CSMA 2022
15ème Colloque National en Calcul des Structures
16-20 Mai 2022, Presqu'île de Giens (Var)

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

E. Prulière[1], Y. Chemisky[2]

[1] *I2M, Arts et Métiers, etienne.pruliere@ensam.eu*
[2] *I2M, Université de Bordeaux, yves.chemisky@u-bordeaux.fr*

Résumé —
Nous proposons un ensemble de librairies open-source permettant d'analyser le comportement de matériaux hétérogènes, en intégrant trois principaux outils de simulation complémentaires : une librairie de génération de microstructures virtuelles (microgen), une librairie de simulation par éléments finis (fedoo) et une librairie 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 maitrise 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 inovantes.

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 fonctionalités.

L'objectif de 3MAH est de proposer :

— un ensemble de librairies 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 perfomances/souplesse d'utilisation/facilité de développement.

Cet objectif ambitieux est réalisé à l'aide d'une API (Application Programming Interface, ou librairie) 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 (librairie simcoon) (ii) un solveur éléments finis pour des simulations sur structure et sur VER de matériaux non linéaires hétérogènes (librairie fedoo) (iii) Un ensemble de scripts permettant de générer et mailler de manièrte automatisée des microstrcutures numériques (librairie microgen).

Les outils présentés doivent enfin être au plus proche des outils de simulation industriels dans la philosophie de simulation. En ce sens, une compatibilité est prévue entre 3MAH et des solutions industrielles de simulation (Abaqus, CodeAster) et de visualisation (ParaView)

2 Simcoon

La librarie 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 apparaitre 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 Boost.Python [1] 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 = sim.L_iso(E,nu,'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 = 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 [2]. Cette loi intègre un écrouissage isotrope non-linéaire et deux écrouissages cinématiques

non-linéaires. L'évolution de la fonction d'écrouissage isotrope s'exprime :

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

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

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

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

$$|\sigma - \mathbf{X}| - H - \sigma^Y = 0\,, \tag{3}$$

où σ^Y est a limite d'élasticité initiale du matériau. Les paramètres à identifier dans cet exemple sont respectivement σ^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 implementé dans simcoon est hybride, avec une partie heuristique (algorithme génétique) et déterministe (à gradient de type Levenberg-Marquardt [3]) . 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
```

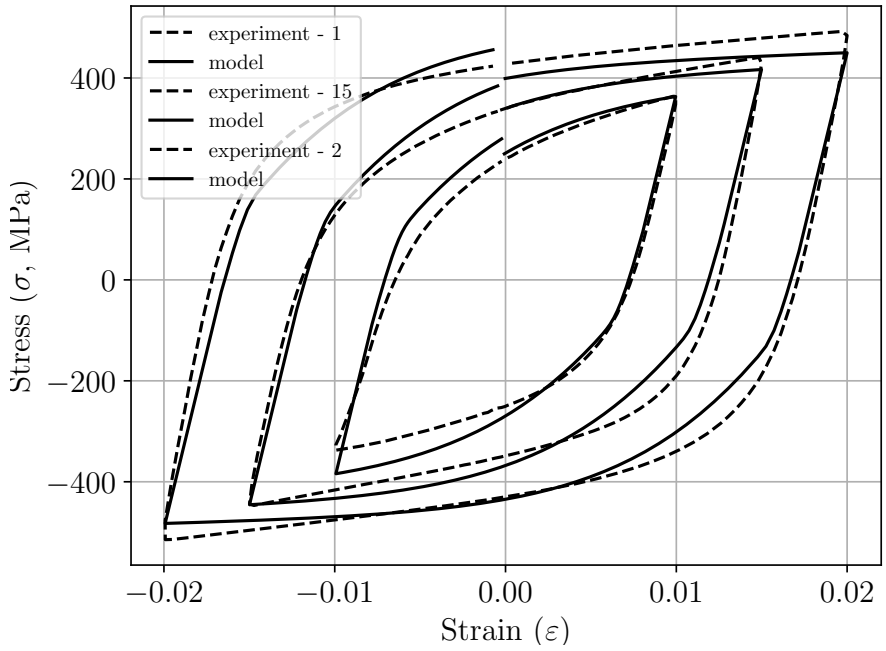

FIGURE 1 – Comparaison essais - modèle identifié.

