

Master of Science in Advanced Mathematics and Mathematical Engineering

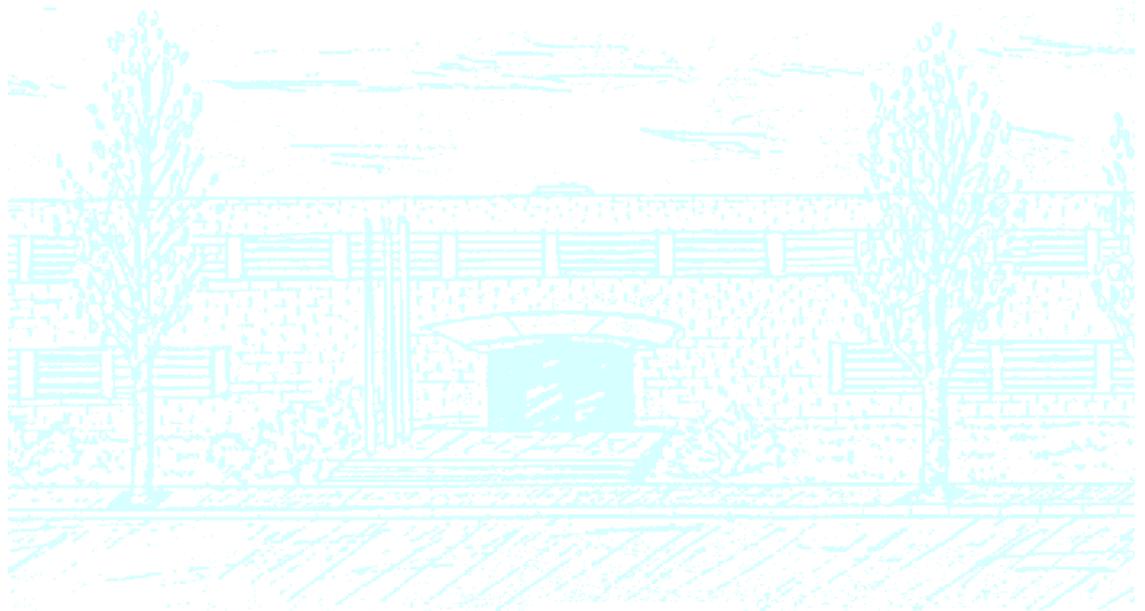
Title: Inverse mechanical analysis for biological tissues.

Author: Felix Lauwaert Luyten.

Advisor: Jose Javier Muñoz Romero.

Department: Departament de Matemàtiques.

Academic year: 2016





Index

1. Introduction	3
1.1. Background	3
1.2. Topic	3
2. Theoretical basis	5
2.1. Direct elasticity problem.....	5
2.2. Inverse elasticity problem.....	7
3. Methodology	9
3.1. Discrete non-regularized inverse problem	9
3.2. Regularised inverse problem.....	10
4. Results	13
4.1. Set 1	15
4.2. Set 2	17
5. Conclusions.....	20
6. Bibliography.....	21
Annexes.....	22
A. Code.....	22
A.1. main.m.....	22
A.2. AssembleA.m	23
A.3. bisection.m	24
A.4. ConvertDof.m.....	27
A.5. DefineBCRectangle.m.....	28
A.6. GenerateMesh.m	31
A.7. Inverse.m	33
A.8. keh8e.m.....	35
A.9. ke4e.m.....	37
A.10. Mask.m	39
A.11. MassMatrix.m	40
A.12. Me2le.m.....	42
A.13. Meq4e.m	43
A.14. Plotter.m.....	44
A.15. Solver.m.....	46



1. Introduction

1.1. Background

The branch of biology that studies biological tissues is a vast field that brings together many different areas of knowledge. Led in most cases by biologists, mathematical tools are often needed to carry out specific tasks involving modelling and simulation techniques.

This study originates from the need to learn about the growth of a fly embryo's spinal cord, specifically during its contraction phase. In this phase, the spinal cord has a spindly shape and it starts contracting to become shorter and wider. In this process, some unknown cells contract, leading to internal forces that deform the soft tissue up to its final shape.

