Institut Supérieur des Matériaux
et Mécaniques Avancés
44 avenue F.A. Bartholdi,
72000, LE MANS
Téléphone : 33 (0) 243 21 40 00
E-mail : ismans@ismans.fr

Université du Luxembourg
Campus Kirchberg
6 rue Richard Coudenhove-Kalergi
L1359, Luxembourg
LUXEMBOURG
Tel : (+ 352) 46 66 44 5000

Extension of 2D FEniCS implementation of Cosserat non-local elasticity to the 3D case

2nd year internship

Trainee:
Camille SAUTOT

Academic year 2013/2014
Beginning: July 1st, 2014
End: August 31st, 2014

Supervisor:
Stéphane Bordas
Professor

INTERNSHIP WITHOUT CONFIDENTIALITY CLAUSE

Thank you to Stéphane Bordas who supervised the internship. Thank you also to Jack Hale who mentored my work, for his attention, and for the time awarded to me.

Contents

Figures

Tables

1. Introduction

The classical elasticity theory is inadequate in the modelling of granular, fibrous or lattice materials. That's why Cosserat brothers [5] developed a theory of elasticity including a local rotation of points as well as the translation, and a couple stress as well as the force stress. Eringen [7] incorporated a micro-inertia for the study of dynamic effects and renamed the Cosserat elasticity micro-polar elasticity. Python language can be used in the implementation of the finite elements method and in the computation of solutions for Cosserat solids. The studies resulting from the existing analytical solutions for 2D problems [7] generate an extension of the models to the 3D case. However in the absence of analytical solution for some 3D problems in Cosserat elasticity, the comparison is made with other theories which are limiting cases of Cosserat theory.

The research of the Faculty of Science, Technology and Communication (FSTC) focuses on informatics, engineering, mathematics, life sciences, physics and material science. This internship was completed in the Research Unit in Engineering Science (RUES). This research unit is organized in four main clusters : construction and design, energy and environment, automation and mechatronics, and geophysics. One of the main objectives is the improvement of the numerical simulation to reduce the required experiment effort. The implementation of the Cosserat theory is part of this project. In order to realise this, the DOLFIN library of FEniCS Project is used. This package, implemented in Python code, permits to mesh a domain and calculate it using the finite elements method. Mechanical problems can be solved by this way. The FEniCS Project is used in various domains with complex geometries difficult to mesh in 3D (for example modelling of hemodynamics or cerebrospinal fluid flow). That's why the DOLFIN mesher only realises meshes with triangle and tetrahedron elements. The special package *gmsh* permits to add more options for mesh complex domains. Progressive and refined meshes can be easily realised. But the DOLFIN solver cannot currently construct hexahedral and quadrilateral finite elements but this functionality is under development.

The objective of the internship is the extension of the existing 2D FEniCS implementation of Cosserat elasticity [9] to the 3D case. The first step is the implementation of a patch-test for a simple problem in classical elasticity as a Timoshenko's beam [1] - this study will show that DOLFIN could offer approximated solutions converging to the analytical solution. The second step is the computation of the stress in a plate with a circular hole. The stress concentration factors around the hole in classical and Cosserat elasticities will be compared, and a convergence study for the Cosserat case will be realised. The third step is the extension to the 3D case with the computation of the stress concentration factor around a spherical cavity in an infinite elastic medium. This computed value will be compare to the analytical solution described by couple-stress theory [10].

2. Timoshenko's beam in classical elasticity

2.1. Problem definition

The first step of the work is the computation of a Timoshenko's beam in classical elasticity. The cantilever beam, shown in Figure 1, with a parabolic end load is represented by a plate of depth D, length L and unit thickness.

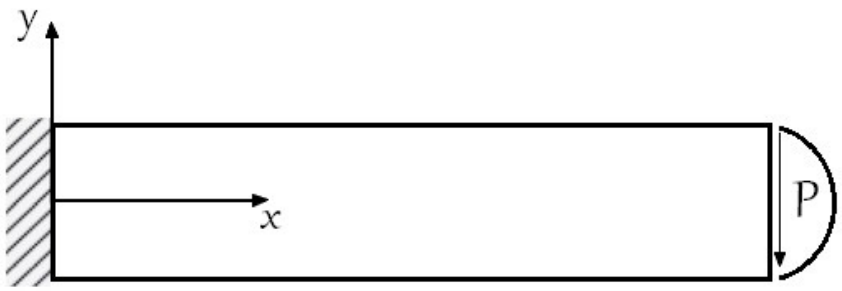

Figure 1: Sketch of the 2D cantilever

Boundary conditions at x = 0 match with the expression of displacements analytical solution [1] :

$$u_x = -\frac{Py}{6EI}\left[(6L-3x)x + (2+\upsilon)y^2 - \frac{3D^2}{2}(1+\upsilon) \right] \quad (1)$$

$$u_y = -\frac{P}{6EI}\left[3\upsilon y^2(L-x) + (3L-x)x^2 \right] \quad (2)$$

where E is Young's modulus, υ is Poisson's ratio and I is the second moment of area of the cross-section. According to [2], the applied end load must match with the shearing stress to have (1) and (2) representing the exact solution. Thus the end load F is equal to the shearing stress and is given by :

$$F = -\frac{P}{2I}\left[\frac{D^2}{4} - y^2\right] ,$$

where P is the load intensity.

The objective of this study is the evaluation of the error between analytical solution and calculated solution. The redaction of a Python code for this problem includes 4 steps : mesh, computation of the variational problem, solving and evaluation of the error. We can easily mesh the beam with regular triangles elements, like in Figure 2. The domain is meshed with equal rectangles which are divided in two triangles. Sub-domains are extracted to apply the load and boundary conditions.

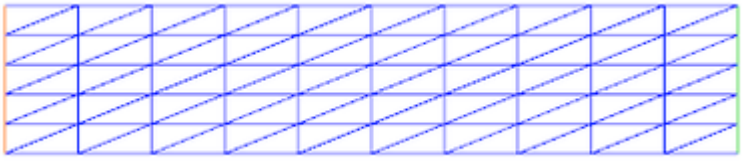

Figure 2: Mesh of the beam

We have to define a test function u and a trial function v, which are functions of the mesh space.

The weak form is used to compute the variational problem :

$$\int_{\Omega}\left(\frac{\mu}{2}\ (\nabla \boldsymbol{u}+\nabla \boldsymbol{u}^{T}).(\nabla \boldsymbol{v}+\nabla \boldsymbol{v}^{T})\right)d\Omega\ +\ \int_{\Omega}(\lambda(\nabla .\boldsymbol{u})(\nabla .\boldsymbol{v}))d\Omega\ =\ \int_{\Gamma}(\boldsymbol{v}.\boldsymbol{F})\,d\Gamma \quad (3)$$

where μ and λ are Lamé constants. The implementation of the variational problem in the Python code is (complete code in Appendix 1) :

```
# Variational problem
u = TrialFunction(V)
v = TestFunction(V)

sigma = lambda v: 2.0*mu*sym(grad(v)) + lamb*tr(sym(grad(v))) * Identity(2)

a = inner(sigma(u), grad(v))*dx
L = inner(F,v)*ds(1)

# Solve
u_h = Function(V)

problem = LinearVariationalProblem(a, L, u_h, bc)
solver  = LinearVariationalSolver(problem)
solver.solve()
```

To evaluate the error we need to compare analytical solution, given by (1) and (2), and computed solution. In DOLFIN, the error can be calculate by a predefined function named *errornorm*. The used expression of the error norm is given by :

$$error\ norm\ =\ \int (u_{exact}-\ u_{computed})^{2}\,dx\quad .$$

For each mesh, the program evaluates the error for first, second and third order space functions, in L^2 norm. To have a normalized value of the error, we have to divide the error norm by the norm of $u_{computed}$. With a graph of the evolution of the error depending on the elements size, we can know if the computed solution converges to the analytical solution given by (1) and (2). We use a linear solver, therefore we have to verify that the results are in the linear field.

2.2. Numerical results

For the numerical application the beam is of dimensions D = 2 m, L = 10 m and the end load is P = 1000 N. The material properties used are E = 210 GPa and ν = 0.3.

2.2.i) Displacements

Stresses and displacements are post-processed with ParaView. Figures 3 and 4 show the computed displacements in x and y directions for one of the mesh densities.

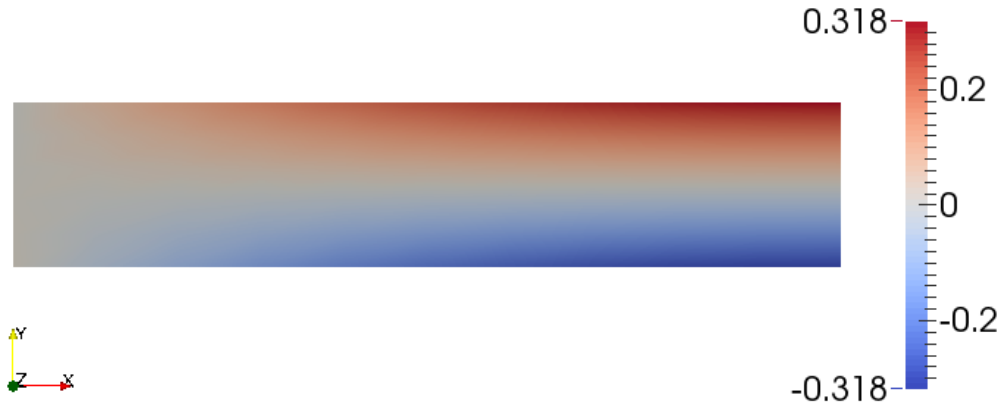

Figure 3: Computed displacement on x direction.