```
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'
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 literature sont retrouvés ici (par exemple dans [4]), 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 deux libraries : CADQuery (https ://-cadquery.readthedocs.io/en/latest/#) pour la gestion des outils CAO, et 'gmsh' (https ://gmsh.info) pour le maillage. La libraries 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 permettant de résoudre des problèmes de mécanique avec la méthode des éléments finis. Initialement Fedoo était une librairie ayant pour objectif de résoudre des problèmes en utilisant la méthode PGD (Proper Generalized Decomposition). Depuis quelques années, le développement de Fedoo s'oriente principalement sur les éléments finis classiques, bien qu'un solver PGD semi-automatique soit toujours proposé. Voici les principales carractéristiques de Fedoo :

- Code intégralement développé en python afin de permettre une utilisation simple et avoir recours à la richesse des librairies python tierces.
- Les performances sont assurées grâce à l'utilisation massive des fonctions numpy vectoriées permettant d'éviter les boucles python lors de l'assemblage des matrices globales. L'utilisation de *pardiso* (https ://www.pardiso-project.org) permet d'assurer la performance et riobustesse du solveur.
- Accès aux lois de comportement Simcoon proposant un grand nombre de comportements non linéaires (non linéarités géométriques et/ou matériels).
- Solveurs non linéaire statique et dynamique implicite (un solveur dynamique explicite dont l'utilisation est limitée à des modèles simples est également proposé).
- 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 ingénieures.
- Une bibliothèque d'éléments variées : poutre, plaque stratifiées, zones cohésives, ...

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. Un script similaire a été proposé l'année dernière à l'école des Arts et Métiers pour permettre un TP éléments finis à distance pendant le confinement. Le TP est réalisable en ligne grâce à l'utilisation d'un jupyter notebook hébergé sur un serveur ("plate_with_hole.ipynb" disponible ici : https ://mybinder.org/v2/gh/pruliere/simple_FE/HEAD)

```
from fedoo import *
import numpy as np

#Define the Modeling Space - Here 2D problem with plane stress assumption.
Util.ProblemDimension("2Dstress")
```

```
#Generate a simple structured mesh "Domain" (plate with a hole).
meshObject = Mesh.HolePlateMesh(Nx=11, Ny=11, Lx=100, Ly=100, R=20, \
        ElementShape = 'quad4', ID ="Domain")
Util.meshPlot2d("Domain") #plot the mesh "Domain"

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

#Create the weak formulation of the mechanical equilibrium equation
WeakForm.InternalForce("ElasticLaw", ID = "WeakForm")

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

#Define a new static problem
Problem.Static("Assembly")

#Definition of the set of nodes for boundary conditions
crd = meshObject.GetNodeCoordinates()
left   = np.where(crd[:,0] == np.min(crd[:,0]))[0]
right = np.where(crd[:,0] == np.max(crd[:,0]))[0]
bottom = np.where(crd[:,1] == np.min(crd[:,1]))[0]

#Boundary conditions
#symetry condition on left (ux = 0)
Problem.BoundaryCondition('Dirichlet','DispX',     0    , left)
#symetry condition on bottom edge (ux = 0)
Problem.BoundaryCondition('Dirichlet','DispY',     0    , bottom)
#displacement on right (ux=0.1mm)
Problem.BoundaryCondition('Dirichlet','DispX', 0.1, right)

Problem.ApplyBoundaryCondition()

#Solve problem
Problem.Solve()

#---------- Post-Treatment ----------
#Get the stress tensor, strain tensor, and displacement (nodal values)
res_nd = Problem.GetResults("Assembly", ['disp', 'Stress','Strain'], 'Node')

#plot the von mises sress
Util.fieldPlot2d("Domain", "ElasticLaw", disp = Problem.GetDisp(),  \
        dataID = 'stress', component='vm', data_min=None, data_max = None, \
        scale_factor = 1, plot_edge = True, nb_level = 10, type_plot = "real")
```