The practical method used to solve this kind of problems is using the method called Traction Force Microscopy (TFM) (view figure 1). On TFM experiments, a cell monolayer is set on an elastic gel, which sticks to the basis (usually a Petri dish). The cells exert some tractions on the elastic gel which cannot be directly measured. However, the corresponding displacements to the elastic gel are, and from them it is possible to estimate those tractions.

However, TFM method has a limited applicability when treating events happening inside of living beings or other situations where it is not possible to set the experimental material.

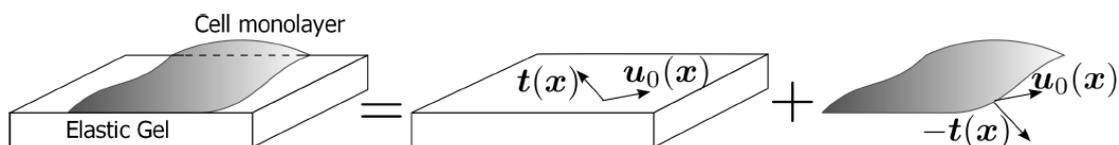


Figure 1: Traction Force Microscopy scheme.

1.2. Topic

This study focuses on the application of inverse finite element methods and computational mechanic techniques in order to simulate the spinal cord's contraction described in the previous section. Specifically, the aim is to know which forces lead to some imposed displacements and boundary conditions and to know the state of

deformation of the full body due to the input data. In order to acquire this objective, the TFM experiment will be discretized in an appropriate way in order to apply the inverse finite element method. An overview of this is seen on figures 2 and 3.

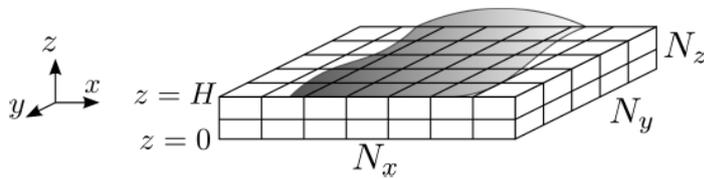


Figure 2: TFM setup discretized with a Cartesian two layer mesh.

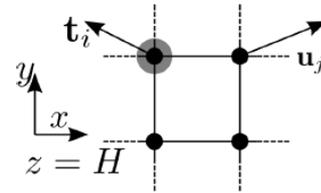


Figure 3: Scheme of the Cartesian mesh, using nodal reactions and displacements.

The scope of this study is to determine the uniqueness of the solution and its stability for different sets of external or internal forces and imposed displacements. This parameters will be studied through numerical analysis and theoretical results will be attempted. The results of the numerical computations will be physically interpreted and illustrated.

2. Theoretical basis

In order to compute the displacements and the tractions exerted by the spinal cord's cells it is convenient to consider it as an elasticity problem and use a finite elements method.

From a computational point of view, extracting the tractions t from measured displacements u_0 requires the solution of an inverse elasticity problem.

2.1. Direct elasticity problem.

In order to introduce the formulation of the inverse elasticity problem, it is convenient to formalise first the direct one. Thus, consider an open and connected domain $\Omega \subset \mathbb{R}^3$ and some boundary conditions defined by Dirichlet boundaries $\Gamma_D \neq \emptyset$ and Neumann boundaries Γ_N . The domain Ω follows a linear elastic constitutive law with Lamé coefficients $\lambda > 0$ and $\mu > 0$. The strong form of the linear elasticity problem may be stated as:

$$\nabla \cdot \boldsymbol{\sigma}(\mathbf{u}) = \mathbf{0}, \quad \forall x \in \text{int}(\Omega), \quad (1)$$

$$\boldsymbol{\sigma}(\mathbf{u})\mathbf{n} = \mathbf{t}, \quad \forall x \in \Gamma_N, \quad (2)$$

$$\mathbf{u} = \mathbf{0}, \quad \forall x \in \Gamma_D, \quad (3)$$

With $\boldsymbol{\sigma}(\mathbf{u}) = \lambda(\nabla \cdot \mathbf{u})\mathbf{I} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)$ denoting the stress tensor. It is assumed that $t \in T \subseteq L^2(\Gamma_N)$.

