'exemple ci-dessous va alors générer un numpy array 6x6 contenant la matrice de rigidité d'un matériau élastique isotrope et renvoyer les paramètres E et ν dans le dictionnaire 'dict_verif' pour vérification (commande `print(dict_verif['props'])`).

```
import numpy as np
from simcoon import simmit as sim

E = 70000.0
nu = 0.3
L = sim.L_iso(E, nu, "Enu")
print(np.array_str(L, precision=2, suppress_small=True))

dict_verif = sim.check_symetries(L)
print(dict_verif['umat_type'])
print(dict_verif['props'])
```

Le résultat de ce script va indiquer *ÉLISO* pour le type de loi de comportement (umat_type) indiquant une symétrie isotrope, et [70000.0 0.3] qui est numpy.array stockant les propriétés matériau (module de Young et coefficient de Poisson pour une symétrie isotrope).

2.2 Exemple de simulation

Le premier exemple concerne l'identification d'une loi de comportement élastoplastique de type Chaboche [1]. Cette loi intègre un écrouissage isotrope non-linéaire et deux écrouissages cinématiques non-linéaires. L'évolution de la fonction d'écrouissage isotrope s'exprime :

$$\dot{H} = b(Q - H) \dot{p} \quad (1)$$

Les deux variables conjuguées d'écoulement cinématique \underline{X}_1 et \underline{X}_2 s'expriment :

$$\dot{X}_1 = \frac{2}{3}C_1\varepsilon^p - D_1\underline{X}_1\dot{p}, \quad \dot{X}_2 = \frac{2}{3}C_2\varepsilon^p - D_2\underline{X}_2\dot{p}, \quad (2)$$

où C_1 et C_2 sont les paramètres d'écoulement et D_1 et D_2 sont les paramètres d'adoucissement. La limite d'écoulement s'exprime alors :

$$|\underline{\sigma} - \underline{X}| - H - \underline{\sigma}^Y = 0, \quad (3)$$

où $\underline{\sigma}^Y$ est la limite d'élasticité initiale du matériau. Les paramètres à identifier dans cet exemple sont respectivement $\underline{\sigma}^Y$, Q , b , C_1 , D_1 , C_2 , D_2 . Trois essais cycliques en traction-compression (à $\pm 1\%$, $\pm 1.5\%$, $\pm 2\%$ de déformation imposée) sont utilisés en considérant un cycle stabilisé. L'algorithme d'identification implémenté dans simcoon est hybride, avec une partie heuristique (algorithme génétique) et déterministe (à gradient de type Levenberg-Marquardt [2]). Le code python utilisé est détaillé ci-dessous, permettant de définir les principaux paramètres de l'algorithme d'identification. La sélection des paramètres à identifier et de la mise en place de la fonction cout nécessite un soin particulier, et un système de fichiers d'entrée permet de renseigner toutes ces informations.

```
import numpy as np
from simcoon import simmit as sim

#Number_of_parameters
n_param = 5
#Number_of_consts
n_consts = 0
#Number_of_files
nfiles = 3

#Number_of_generations
ngen = 20
#Aleatory/Mesh space population : 0=mesh 1=meshlimit 2=random 3=defined
aleaspace = 2
#Space or aleatory population : apop in case of aleatory , spop in case of mesh
apop = 20
#Number of "gradient-based" individual
ngboys = 1
#Max population per subgeneration
maxpop = 10

path_data = dir + '/data'
path_keys = dir + '/keys'
path_results = dir + '/results'
outputfile = 'id_params.txt'
materialfile = 'material.dat'
simul_type = 'SOLVE'

sim.identification(simul_type, n_param, n_consts, nfiles, ngen, aleaspace, apop, ngboys,
maxpop, path_data, path_keys, path_results, materialfile, outputfile)
```

2.3 Example 2

Un deuxième exemple concerne une analyse, à portée pédagogique, de l'influence du type de référentiel corotationnel utilisé pour exprimer la loi de comportement. Trois référentiel corotationnel sont implémentés dans Simcoon, en fonction de l'expression de la vitesse de rotation du repère.

```
import numpy as np
from simcoon import simmit as sim

umat_name = 'ELISO' #5 character code constitutive model
nstatev = 1 #The number of scalar variables required
props = np.array([70000, 0.2, 1.E-5]) #E, nu, alpha
path_data = 'data'
path_results = 'results'
```