Figure 4: Computed displacement on y direction.

The symmetry of x-displacement and the deformed shape viewed for y-displacements match with the results for a Timoshenko's beam. Moreover values of computed displacements are very close to the analytical solution. But the study of the error is necessary to confirm the convergence. The deformed shape of the beam is shown in Figure 5.

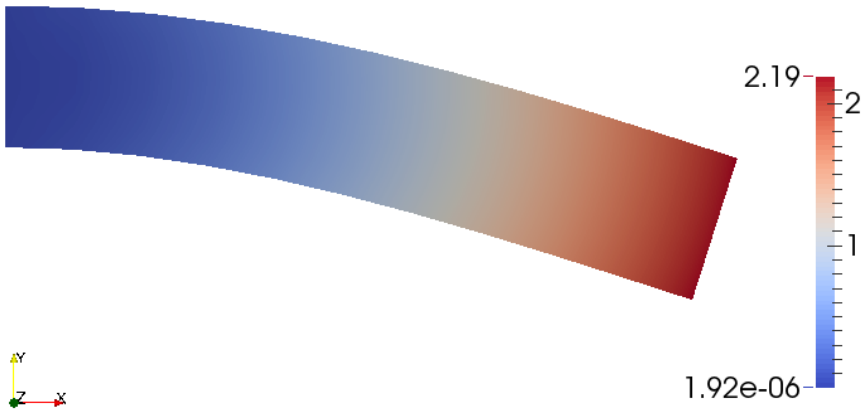

Figure 5: Deformed shape

The value of the maximum displacement is 2.19 mm. This result observes the geometric linearity

hypothesis.

2.2.ii) Stresses

Define the stress is necessary for the computation of the solution, because the variational weak form (3) results from this definition. The stress is given by :

$$\sigma = \mu(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) + \lambda(\nabla . u) I \quad ,$$

where μ and λ are Lamé constants. The expression of the parabolic end load applied to the beam must be the same as the shearing stress :

$$\tau_{xy} = -\frac{P}{2I}[\frac{D^2}{4} - y^2] \quad .$$

So in post-processing we have to verify that the shear stress corresponds to the exact solution. The stress calculated from the computed solution is represented in Figure 6.

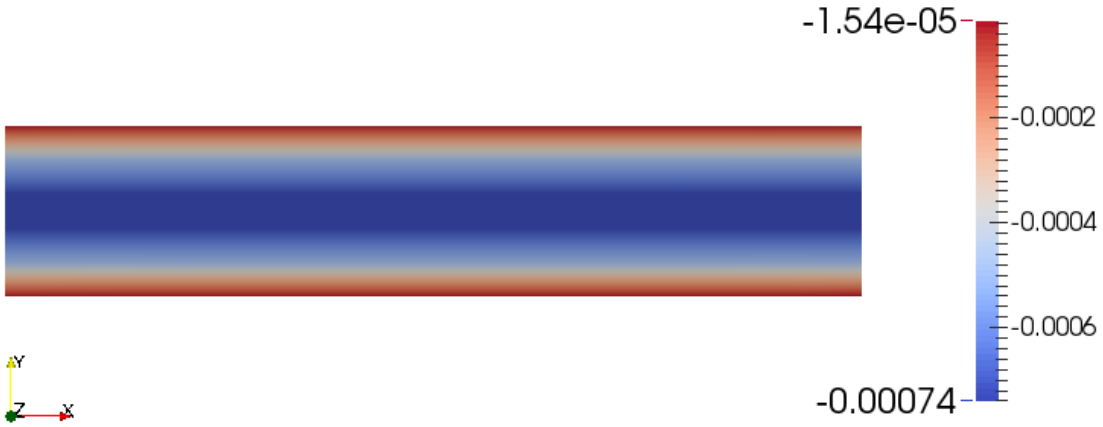

Figure 6: Shear stress

The form of the computed shear stress in the beam matches with the analytical solution. The results is stress and displacements seem to match with analytical expressions, but we have to check the convergence.

2.2.iii) Convergence

The solution is computed for 6 different densities of mesh : 10x5, 20x10, 30x15, 40x20, 50x25 and 60x30. For the convergence study of the computed solution, the graph of the error depending on the elements size is necessary. Plotting of the graphs are visible in Figures 7.

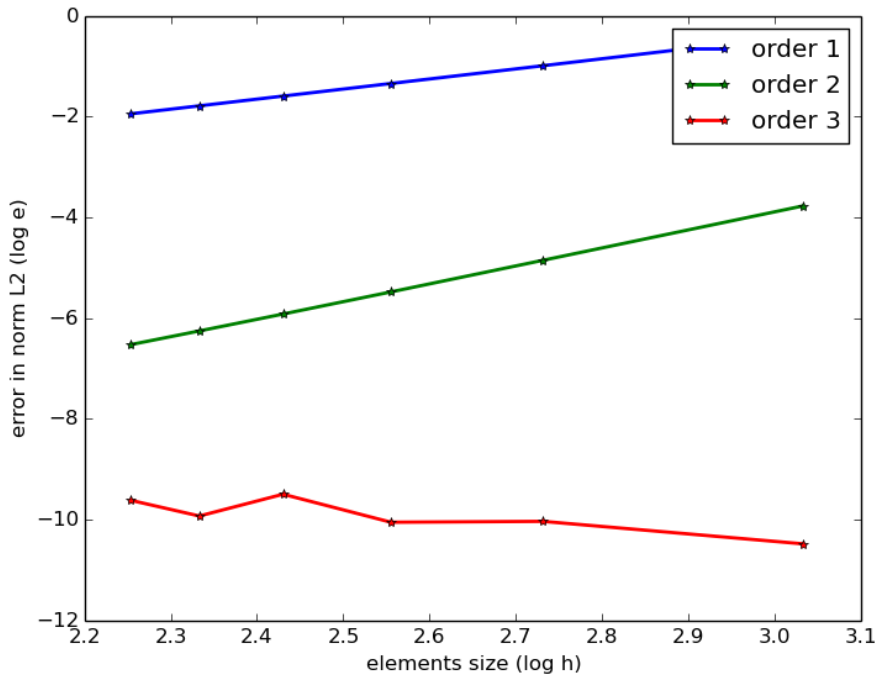

Figure 7: Graph of the error in norm L^2 depending on the elements size.

A space function of first or second order ensure the convergence for the solution : the error decreases with the elements size. We can see that the convergence with an order 2 is quicker than with the order 1. The third order ensures instant convergence because the polynomial order of the exact solution is 3. Thus the computed solution matches with the analytical solution with an error of 10^{-10}.

3. Plate with hole in Cosserat elasticity

3.1. Literature review

Classical elasticity theory is inadequate in the modelling of some newer materials with a granular, fibrous or lattice structure. Cosserat elasticity theory, developed by Cosserat brothers [5], offers better results in numerical computations introducing a local rotation of points independent of translation. Therefore the material can transmit a couple stress as well as the usual force. The field equations for this theory are given by Mindlin [6], and Eringen [7] incorporates a micro-inertia and renames Cosserat elasticity micro-polar elasticity. An isotropic Cosserat solid has six elastic constants [8] : λ, μ, α, β, γ and κ, in contrast to the two elastic constants of a classically elastic solid. These constants permit to describe :

Young's modulus $E = (2\mu+\kappa)\frac{(3\lambda+2\mu+\kappa)}{(2\lambda+2\mu+\kappa)}$,

shear modulus $G = \frac{(2\mu+\kappa)}{2}$,

Poisson's ratio $\nu = \frac{\lambda}{(2\lambda+2\mu+\kappa)}$,

characteristic length for torsion $l_t = [\frac{(\beta+\gamma)}{(2\mu+\kappa)}]^{(1/2)}$,

characteristic length for bending $l_b = [\frac{\gamma}{2}(2\mu+\kappa)]^{(1/2)}$,

coupling number $N = [\frac{\kappa}{2}(\mu+\kappa)]^{(1/2)}$, which determines the strength of coupling between the displacement and local rotation fields, and

polar ratio $\Psi = \frac{(\beta+\gamma)}{(\alpha+\beta+\gamma)}$.

If α, β, γ, κ are equal to 0, the solid becomes classically elastic.

In plane strain, stress and strain can be written $\sigma=\{\sigma_{xx}\ \sigma_{yy}\ \sigma_{xy}\ \sigma_{yx}\ m_{xz}\ m_{yz}\}^T$, where σ is the stress and m is the couple stress, and $\epsilon=\{\epsilon_{xx}\ \epsilon_{yy}\ \epsilon_{xy}\ \epsilon_{yx}\ \chi_{xz}\ \chi_{yz}\}^T$, where ε is the strain and χ the torsion. The constitutive equations for isotropy [9] are : $\sigma = D.\epsilon$, where

$$D=G.\begin{bmatrix} \frac{2(1-\nu)}{1-2\nu} & \frac{2\nu}{1-2\nu} & 0 & 0 & 0 & 0 \\ \frac{2\nu}{1-2\nu} & \frac{2(1-\nu)}{1-2\nu} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{1-N^2} & \frac{1-2N^2}{1-N^2} & 0 & 0 \\ 0 & 0 & \frac{1-2N^2}{1-N^2} & \frac{1}{1-N^2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 4l^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4l^2 \end{bmatrix}$$

Displacements can be calculated by :

$$\epsilon = \begin{Bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & \frac{\partial}{\partial x} & -1 \\ \frac{\partial}{\partial y} & 0 & 1 \\ 0 & 0 & \frac{\partial}{\partial x} \\ 0 & 0 & \frac{\partial}{\partial y} \end{Bmatrix} . \begin{Bmatrix} u \\ v \\ \phi \end{Bmatrix} \qquad (4)$$

The Research Unit in Engineering Science of the University of Luxembourg is studying inclusion in elastic materials. Thus the first model of this work is a infinite plate with a circular hole.