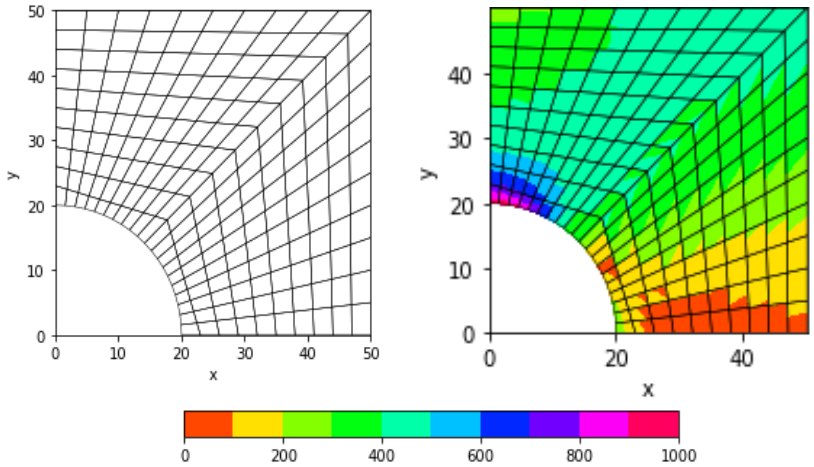

FIGURE 3 – Maillage initiale et contraintes de Von-Mises sur la plaque trouée

5 Utilisation combinée des outils 3MAH

Non 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 permet 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 homogénéisés 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 simcooon 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.

```
from fedoo import *
import numpy as np

#Define the Modeling Space - Here 3D problem
Util.ProblemDimension("3D")

#Import the mesh generated with Microgen
Mesh.ImportFromFile('MeshPeriodic.msh', meshID = "Domain")

#Get the imported mesh
mesh = Mesh.GetAll()["Domain2"]

#Get the bounding box (corners coordinates and center)
Xmin, Xmax, crd_center = mesh.GetBoundingBox(return_center = True)
# total volume of the bounding box
Volume = (Xmax-Xmin).prod()

#Nearest node to the center of the bounding box for boundary conditions
center = mesh.GetNearestNode(crd_center)

# Add 2 virtual nodes for macro strain
StrainNodes = Mesh.GetAll()["Domain2"].AddNodes(crd_center, 2)

# Material definition and simcoon elasto-plastic constitutive law
Re = 300
k = 1000
m = 0.25
alpha = 1e-5
props = np.array([[1e5, 0.3, alpha, Re, k, m]])
Material = ConstitutiveLaw.Simcoon("EPICP", props, 8, ID='ConstitutiveLaw')

#Create the weak formulation of the mechanical equilibrium equation
wf = WeakForm.InternalForce("ConstitutiveLaw", ID = "WeakForm", nlgeom=False)

# Assembly
assemb = Assembly.Create("WeakForm", "Domain2", 'tet4', ID="Assembly")

# Type of problem
Problem.NonLinearStatic("Assembly")

# Set the desired ouputs at each time step
Problem.AddOutput('results', 'Assembly', ['disp', 'cauchy', 'PKII', 'strain', '
    cauchy_vm', 'statev'], output_type='Node', file_format ='vtk')
```

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

Util.DefinePeriodicBoundaryCondition('Domain2',
        [StrainNodes[0], StrainNodes[0], StrainNodes[0],
         StrainNodes[1], StrainNodes[1], StrainNodes[1]],
          ['DispX', 'DispY', 'DispZ', 'DispX', 'DispY', 'DispZ'], dim='3D')

#fixed point on the center to avoid rigid body motion
Problem.BoundaryCondition('Dirichlet', 'Disp', 0, center)

#Enforced mean strain
Problem.BoundaryCondition('Dirichlet', 'Disp', [E[0], E[1], E[2]], [
                          StrainNodes[0]])  # EpsXX, EpsYY, EpsZZ
Problem.BoundaryCondition('Dirichlet', 'Disp', [E[3], E[4], E[5]], [
                          StrainNodes[1]])  # EpsXY, EpsXZ, EpsYZ

Problem.ApplyBoundaryCondition()

# ------------------ Non linear solver ----------------------------------------
Problem.SetSolver('CG') #conjugate gradient solver
Problem.NLSolve(dt=0.2, tmax=1, update_dt=False, ToleranceNR=0.1)

# ------------------ Post-Treatment -------------------------------------------
# Get the stress and strain tensor (PG values)
res = Problem.GetResults('Assembling', ['Strain','Stress'], 'GaussPoint')
TensorStrain = res['Strain']
TensorStress = res['Stress']
```

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 (obtenus avec paraview). Notons que le post-traitement peut également être entièrement automatisé en utilisant un script python, ce qui permet d'obtenir une chaine de simulation automatique de la génération du modèle à la génération d'une base de données de résultats post-traités.

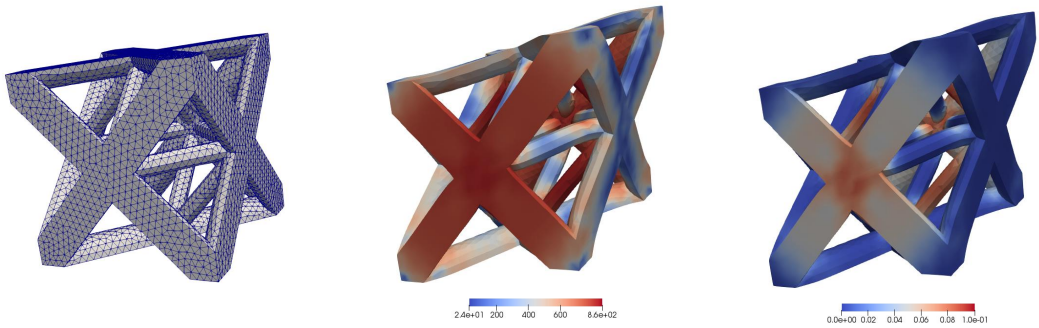

FIGURE 4 – Octet-truss avec conditions aux limites périodiques : maillage généré par microgen (gauche), contraintes de von-mises (centre), plasticité cumulée (droite)

Références

[1] https ://www.boost.org/doc/libs/1_78_0/libs/python/doc/html/index.html

[2] , J., & Chaboche, J. L. (2002). Mechanics of solid materials. Cambridge, UK : Cambridge University Press.

[3] 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.

[4] V. Mora *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.