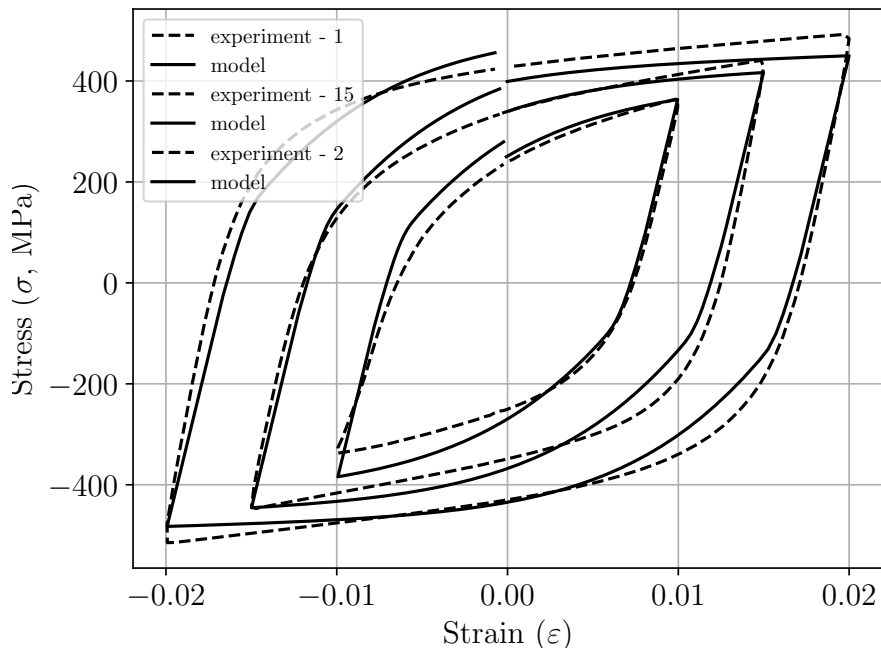


FIGURE 1 – Comparaison essais - modèle identifié.

```

pathfile = 'path.txt'

for i in range(0,2):
    corate_type = i
    outputfile = 'results_ELISO_' + str(i) + '.txt'
    sim.solver(umat_name, props, nstatev, 0., 0., 0., 0,
               corate_type, 'data', 'results', 'path.txt', outputfile)

```

Le code va générer la simulation numérique demandée suivant le chargement renseigné dans le fichier 'path.txt'. Ici, le code teste respectivement les vitesses de rotation de Jaumann ($corate = 0$), Green-Naghdi ($corate=1$) et logarithmique ($corate=2$). En considérant un essai piloté en cisaillement simple (le gradient de la transformation est imposé), les résultats présentés dans la littérature sont retrouvés ici (par exemple dans [3]), notamment les oscillations observées sur les composantes de la déformation cumulée.

3 microgen

La librairie 'microgen' est dédiée à la génération et au maillage de cellules hétérogènes. L'analyse de ces cellules peut se faire ensuite directement en utilisant les outils d'homogénéisation périodique implémentés dans 'fedoo'. 'microgen' repose principalement sur trois librairies : CADQuery (<https://cadquery.readthedocs.io/en/latest/#>) pour la gestion des outils CAO, ainsi que pyvista [5] et gmsh (<https://gmsh.info>) pour le maillage et la visualisation. La librairie CADQuery permet d'utiliser le moteur CAO OpenCascade directement en utilisant des commandes python, ce qui permet de facilement générer des opérations CAO combinées pour la génération de microstructures hétérogènes complexes. Par exemple, 'microgen' permet de simuler différents types de matériaux architecturés, en particulier des surfaces triplement périodiques.



FIGURE 2 – Matériaux architecturés conçus et maillés avec microgen, utilisant un appel de fonctions CADquery et Gmsh.