3.2. Problem definition

The square plate with a circular hole is a section of an infinite elastic medium. A load is

applied in the plane of this section. Therefore we can use the plane strain hypothesis. First we calculate the stress concentration factor around the hole in classical elasticity ; we know that if we apply a traction force F on a side, the stress concentration factor is equal to 3F. Then we compare it with the stress concentration factor, defined by Eringen [7], in a convergence study. The stress concentration factor (SCF) is defined as :

$$SCF_{Cosserat} = \frac{3+F_1}{1+F_1} \text{ , with } F_1 = 8(1-\nu)\frac{b^2}{c^2}\left(4 + \frac{R^2}{c^2} + \frac{2R}{c} \times \frac{K_0(R/c)}{K_1(R/c)}\right) ,$$

where ν is Poisson's ratio, R is radius of the circle, b is intrinsic length scale corresponding to the average grain size of the material, c is equal to b/N, and K is a modified Bessel function of the second kind. Finally we compare stress concentration factor in classical elasticity and Cosserat elasticity.

The structure has two symmetries, thus we consider a quarter of the square plate, and apply appropriated boundary conditions, as shown in Figure 8.

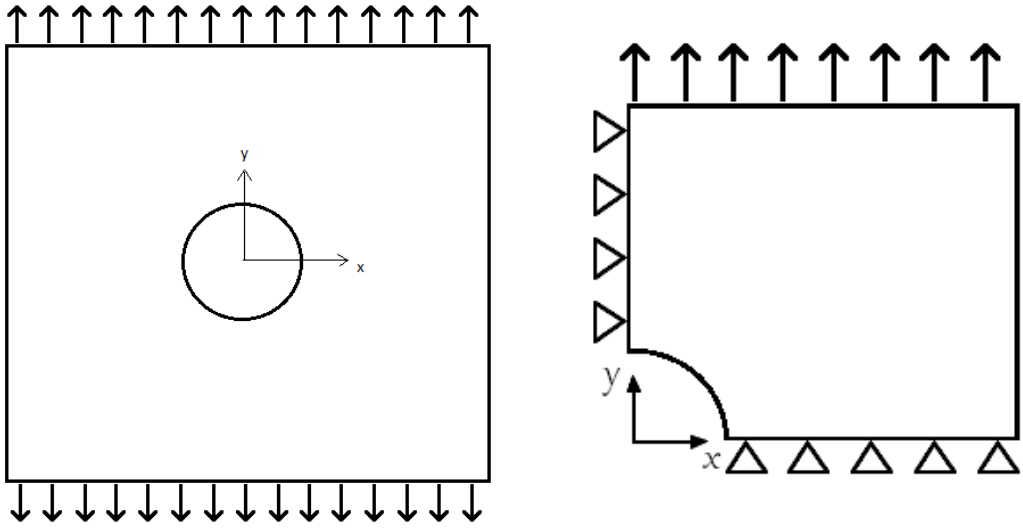

Figure 8: Plate with hole and boundary conditions

A triangle mesh is realised using the *msh* package (Figure 9).

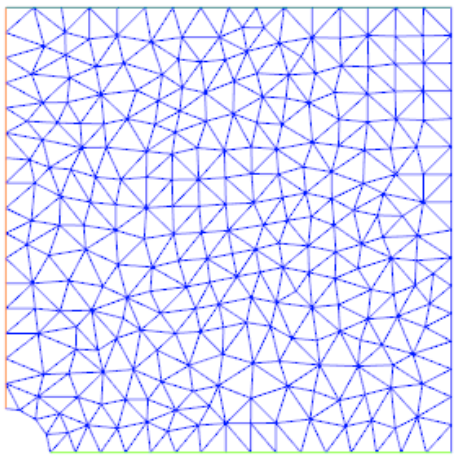

Figure 9: Mesh of the plate

Five meshes are used in the convergence study : the error between the analytical and the computed

SCF is evaluated for different mesh densities, and plotted depending on the elements size.

3.3. Numerical results

3.3.i) Classical elasticity

The objective of this step is to calculate the stress concentration factor with an unit load traction applied on the right side of the plate. The stress in the plate is calculated with computed displacements. First component of the stress is visualized with ParaView, shown in Figure 10.

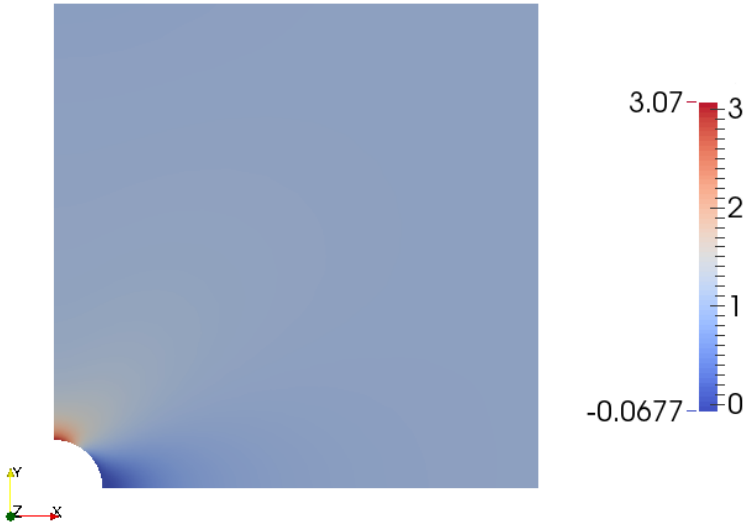

Figure 10: Stress in the plate in classical elasticity (σ_{xx})

We verify that the stress concentration factor is near to 3, because we apply a unit traction force. This result well approximate the analytical solution, that's why we can use this mesh and Python to study the Cosserat case.

3.3.ii) Cosserat elasticity

Stress in the plate is computed using the matrix D and equation (4). A Python code is written to calculate displacements, strain and stress (cf. Appendix 2). A unit traction force is applied on the top side of the plate. We compare the computed and the analytical stress concentration factors depending on the mesh density. Elements of degree 2 are used in the convergence study. The results presented in the Table 1 are for $\nu = 0.3$, $l = 0.2$ and $N = 0.8$.

Analytical SCF = 2.996	
Elements size	Error (%)
10.63	10.20
5.17	7.08
3.24	5.83
2.25	2.92
1.80	1.38

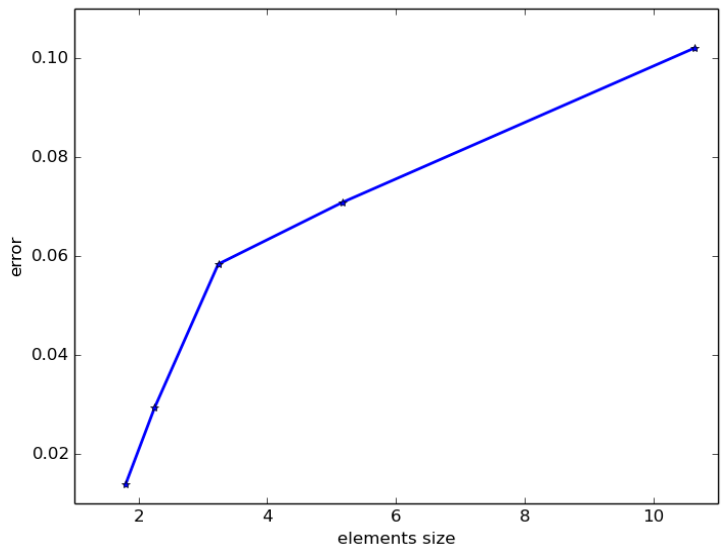

Table 1: Convergence of the computed SCF in Cosserat elasticity

The analytical SCF is equal to 2.996. Comparing the computed SCF with this value, we see that the error decreases with the elements size. Thus the computed solution convergences to the analytical solution defined by Eringen [7].

4. Extension to the 3D case : spherical inclusion

The next step of the study is the extension of the 2D implementation to the 3D case. We consider a spherical cavity in an infinite medium subject to uniform far-field tension. We first compute the solution in classical elasticity to evaluate the stress concentration factor. Then we implement the problem in Cosserat elasticity and evaluate the effect of one varying parameter on the stress concentration factor. In Cosserat elasticity, the implementation of asymmetric stress, couple stress, strain and curvature tensors is needed. These tensors are defined for a linear isotropic continuum by constitutive equations [7] :

strain tensor : $\epsilon_{kl} = u_{l,k} - e_{klm}\phi_m$,

curvature (torsion) tensor : $\chi_{kl} = \phi_{l,k}$,

stress tensor : $\sigma_{kl} = \lambda\epsilon_{rr}\delta_{kl} + (\mu+\kappa)\epsilon_{kl} + \mu\epsilon_{lk}$, and

couple stress tensor : $m_{kl} = \alpha\chi_{rr}\delta_{kl} + \beta\chi_{lk} + \gamma\chi_{kl}$,

where u is the displacement and φ the rotation, and δ_{kl} is Kronecker symbol and e_{klm} is permutation symbol or Levi-Civita symbol. The complete weak form for a 3D problem in Cosserat elasticity is :

$$\iiint_V (\sigma_{ij}.\epsilon_{ij} + m_{ij}.\chi_{ij})\,dV = \iint_S (\boldsymbol{t}_i.\boldsymbol{u}_i + \boldsymbol{Q}_i.\boldsymbol{\phi}_i)\,dS + \iiint_V (\boldsymbol{p}_i.\boldsymbol{u}_i + \boldsymbol{q}_i.\boldsymbol{\phi}_i)\,dV \quad , (5)$$

where t is the traction vector, Q is the couple vector, p is the body force vector and q is the body moment vector.