4 Fedoo

4.1 Généralités

Fedoo est une librairie développée intégralement en python pour résoudre des problèmes mécanique avec la méthode des éléments finis. Des fonctions haut-niveau permettent de résoudre un problème mécanique ou thermique complet avec un script concis et lisible, en suivant les différentes étapes de la méthode éléments finis : formulation faible, maillage, assemblage, post-traitement, Pour une utilisation plus avancée, il est possible d'écrire des nouvelles formulations faibles, de développer des nouveaux éléments (interpolation différenciée pour chaque variable, interpolation géométrique, nombre de points d'intégrations, ...) ou de nouvelles lois de comportements.

Voici les principales caractéristiques de Fedoo :

- Code intégralement développé en python afin de permettre une utilisation simple et avoir recours à la richesse des librairies python tierces.
- Performances relativement bonnes grâce à l'utilisation massive des fonctions numpy vectorisées permettant d'éviter les boucles python lors de l'assemblage des matrices globales. L'utilisation solveurs optimisés comme *pardiso* (<https://www.pardiso-project.org>) permet d'assurer la performance et robustesse du solveur.
- Accès aux lois de comportement Simcoon proposant un grand nombre de comportements non linéaires et permettant de travailler en grandes transformations (formatulation lagrangien total ou actualisé).
- Solveurs non linéaire statique et dynamique implicite (un solveur dynamique explicite dont l'utilisation est limitée à des modèles simples est également proposé) avec possibilité d'inclure des contacts (node 2 surface ou self contact).
- Automatisation de l'application de conditions aux limites périodiques (y compris dans le cas de maillages non périodiques) et du calcul des matrices tangentes du milieu homogène équivalent (pour utilisation dans le cadre de la méthode FE^2 par exemple.)
- Accès ouvert à l'ensemble des variables, des matrices assemblées, des lois de comportement, des éléments utilisées, Permettant l'utilisation dans un cadre pédagogique et facilitant le développement de fonctions utilisateurs pour des applications recherches ou ingénieries.
- Une bibliothèque d'éléments variées : poutre, plaque stratifiées, zones cohésives, ...
- Fonctions de visualisation des résultats directement intégrée avec la librairie Pyvista [5] basée sur vtk [6].

4.2 Exemple simple

Un exemple de script pédagogique simple est donné ci-dessous. Il s'agit du problème classique de la plaque trouée. Un quart de la plaque est représenté et des conditions de symétrie sont appliquées.

```
import fedoo as fd
fd.ModelingSpace("2Dstress") #2D problem with plane stress assumption
```

```

# Generate a simple structured mesh "Domain" (plate with a hole).
mesh = fd.mesh.hole_plate_mesh(nr=11, nt=11, length=100, height=100, radius=20,
    elm_type = 'quad4', sym=False, name = "Domain")

# Define an elastic isotropic material with E = 2e5MPa et nu = 0.3 (steel)
fd.constitutivelaw.ElasticIsotrop(2e5, 0.3, name = 'ElasticLaw')

# Create the weak formulation of the mechanical equilibrium equation
fd.weakform.StressEquilibrium("ElasticLaw", name = "WeakForm")

# Create a global assembly
fd.Assembly.create("WeakForm", "Domain", name="Assembly", MeshChange = True)

# Define a new static problem
pb = fd.problem.Linear("Assembly")

# Definition of the set of nodes for boundary conditions
left = mesh.find_nodes('X', mesh.bounding_box.xmin)
right = mesh.find_nodes('X', mesh.bounding_box.xmax)

# Boundary conditions
# displacement on left (ux=uy=0)
pb.bc.add('Dirichlet', left, 'Disp', 0)

# displacement on right (uy=0 ; ux=1mm)
pb.bc.add('Dirichlet', right, 'DispY', 0)
pb.bc.add('Dirichlet', right, 'DispX', 1)

# Solve problem
pb.solve()

#extract the results from the Assembly object
results = pb.get_results("Assembly", ["Stress", "Disp", "Strain"])

# Plot the von-mises stress (averaged at nodes) on a deformed mesh
# with a scale factor = 10
results.plot("Stress", component="vm", data_type = "Node", scale=10)

```

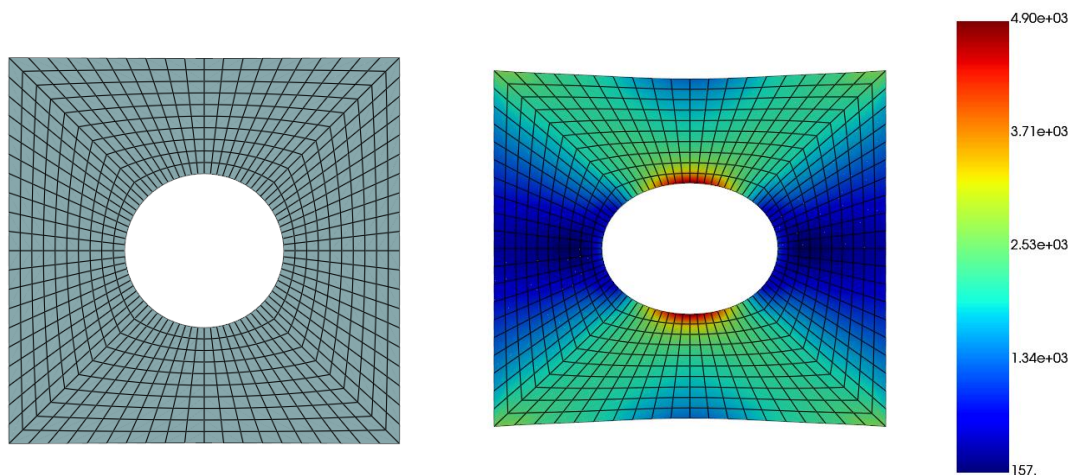


FIGURE 3 – Maillage initiale et contraintes de Von-Mises sur la plaque trouée

5 Utilisation combinée des outils 3MAH

Nous proposons ci-dessous une étude de cas plus aboutie qui consiste à déterminer le comportement homogène équivalent d'une cellule élémentaire d'un matériau architecturé en acier. Cet exemple per-

met de mettre en évidence l'interopérabilité des outils 3MAH. Un seul script peut permettre de générer une batterie de cellules élémentaires complexes à la volée avec le maillage correspondant (microgen), affecter un comportement aux matériaux constitutifs (simcoon), appliquer des conditions aux limites périodiques (simcoon/fedoo) et déterminer des matrices tangentes homogénéisées suivant différents trajets de chargement (fedoo). L'utilisation de la bibliothèque python "multiprocessing" permet en outre de lancer des calculs sur plusieurs processeur en parallèle. Avec la place de plus en plus importante prise par les méthodes d'apprentissage (IA, machine learning), il devient important de pouvoir disposer d'un code souple permettant de générer des bases de données de grande taille en une seule opération.

Dans le script ci-dessous, la loi de comportement utilisée est la loi de "EPICP" de simcoon qui une une loi élasto-plastique isotrope avec une fonction d'écrouissage sous forme de loi puissance. Pour des raisons de lisibilité, le script ci-dessous ne contient pas la génération du maillage sous microgen qui est traitée séparément. Les non-linéarités géométriques sont désactivées. Il est possible de les activer (nlgeom = True), mais dans ce cas les conditions aux limites doivent être cohérentes (par exemple en imposant le gradient du déplacement). Ce n'est pas proposé ici pour limiter la taille du script et de faciliter sa compréhension.

```
import fedoo as fd

# Define the Modeling Space – Here 3D problem
fd.ModelingSpace("3D")

# Import the mesh generated with Microgen
mesh = fd.mesh.import_file('MeshPeriodic.msh')['tet4']

# Get the bounding box (corners coordinates and center)
bounds = mesh.bounding_box
volume = bounds.volume
crd_center = bounds.center
# Nearest node to the center of the bounding box for boundary conditions
center = mesh.nearest_node(crd_center)

# Material definition and simcoon elasto-plastic constitutive law
Re = 300
k = 1000
m = 0.25
alpha = 1e-5
props = [1e5, 0.3, alpha, Re, k, m]
material = fd.constitutivelaw.Simcoon("EPICP", props, name='ConstitutiveLaw')

# Create the weak formulation of the mechanical equilibrium equation
wf = fd.weakform.StressEquilibrium("ConstitutiveLaw", name = "WeakForm", nlgeom=False)

# Assembly
assemb = fd.Assembly.create("WeakForm", mesh, 'tet4', name="Assembly")

# Type of problem
pb = fd.problem.NonLinear("Assembly")

# Set the desired outputs at each time step
res = pb.add_output('results', 'Assembly', ['Disp', 'Stress', 'Strain', 'Statev'])

# Boundary conditions for the linearized strain tensor
E = [0, 0, 0, 0.1, 0, 0] # [EXX, EYY, EZZ, 2EXY, 2EXZ, 2EYZ]

bc_periodic = fd.constraint.PeriodicBC(periodicity_type="small_strain", dim=3)
pb.bc.add(bc_periodic)

# Fixed point on the center to avoid rigid body motion
pb.bc.add('Dirichlet', center, 'Disp', 0)

# Enforced mean strain
pb.bc.add('Dirichlet', "MeanStrain", E) # EpsXX, EpsYY, EpsZZ
```

```

# ----- Non linear solver -----
pb.nlsolve(dt=0.2, tmax=1, update_dt=False, tol_nr=0.01)

# ----- Post-Treatment -----
res.plot() #plot the raw mesh
res.plot('Stress', "vm") #plot the von-mises stress
res.plot('Statev', 1, 'Node') #plot the cumulated strain, averaged at nodes

# get the homogenized tangent matrix
HomogenizedTangentMatrix = fd.homogen.get_tangent_stiffness(pb)