4.1. Problem definition

We model the complete structure by a cube with a free-stress spherical cavity of radius R. A

unit traction force is apply on both top and bottom faces. The two symmetry planes are leveraged in the study and the problem is reduced to a cube with a quarter sphere in a corner. Therefore boundary conditions are applied on three faces and the traction load is applied on the top face as shown in Figure 11.

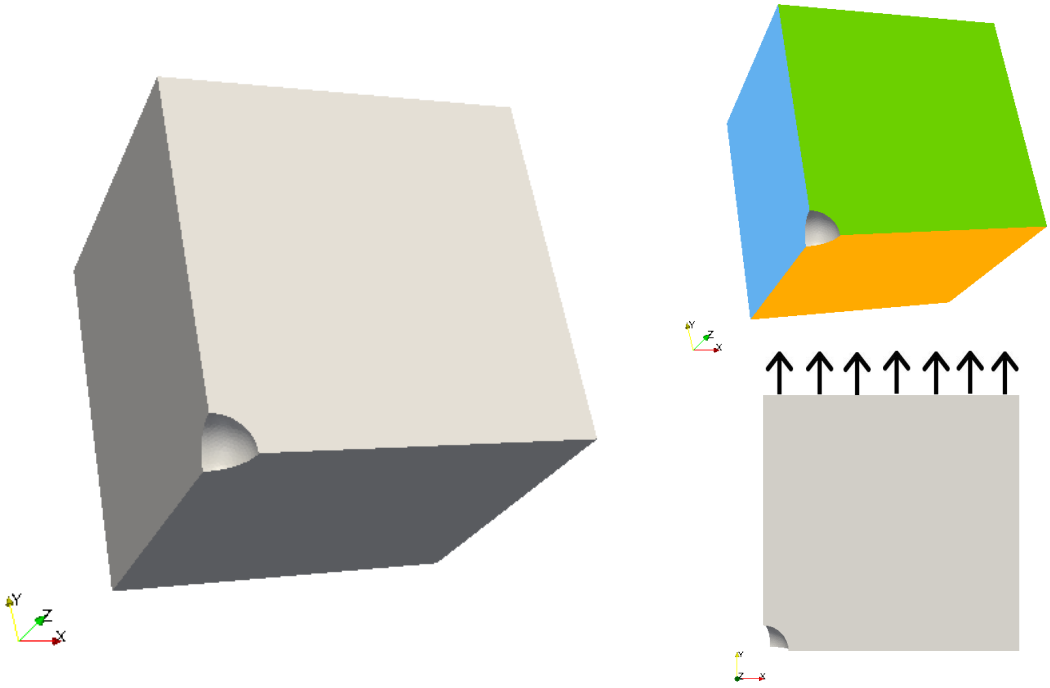

Figure 11: Reduced problem and boundary conditions

The three colored faces are submitted to the following boundary conditions : normal translation to the face and rotations in the plane of the face are blocked. The mesh of the structure, Figure 12, is realised with the *gmsh* package which permits to do a progressive mesh with tetrahedron elements of degree 2 (code in Appendix 3), tetrahedron elements of degree 1 are too stiff.

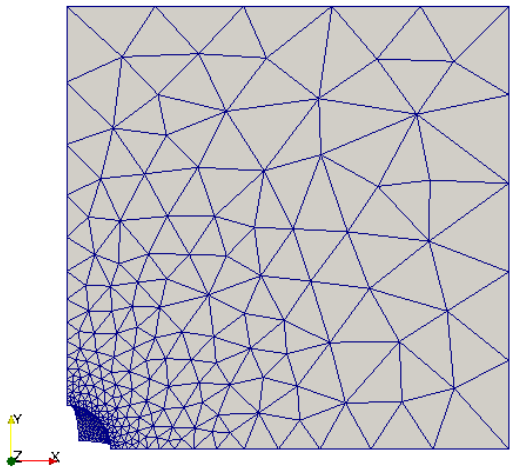

Figure 12: Mesh of the structure

The stress concentration factor (SCF) in classical elasticity and couple-stress theory for this 3D problem are given by [10] and [11] :

$$SFC_{classical\ elasticity} = \frac{3(9-5\nu)}{2(7-5\nu)} \quad \text{, and} \quad SFC_{couple-stress} = \frac{3[9-5\nu+6k'(1-\nu)(1+k)]}{2[7-5\nu+18k'(1-\nu)(1+k)]} \quad ,$$

with $k = R/l$, $k' = \frac{(3+\eta)}{9+9k+4k^2+k^3+\eta(3+3k+k^2)}$, where η is the ratio of the transverse curvature to the principal curvature (-1 ≤ η ≤ 1). The couple-stress theory corresponds to a limiting case of Cosserat theory in which N is near to 1. Because we only apply a traction force, the weak form (5) is simplified as :

$$\iiint_V (\sigma_{ij} . \epsilon_{ij} + m_{ij} . \chi_{ij}) dV = \iint_S (\boldsymbol{t}_i . \boldsymbol{u}_i) dS \quad . \qquad (6)$$

We first realise a convergence study using different mesh densities. We have to verify that the error between the computed and analytical stress concentration factors decreases with the size of the elements. Then we study the effect of the variation of different parameters. In classical elasticity, the SCF is evaluated depending on the Poisson's ratio. In Cosserat elasticity, we evaluate the evolution of the SCF depending on the Poisson's ratio ν, the characteristic length l, and the coupling number N.

4.2. Numerical results

For the numerical application G = 1000, the cube dimension is 100 and the radius of the spherical cavity is 10.

4.2.i) Classical elasticity

The analytical SCF is equal to 2.045 for ν = 0.3. We compare this value with the computed SCF. The stress around the cavity is shown in Figure 13.

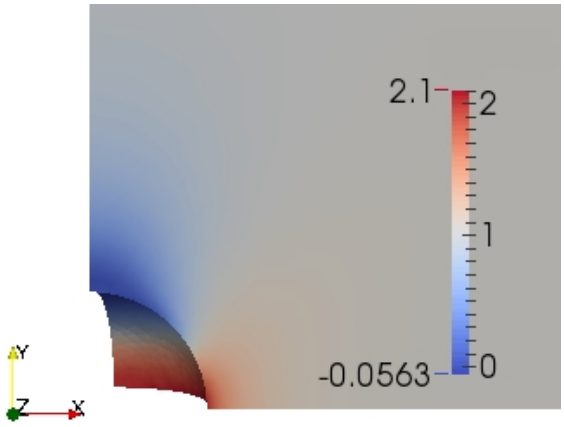

Figure 13: Stress around the cavity (σ_{yy})

For this mesh density, the computed stress concentration factor is equal to 2.1. This value well approximate the analytical solution but we have to realise a convergence study using different mesh densities. The SCF is computed for 4 different meshes (Python code in Appendix 3). The result of the convergence study is shown in Figure 14.

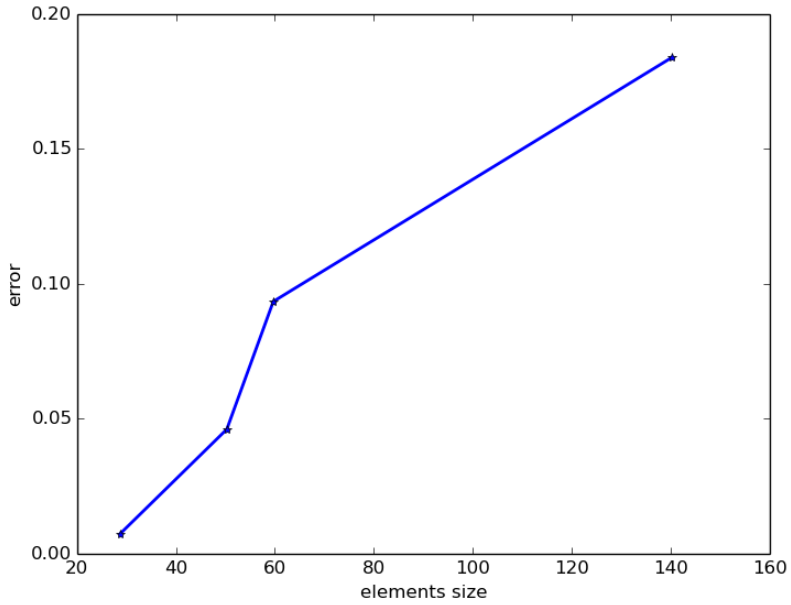

Figure 14: Graph of the error in SCF depending on the elements size

The error decreases with the elements size, thus the computed solution converges to the analytical SCF.

For evaluate the evolution of the SCF depending on ν, we use the finest mesh to have a minimal error in the results. The values of the analytical and computed SCF are shown in the Table 2.