```

La figure ci-dessous présente les résultats de l'opération de génération CAO et de maillage, ainsi que des résultats de la simulation par éléments finis. Nous pouvons ainsi obtenir une chaîne de simulation automatique de la génération d'une microstructure paramétrique à la simulation suivant différents trajets de chargement permettant de construire une base de données pouvant servir à des modèles d'apprentissage.

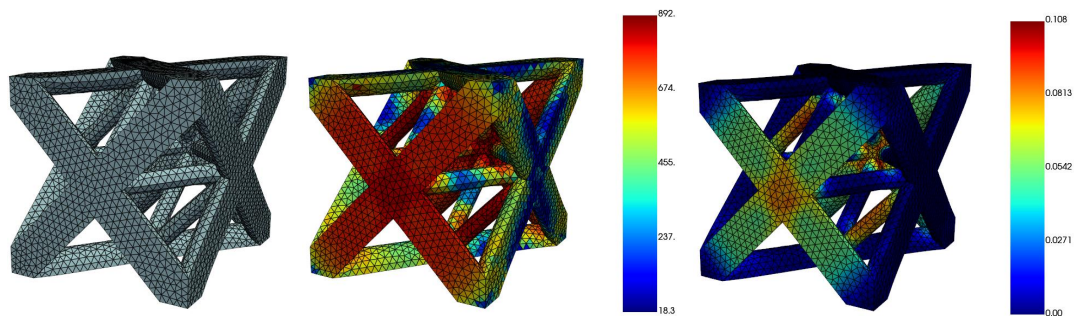


FIGURE 4 – Octet-truss avec conditions aux limites périodiques : maillage généré par microgen (gauche), contraintes de von-mises (centre), plasticité cumulée (droite)

Références

- [1] Lemaitre, J., & Chaboche, J. L. (2002). *Mechanics of solid materials*. Cambridge, UK : Cambridge University Press.
- [2] Meraghni, F., Chemisky, Y., Piotrowski, B., Echchorfi, R., Bourgeois, N., & Patoor, E. (2014). Parameter identification of a thermodynamic model for superelastic shape memory alloys using analytical calculation of the sensitivity matrix. *European Journal of Mechanics, A/Solids*, 45, 226–237.
- [3] Mora, V. *Etude de l'intégration temporelle du tenseur taux de déformation. Application à la modélisation de l'élastoplasticité en grandes transformations*, Thèse de l'Université de Bretagne Sud, 2004.
- [4] Wenzel Jakob and Jason Rhinelander and Dean Moldovan, (2017), pybind11 – Seamless operability between C++11 and Python, <https://github.com/pybind/pybind11>
- [5] Sullivan et al., (2019). PyVista : 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software*, 4(37), 1450, <https://doi.org/10.21105/joss.01450>
- [6] Schroeder, Will; Martin, Ken; Lorensen, Bill (2006), *The Visualization Toolkit* (4th ed.), Kitware, ISBN 978-1-930934-19-1