ν	Analytical SCF	Computed SCF	Error (%)
0.0	1.928	1.917	0.57
0.1	1.961	1.949	0.61
0.2	2.000	1.987	0.65
0.3	2.045	2.031	0.68
0.4	2.100	2.082	0.85
0.49	2.159	2.127	1.48

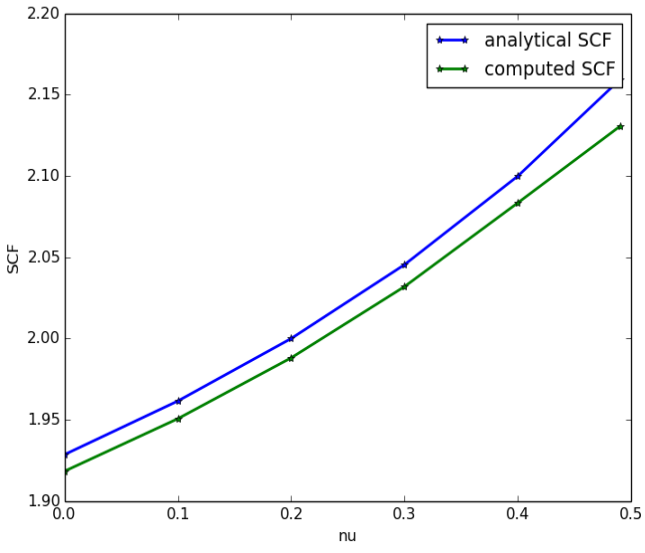

Table 2: SCF depending on ν (in classical elasticity)

The computed SCF well approximate the analytical SCF with an error near to 1% for the different values of the Poisson's ratio. We see that the maximum stress around the cavity increases with ν.

4.2.ii) Cosserat elasticity

The weak form (6) is used in the computation of the solution. We first evaluate the computed SCF in the structure and we compare it with the analytical SCF given by [10]. For this study ν = 0.3, l = 0.2 and N = 0.93. The graph of the convergence study is shown in Figure 15.

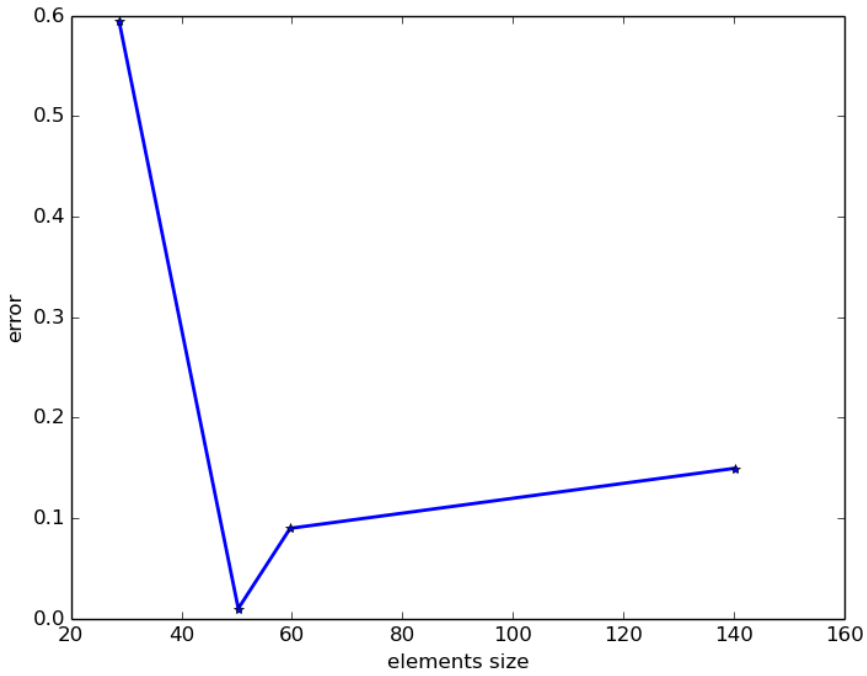

Figure 15: Graph of the error in SCF depending on the elements size

The computed solution does not converge to the analytical SCF. The weak form implemented in the code (Appendix 3) is for the Cosserat theory in which 0 < N < 0.9. But the analytical expression of the SCF is for the couple-stress theory in which N is near to 1. This value of N introduces an error in the computation the solution, thus the solution can't match with the analytical SCF. Moreover realise the comparison with 0 < N < 0.9 does not make sense, because the analytical solution would be incorrect.

To evaluate the differences between Cosserat theory and couple-stress theory, we evaluate the SCF depending on different varying parameters. The results presented in Table 3 correspond to the value of l = 0.2, N = 0.93 and various values of ν.

ν	Analytical SCF	Computed SCF	Error (%)
0.1	1.909	1.964	2.88
0.2	1.948	2.028	4.11
0.3	1.995	2.158	8.17
0.4	2.050	2.436	18.83
0.49	2.111	4.013	90.09

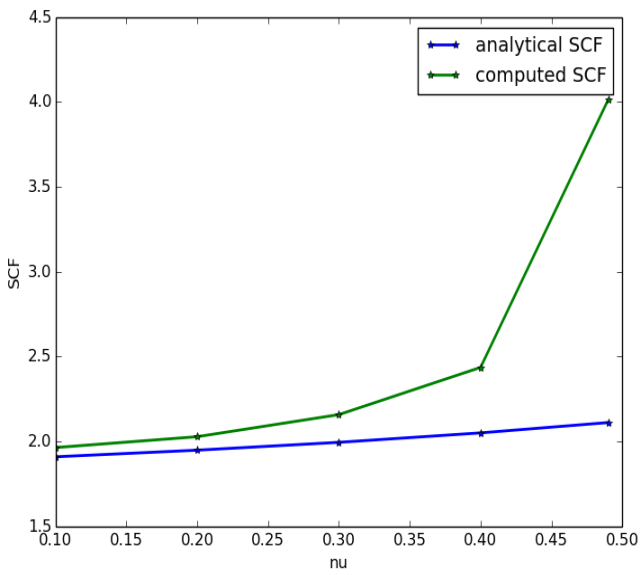

Table 3: SCF depending on ν ; N = 0.93, l = 0.2

For ν = 0.49, an important error is observed. It is because of this value of the Poisson's ratio is a limiting case of the linear elasticity. The value ν = 0.5 is for incompressible materials, like rubber. Near to this value, the computed solution differs from the analytical solution because of the numerical locking. Moreover, we can see that when the Poisson's ratio and the coupling number take limiting values the computed SCF mismatches with the couple-stress SCF with an error of 90%.

The results presented in Table 4 correspond to the value of ν = 0.3, N = 0.93 and various values of l.

l	Analytical SCF	Computed SCF	Error (%)
1.0E-4	2.045	2.146	4.94
R/5	1.717	1.981	15.4
R/4	1.674	1.953	16.6
R/3	1.617	1.927	19.2
R/2	1.543	1.903	23.3
R/1	1.450	1.887	30.1

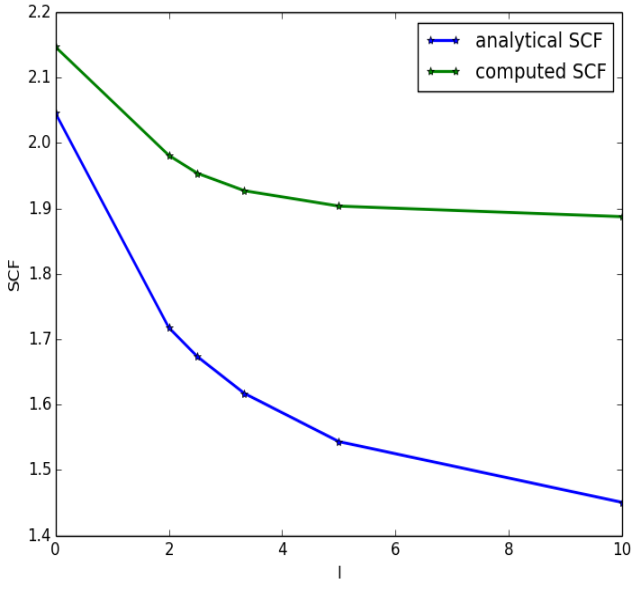

Table 4: SCF depending on l ; N = 0.93, ν = 0.3

For a value of l near to 0, the computed and analytical values are the same as in classical elasticity. This value of l removes the micro-rotation introduced by Cosserat theory. However for the other

values of l, the couple-stress theory can't be used to evaluate the SCF of a Cosserat solid. With N near to 0.9, we expected the computed solution to match with the analytical SCF with a minimal error. We see that the error increases with the l value, thus the analytical SCF given by Mindlin [10] could approximate the Cosserat solution for a intrisic length scale inferior to R/5.

The results presented in Table 5 correspond to the value of ν = 0.3, l = 0.2 and various values of N.

N	Analytical SCF	Computed SCF	Error (%)
0.001	1.995	2.141	7.31
0.25	1.995	2.144	7.46
0.5	1.995	2.137	7.11
0.75	1.995	1.970	1.25
0.9	1.995	1.994	0.05

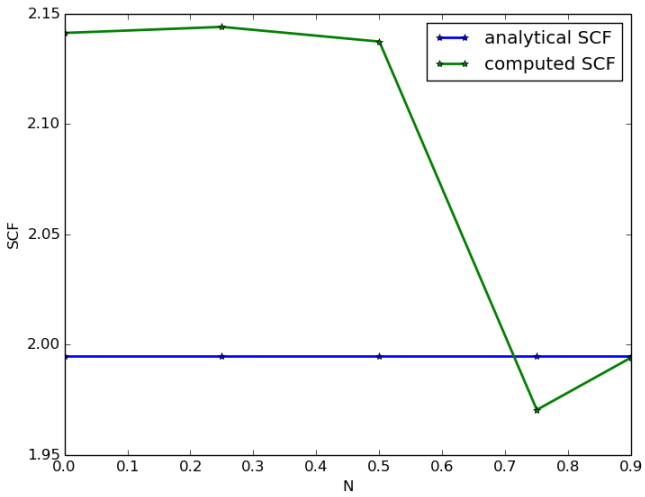

Table 5: SCF depending on N ; ν = 0.3, l = 0.2

The expression of the analytical SCF does not depend on N. We can see that the computed solution does not match with the exact value. This difference is due to the couple-stress theory in which N is near to 1. By varying this parameter, we have to use the Cosserat theory. It should be noted that for N = 0.9, the computed solution matches with the analytical SCF : couple-stress theory can be used in this case.

5. Conclusion

The main objective of the internship is the extension of the 2D implementation of Cosserat elasticity to the 3D case. The first part of the realised work is the convergence study of computed solution for a Timoshenko's beam. The second part is about the Cosserat theory applied to a 2D plate with a circular hole. The stress concentration factor is evaluated around the hole and compared with the analytical expression. The third part is the extension of the existing 2D implementation to the 3D case. The computed stress concentration factor is compared to the analytical solution given for the couple-stress theory.

In the computation of the displacements solution for a Timoshenko's beam, we must use particular boundary conditions to have the computed solution converging to the exact solution given by Timoshenko. The study of convergence shows that the order 2 solution converges faster than order 1 with an average error of 10^{-6}. But the solution computed with elements of degree 3 matches with the exact solution with an average error of 10^{-10}. This result is due to the order of the polynomial exact solution which is at the degree 3. Therefore in the Timoshenko's beam problem, using the analytical solution for boundary conditions, elements must be at degree 3 to have the best computed solution.

The plate with a circular hole is implemented like a plane strain problem. In this part an existing implement in 2D is used to realise the convergence study. The value of computed stress concentration factor is compared to an analytical expression using modified Bessel functions. This expression gives results near to the analytical expression in classical elasticity in which the stress concentration factor is equal to 3F when a traction load F is applied. In the extension of this problem, a spherical cavity in an infinite elastic medium is modeled by a cube with a quarter sphere inclusion at a corner. The unit load is applied on the top face and stress concentration factor is evaluated around the spherical hole. The computed value is compared to an analytical expression given in couple-stress theory. This theory is a limiting case of Cosserat elasticity using a coupling number N near to 1. The convergence study shows that the computed solution does not match with the analytical stress concentration factor because $N = 0.9$ is the maximum value for a Cosserat solid. To evaluate the difference between these two theories, the stress concentration factors are computed for different varying parameters. The variation of N demonstrates that the two solutions only match, with an error of 0.05 %, for $N = 0.9$. This value is the border between the two theories.

The Research Unit in Engineering Science uses high performance computers, thus a future work could be the generation of very large models from image data of microstructures. The meshing software from Simpleware Ltd permits to convert 3D scan datas into finite element models. Meshes from this software can be used in the Python codes produced during this internship for the computation of more complex problems in Cosserat elasticity.

6. References

[1] C.E. Augarde, A.J. Deeks, The use of Timoshenko's exact solution for a cantilever beam in adaptive analysis, Finite Elem. Anal. (2008).

[2] S.P. Timoshenko, J.N. Goodier, Theory of Elasticity, McGraw-Hill, New York, 1970.

[3] T. Belytschko, Y.Y. Lu, L. Gu, Element-free Galerkin methods, Int. J. Numer. Methods Eng. 37 (1994) 229–256.

[4] J. Hale (2013) Meshless methods for shear-deformable beams and plates based on mixed weak forms. Thesis, Imperial College London, London, United Kingdom.

[5] Cosserat E. et F. Théorie des Corps Déformables, Librairie Scientifique A. Hermann et Fils, Paris, 1909.

[6] R.D. Mindlin, Stress functions for a Cosserat continuum, Int. J. Solids Structures 1 (1965) 265-271.

[7] A.C. Eringen, Theory of micropolar elasticity, Technical Report No.1 (1967).

[8] R. Lakes, Experimental methods for study of cosserat elastic solids and other generalized elastic continua, in Continuum models for materials with micro-structure, 1995.

[9] E. Providas, M.A. Kattis, Finite element method in plane Cosserat elasticity, Computers and Structures 80 (2002) pp 2059–2069.

[10] R.D. Mindlin and H.F. Tiersten, Effects of couple stresses in linear elasticity, Archive of Rational mechanics and analysis (1962) pp 442-443.

[11] M. H. Sadd, Elasticity : Theory, applications and numerics, Elsevier (2004) pp 360-361

7. Appendices

7.1. Appendix 1 : Python codes for Timoshenko's beam

The two following Python codes are used in the convergence study of displacements and stress computed in a Timoshenko's beam. The error between analytical and computed solutions is plotted depending on the elements size for 3 different degree of elements.

convergence.py

```
# Convergence of the computed solution depending on the mesh density

import matplotlib.pyplot as plt
import numpy as np
import math as math

from solver import Calculate

# Degree
degree = 1

# Parameters
nu = 0.3
E = 210.0
T = 1.0e3 # thickness
D = 2.0e3 # depth
L = 10.0e3 # length
I = (T*D**3)/12.0
P = 1000.0

# Convergence
h = [10,20,30,40,50,60] # mesh density
d = [1,2,3]

elements_size = []
errors = np.zeros((len(d),len(h)))

i = 0
for hx in h :
      hy = int(0.5*hx)
      for degree in d :
            error_L2, size = Calculate(E, nu, I, D, L, P, degree, hx, hy)
            errors[degree-1,i] = math.log10(error_L2)
      elements_size.append(math.log10(size))
      i = i+1

print elements_size
print errors

plt.plot(elements_size, errors[0,:], "-*", label="order 1", linewidth=2)
plt.plot(elements_size, errors[1,:], "-*", label="order 2", linewidth=2)
plt.plot(elements_size, errors[2,:], "-*", label="order 3", linewidth=2)
plt.legend()
plt.xlabel("elements size (log h)")
plt.ylabel("error in norm L2 (log e)")
plt.show()
```

solver.py *(called in convergence.py)*

```
# Solver used in convergence.py
# Computation of the displacement in the beam
# Error with the analytical solution

from dolfin import *

def Calculate(E, nu, I, D, L, P, degree, hx, hy):

# Parameters
      lamb = (E*nu) / ((1.0+nu)*(1.0-2.0*nu))
      mu = E / (2.0*(1.0+nu))

      # Analytical solution
      class AnalyticalSolution(Expression):
            def __init__(self, E, nu, I, D, L, P):
                  self.nu = nu/(1.0 - nu)
                  self.E = E/(1.0 - nu**2.0)

                  self.I = I
                  self.D = D
                  self.L = L
                  self.P = P

            def eval(self,value,x):
                  E = self.E
                  nu = self.nu
                  I = self.I
                  D = self.D
                  L = self.L
                  P = self.P

                  value[0] = ((P*x[1])/(6.0*E*I)) * \
                        ((6.0*L - 3.0*x[0])*x[0]+(2.0 + nu)*(x[1]**2.0) -\
                        (1.5*(D**2.0))*(1.0+nu) )

                  value[1] = ((-P)/(6.0*E*I)) * \
                        ( (3.0*nu*(x[1]**2.0))*(L - x[0]) + \
                        (3.0*L - x[0])*(x[0]**2.0) )

            def value_shape(self):
                  return (2,)

      u_e = AnalyticalSolution(E,  nu,  I,  D,  L,  P)

      #
      class Charge(Expression):
            def __init__(self, E, I, D, P):
                  self.nu = nu/(1.0 - nu)
                  self.E = E/(1.0 - nu**2.0)

                  self.I = I
                  self.D = D
                  self.P = P

            def eval(self,value,x):
                  E = self.E
                  I = self.I
                  D = self.D
                  P = self.P
```

```
            value[0] = 0.0

            value[1] = - (P/(2.0*I)) * (- x[1]**2.0 + (D/2.0)**2.0 )

      def value_shape(self):
            return (2,)

F = Charge(E, I, D, P)

# Mesh
mesh = RectangleMesh(0.0, -D/2.0, L, D/2.0, hx, hy, "right")
#plot(mesh, interactive=True)
V = VectorFunctionSpace(mesh, "CG", degree)

# BC
class LeftEdge(SubDomain):
      def inside(self, x, on_boundary):
            tol = 1e-6
            return on_boundary and abs(x[0]) < tol

class RightEdge(SubDomain):
      def inside(self, x, on_boundary):
            tol = 1e-6
            return on_boundary and abs(x[0] - L) < tol

left_edge = LeftEdge()
right_edge = RightEdge()

sub_domains = MeshFunction("size_t", mesh, mesh.topology().dim() - 1)
sub_domains.set_all(0)
right_edge.mark(sub_domains, 1)
left_edge.mark(sub_domains, 2)

bc = DirichletBC(V,  u_e,  left_edge)
ds = Measure("ds")[sub_domains]

# Variational problem
u = TrialFunction(V)
v = TestFunction(V)

sigma = lambda v : 2.0*mu*sym(grad(v)) +
      lamb * tr(sym(grad(v))) * Identity(2)

a = inner(sigma(u), grad(v))*dx
L = inner(F,v)*ds(1)

# Solve
u_h = Function(V)

problem = LinearVariationalProblem(a,  L,  u_h,  bc)
solver  = LinearVariationalSolver(problem)
solver.solve()

# Error
error_L2 = errornorm(u_e, u_h, "L2")/norm(u_h, "L2")

hm = mesh.hmax()

print "Degree : " + str(degree)
print "Elements size : " + str(hm)
```

```
      print "Error in L2 norm: " + str(error_L2)

      return error_L2, hm
```

7.2. Appendix 2 : Python code for plate with hole

The following Python code permits the computation of the error between analytical and computed SCF around the circular hole. The error is plotted depending on the elements size.

plate_with_hole.py

```
# Computation of the solution in the plate for different meshes

from dolfin import *
import numpy as np
from mshr import Rectangle, Circle, generate_mesh
import matplotlib.pyplot as plt

# Parameters
R = 10.0 # radius
plate = 100.0 # plate dimension

nu = 0.3 # Poisson's ratio
G = 1000.0 # shear modulus

l = 0.2 # intrinsic length scale
N = 0.8 # coupling parameter
T = 1.0 # load
c = l/N

# Convergence
h = [15, 30, 50, 70, 90] # mesh density

elements_size = []
errors = []
SCF_0 = []

# Analytical solution
def AnalyticalSolution(nu, l, c, R):

      # Modified Bessel function of the second kind of real order v :
      from scipy.special import kv

      F = 8.0*(1.0-nu)*((l**2)/(c**2)) * \
            1.0 / (( 4.0 + ((R**2)/(c**2)) + \
            ((2.0*R)/c) * kv(0,R/c)/kv(1,R/c) ))

      SCF = (3.0 + F) / (1.0 + F) # stress concentration factor

      return (SCF)

SCF = AnalyticalSolution(nu, l, c, R)

# Matrix
def D_Matrix(G, nu, l, N):
      d = np.array([ \
            [(2.0*(1.0 - nu))/(1.0 - 2.0*nu), (2.0*nu)/(1.0 - 2.0 * nu),
0.0,0.0,0.0,0.0], \
            [(2.0*nu)/(1.0 - 2.0*nu), (2.0*(1.0 - nu)) / (1.0 - 2.0*nu),
0.0,0.0,0.0,0.0], \
            [0.0,0.0, 1.0/(1.0 - N**2), (1.0 - 2.0*N**2)/(1.0 - N**2),
```

```
0.0,0.0], \
            [0.0,0.0, (1.0 - 2.0*N**2)/(1.0 - N**2), 1.0/(1.0 - N**2),
0.0,0.0], \
            [0.0,0.0,0.0,0.0, 4.0*l**2, 0.0], \
            [0.0,0.0,0.0,0.0,0.0, 4.0*l**2] ])

      d *= G
      D = Constant(d)

      return D

# Strain
def strain(v, eta):
      strain = as_vector([ \
            v[0].dx(0),
            v[1].dx(1),
            v[1].dx(0) - eta,
            v[0].dx(1) + eta,
            eta.dx(0),
            eta.dx(1)])

      return strain

for hx in h :

      # Mesh
      geometry = Rectangle(Point(0,0),Point(plate, plate)) - \
            Circle(Point(0,0), R, hx)
      mesh = generate_mesh(geometry, hx)
      hm = mesh.hmax()

      U = VectorFunctionSpace(mesh, "Lagrange", 2, 2) # disp space
      S = FunctionSpace(mesh, "Lagrange", 1) # micro rotation space

      V = MixedFunctionSpace([U, S])
      U, S = V.split()
      U_1, U_2 = U.split()

      # Boundary conditions
      class BotBoundary(SubDomain):
            def inside(self, x, on_boundary):
                  tol = 1e-6
                  return on_boundary and abs(x[1]) < tol

      class LeftBoundary(SubDomain):
            def inside(self, x, on_boundary):
                  tol = 1e-6
                  return on_boundary and abs(x[0]) < tol

      class TopBoundary(SubDomain):
            def inside(self,x,on_boundary):
                  tol = 1e-6
                  return on_boundary and abs(x[1] - plate) < tol

      t = Constant((0.0, T))
      boundary_parts = MeshFunction("size_t", mesh, mesh.topology().dim() - 1)
      boundary_parts.set_all(0)

      bot_boundary = BotBoundary()
      left_boundary = LeftBoundary()
      top_boundary = TopBoundary()
```

```
        top_boundary.mark(boundary_parts, 1)

        ds = Measure("ds")[boundary_parts]

        u_0 = Constant(0.0)
        left_U_1 = DirichletBC(U_1, u_0, left_boundary)
        bot_U_2 = DirichletBC(U_2, u_0, bot_boundary)
        left_S = DirichletBC(S, u_0, left_boundary)
        bot_S = DirichletBC(S, u_0, bot_boundary)

        bc = [left_U_1, bot_U_2, left_S, bot_S]

        # Variational problem
        u, psi = TrialFunctions(V)
        v, eta = TestFunctions(V)

        D = D_Matrix(G, nu, l, N)

        a = inner(strain(v, eta), D*strain(u, psi))*dx
        L = inner(t, v)*ds(1)

        U_h = Function(V)
        problem = LinearVariationalProblem(a, L, U_h, bc)
        solver = LinearVariationalSolver(problem)
        solver.solve()
        u_h, psi_h = U_h.split()

        # Stress
        epsilon = strain(u_h, psi_h)
        sigma = D*epsilon
        sigma_yy = project(sigma[1])

        error = abs((sigma_yy(10.0, 1e-6) - SCF) / SCF)

        elements_size.append(hm)
        SCF_0.append(sigma_yy(10.0, 1e-6))
        errors.append(error)

print ("Analytical SCF : ") + str(SCF)
print elements_size
print errors
print SCF_0

file = File("sigma.pvd")
file << sigma_yy

plt.plot(elements_size, errors, "-*", linewidth=2)
plt.xlabel("elements size")
plt.ylabel("error")
plt.show()
```

7.3. Appendix 3 : Python code for 3D problem

In the following code the geometry of the problem is defined : an unit cube with a quarter sphere in the bottom left-hand corner.

mesh.geo *(converted in xml format and called in cosserat_3D.py)*

```
R = 10.0; // radius
D = 100.0; // cude dimension
```

```
f = 1; // fine mesh around the inclusion
c = 20; // coarse mesh

Point(1) = {0,0,0,f} ; Point(2) = {R,0,0,f} ; Point(3) = {D,0,0,c} ;
Point(4) = {D,0,-D,c} ;
Point(5) = {0,0,-D,c} ; Point(6) = {0,0,-R,f} ;

Point(7) = {0,D,0,c} ; Point(8) = {D,D,0,c} ;
Point(9) = {D,D,-D,c} ; Point(10) = {0,D,-D,c} ;
Point(11) = {0,R,0,f} ;

Circle(1) = {2,1,6} ; Circle(2) = {11,1,2} ; Circle(3) = {6,1,11} ;
Line(4) = {2,3} ; Line(5) = {3,4} ; Line(6) = {4,5} ; Line(7) = {5,6} ;
Line(8) = {7,8} ; Line(9) = {8,9} ; Line(10) = {9,10} ; Line(11) = {10,7} ;
Line(12) = {7,11} ; Line(13) = {8,3} ; Line(14) = {9,4} ; Line(15) = {10,5} ;

Line Loop(100) = {4,5,6,7,-1} ; Plane Surface(100) = {100} ;
Line Loop(101) = {5,-14,-9,13} ; Plane Surface(101) = {101} ;
Line Loop(102) = {14,6,-15,-10} ; Plane Surface(102) = {102} ;
Line Loop(103) = {7,3,-12,-11,15} ; Plane Surface(103) = {103} ;
Line Loop(104) = {8,9,10,11} ; Plane Surface(104) = {104} ;
Line Loop(105) = {4,-13,-8,12,2} ; Plane Surface(105) = {105} ;
Line Loop(106) = {1,3,2} ; Ruled Surface(106) = {106} ;

Surface Loop(200) = {106,100,101,102,103,104,105} ;
Volume(200) = {200} ;
```

This two codes realise the convergence study for the 3D case in Cosserat elasticity. Analytical SCF of the couple-stress theory is computed and compared with the computed SCF in Cosserat elasticity. The error is plotted depending on the mesh density.

convergence_3D_cosserat.py

```
# Plot the error in SCF depending on the elements size

from dolfin import *
import matplotlib.pyplot as plt

from solver_3D import computation

# Parameters
R = 10.0 # radius
cube = 100.0 # dim

T = 1.0 # traction force

nu = 0.3 # Poisson's ratio
mu = 1000.0 # shear modulus G
lmbda = ( 2.0 * mu * nu ) / (1.0-2.0*nu) # 1st Lame constant

l = 0.2 # intrinsic length scale
N = 0.93 # coupling parameter

# Analytical solution
def AnalyticalSolution(R, l, nu):

      k = R / l
      eta = 0.2 # ratio of the transverse curvature to the principal curvature

      k_1 = (3.0+eta) / ( 9.0 + 9.0*k + 4.0*k**2 + eta*(3.0 + 3.0*k + k**2) )
```

```
      SCF = ( 3.0*(9.0 - 5.0*nu + 6.0*k_1*(1.0-nu)*(1.0+k)) ) / \
            ( 2.0*(7.0 - 5.0*nu + 18.0*k_1*(1.0-nu)*(1.0+k)) )

      return (SCF)

SCF_a = AnalyticalSolution(R, l, nu)

error = []
h = []

mesh = Mesh("meshes/1.xml")
hm = mesh.hmax()
SCF_0 = computation(mesh, R, cube, T, nu, mu, lmbda, l, N)
e = abs(SCF_0 - SCF_a) / SCF_a
error.append(e)
h.append(hm)

mesh = Mesh("meshes/2.xml")
hm = mesh.hmax()
SCF_0 = computation(mesh, R, cube, T, nu, mu, lmbda, l, N)
e = abs(SCF_0 - SCF_a) / SCF_a
error.append(e)
h.append(hm)

mesh = Mesh("meshes/3.xml")
hm = mesh.hmax()
SCF_0 = computation(mesh, R, cube, T, nu, mu, lmbda, l, N)
e = abs(SCF_0 - SCF_a) / SCF_a
error.append(e)
h.append(hm)

mesh = Mesh("meshes/4.xml")
hm = mesh.hmax()
SCF_0 = computation(mesh, R, cube, T, nu, mu, lmbda, l, N)
e = abs(SCF_0 - SCF_a) / SCF_a
error.append(e)
h.append(hm)

plt.plot(h, error, "-*", linewidth=2)
plt.xlabel("elements size")
plt.ylabel("error")
plt.show()
```

solver_3D.cosserat.py *(called in convergence_3D_cosserat.py)*

```
# Computation of the solution in Cosserat elasticity

from dolfin import *

def computation(mesh, R, cube, T, nu, mu, lmbda, l, N) :

      # Micropolar elastic constants
      alpha = ( mu * N**2 ) /  (N**2 -1.0)
      beta = mu * l
      gamma = mu * l**2
      kappa = gamma

      # Strain and torsion
```

```
      def strain(v, eta):
            strain = as_tensor([ \
                  [ v[0].dx(0), \
                  v[1].dx(0) - eta[2], \
                  v[2].dx(0) + eta[1] ] , \

                  [ v[0].dx(1) + eta[2], \
                  v[1].dx(1), \
                  v[2].dx(1) - eta[0] ] , \

                  [ v[0].dx(2) - eta[1], \
                  v[1].dx(2) + eta[0], \
                  v[2].dx(2) ] ] )

            return strain

      def torsion(eta):
            torsion = as_tensor([ \
                  [ eta[0].dx(0), eta[1].dx(0), eta[2].dx(0) ], \
                  [ eta[0].dx(1), eta[1].dx(1), eta[2].dx(1) ], \
                  [ eta[0].dx(2), eta[1].dx(2), eta[2].dx(2) ] ])

            return torsion

      # Stress and couple stress
      def stress(lmbda, mu, kappa, epsilon) :
            stress = as_tensor([ \
                  [lmbda*epsilon[0,0]+(mu+kappa)*epsilon[0,0]+ mu*epsilon[0,0],
\
                  (mu+kappa)*epsilon[0,1] + mu*epsilon[1,0], \
                  (mu+kappa)*epsilon[0,2] + mu*epsilon[2,0] ], \

                  [ (mu+kappa)*epsilon[1,0] + mu*epsilon[0,1], \
                  lmbda*epsilon[1,1] + (mu+kappa)*epsilon[1,1] +
mu*epsilon[1,1], \
                  (mu+kappa)*epsilon[1,2] + mu*epsilon[2,1] ], \

                  [ (mu+kappa)*epsilon[2,0] + mu*epsilon[0,2], \
                  (mu+kappa)*epsilon[2,1] + mu*epsilon[1,2], \
                  lmbda*epsilon[2,2] + (mu+kappa)*epsilon[2,2] +
mu*epsilon[2,2]] ])

            return stress

      def couple(alpha, beta, gamma, chi) :
            couple = as_tensor([ \
                  [ (alpha + beta + gamma)*chi[0,0], \
                  beta*chi[1,0] + gamma*chi[0,1], \
                  beta*chi[2,0] + gamma*chi[0,2] ], \

                  [ beta*chi[0,1] + gamma*chi[1,0], \
                  (alpha + beta + gamma)*chi[1,1], \
                  beta*chi[2,1] + gamma*chi[1,2] ], \

                  [ beta*chi[0,2] + gamma*chi[2,0], \
                  beta*chi[1,2] + gamma*chi[2,1], \
                  (alpha + beta + gamma)*chi[2,2]] ])

            return couple

      # Function Space
```

```
U = VectorFunctionSpace(mesh, "Lagrange", 2, 3) # displacement space
S = VectorFunctionSpace(mesh, "Lagrange", 1, 3) # micro rotation space
V = MixedFunctionSpace([U, S]) # dim 6

U, S = V.split()
U_1, U_2, U_3 = U.split()
S_1, S_2, S_3 = S.split()

# Boundary conditions
class BotBoundary(SubDomain):
      def inside(self, x, on_boundary):
            tol = 1e-6
            return on_boundary and abs(x[1]) < tol

class LeftBoundary(SubDomain):
      def inside(self, x, on_boundary):
            tol = 1e-6
            return on_boundary and abs(x[0]) < tol

class FrontBoundary(SubDomain):
      def inside(self, x, on_boundary):
            tol = 1e-6
            return on_boundary and abs(x[2]) < tol

class TopBoundary(SubDomain):
      def inside(self,x,on_boundary):
            tol = 1e-6
            return on_boundary and abs(x[1] - cube) < tol

boundary_parts = MeshFunction("size_t", mesh, mesh.topology().dim() - 1)
boundary_parts.set_all(0)

bot_boundary = BotBoundary()
left_boundary = LeftBoundary()
front_boundary = FrontBoundary()
top_boundary = TopBoundary()
top_boundary.mark(boundary_parts, 1)

ds = Measure("ds")[boundary_parts]

u_0 = Constant(0.0)

left_U_1 = DirichletBC(U_1, u_0, left_boundary)
left_S_2 = DirichletBC(S_2, u_0, left_boundary)
left_S_3 = DirichletBC(S_3, u_0, left_boundary)

bot_U_2 = DirichletBC(U_2, u_0, bot_boundary)
bot_S_1 = DirichletBC(S_1, u_0, bot_boundary)
bot_S_3 = DirichletBC(S_3, u_0, bot_boundary)

front_U_3 = DirichletBC(U_3, u_0, front_boundary)
front_S_1 = DirichletBC(S_1, u_0, front_boundary)
front_S_2 = DirichletBC(S_2, u_0, front_boundary)

bcs = [left_U_1, left_S_2, left_S_3, bot_U_2, bot_S_1, \
      bot_S_3, front_U_3, front_S_1, front_S_2]

# Variational problem
u, phi = TrialFunctions(V)
v, eta = TestFunctions(V)
```

```
    epsilon_u = strain(u, phi)
    epsilon_v = strain(v, eta)
    chi_u = torsion(phi)
    chi_v = torsion(eta)

    sigma_u = stress(lmbda, mu, kappa, epsilon_u)
    sigma_v = stress(lmbda, mu, kappa, epsilon_v)
    m_u = couple(alpha, beta, gamma, chi_u)
    m_v = couple(alpha, beta, gamma, chi_v)

    t = Constant((0.0, T, 0.0))
    a = inner(epsilon_v, sigma_u)*dx + inner(chi_v, m_u)*dx
    L = inner(t, v)*ds(1)

    U_h = Function(V)
    problem = LinearVariationalProblem(a, L, U_h, bcs)
    solver = LinearVariationalSolver(problem)
    solver.solve()
    u_h, phi_h = U_h.split()

    epsilon_u_h = strain(u_h, phi_h)
    sigma_u_h = stress(lmbda, mu, kappa, epsilon_u_h)
    sigma_yy = project(sigma_u_h[1,1])
    SCF = sigma_yy(R, 0.0, 0.0)

    return SCF
```

Abstract

The present report exposes the realised work during an intership completed at the University of Luxembourg. The main objective is the extension of the existing 2D FEniCS implementation of Cosserat elasticity to the 3D case. The studied structure is an inclusion in an infinite elastic medium. The first model is a 2D plate with a circular hole, extended to the 3D infinite medium with a spherical cavity. The stress concentration factor computed around the inclusion is compared with analytical solutions from Cosserat theory in 2D or couple-stress theory in 3D. The evaluation of the stress concentration factor depending on different varying parameters permits to point the main difference between Cosserat theory and couple-stress theory developed by Mindlin, and also to predict the behaviour of Cosserat solids.

Résumé

Ce rapport expose le travail réalisé lors d'un stage à l'Université du Luxembourg. Le principal objectif est l'extension au cas 3D de l'implémentation FEniCS 2D en élasticité Cosserat. La structure étudiée est une inclusion dans un milieu élastique infini. Le premier modèle est une plaque 2D percée d'un trou circulaire, le modèle 3D est une cavité sphérique au sein d'un milieu élastique infini. Le facteur de concentration de contrainte calculé autour de l'inclusion est comparé avec les solutions analytiques de l'élasticité Cosserat en 2D ou avec celle de l'élasticité 'couple-stress' en 3D. L'évaluation du facteur de concentration de contrainte dépendant de différents paramètres variables permet de pointer la principale différence entre la théorie Cosserat et la théorie 'couple-stress' développée par Mindlin, et aussi d'appréhender le comportement des solides suivant le modèle de l'élasticité Cosserat.