Defining the spaces U and V as:

$$U = V = (H_0^1(\Omega))^3 = \left\{ \mathbf{v} \in (H^1(\Omega))^3 : v_i = 0 \text{ on } \Gamma_D, i = 1,2,3 \right\},$$

and setting the scalar product $(\mathbf{u}, \mathbf{v}) = \int_{\Omega} \mathbf{u} \cdot \mathbf{v} d\Omega$ and the norm $\|\mathbf{v}\|_{\Omega} = (\sum_{i=1}^3 \|v_i\|_{1,\Omega}^2)^{1/2}$, where $\|v_i\|_{1,\Omega}^2 = \int_{\Omega} (|v_i|^2 + \nabla v_i \cdot \nabla v_i) d\Omega$.

After multiplying (1), (2) and (3) by a test function v and integrating by parts, the weak form reads:

$$\text{Find } \mathbf{u} \in U \text{ s. t. } a(\mathbf{u} \cdot \mathbf{v}) = b(\mathbf{v}), \quad \forall \mathbf{v} \in V. \quad (4)$$

The bilinear forms $a(\cdot, \cdot)$ and $b(\cdot)$ are given by:

$$a(\mathbf{u}, \mathbf{v}) := \int_{\Omega} \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) d\Omega,$$

$$b(\mathbf{v}) := \int_{\Gamma_N} \mathbf{v} \cdot \mathbf{t} d\Gamma,$$

where $\boldsymbol{\varepsilon}(\mathbf{v}) = \frac{1}{2}(\nabla \mathbf{v} + (\nabla \mathbf{v})^T)$ is the small strain tensor.

Since the bilinear form $a(\cdot, \cdot)$ is continuous and elliptic with respect to V , and assuming that $\Gamma_D \neq \emptyset$, equation (4) has a unique solution $\mathbf{u}[\mathbf{t}]$.

The domain Ω can be defined as $\Omega_0 \cup \Omega_1$ where $\Omega_0 \subseteq \Omega$ and $\Omega_1 = \Omega \setminus \Omega_0$. Suppose a known displacement \mathbf{u}_1 that satisfies the elasticity equations:

$$\begin{aligned} \nabla \cdot \boldsymbol{\sigma}(\mathbf{u}_1) &= \mathbf{0}, & \forall \mathbf{x} \in \text{int}(\Omega_1), \\ \boldsymbol{\sigma}(\mathbf{u}_1)\mathbf{n} &= \mathbf{t}, & \forall \mathbf{x} \in \Gamma_N \cap \Omega_1, \\ \mathbf{u}_1 &= \mathbf{0}, & \forall \mathbf{x} \in \Gamma_D \cap \Omega_1. \end{aligned} \quad (5)$$

Then, the solution \mathbf{u}_0 will be denoted by $\mathbf{u}[\mathbf{t}, \mathbf{u}_1]$ and satisfies the elasticity problem in Ω_0 in a compatible way with \mathbf{u}_1 and the boundary conditions (2) and (3):

$$\begin{aligned} \nabla \cdot \boldsymbol{\sigma}(\mathbf{u}_0) &= \mathbf{0}, & \forall \mathbf{x} \in \text{int}(\Omega_0), \\ \boldsymbol{\sigma}(\mathbf{u}_0)\mathbf{n} &= \mathbf{t}, & \forall \mathbf{x} \in \Gamma_N \cap \Omega_0, \\ \mathbf{u}_0 &= \mathbf{0}, & \forall \mathbf{x} \in \Gamma_D \cap \Omega_0, \\ \mathbf{u}_0 &= \mathbf{u}_1, & \forall \mathbf{x} \in \partial\Omega_0 \setminus (\Gamma_N \cup \Gamma_D). \end{aligned} \quad (6)$$

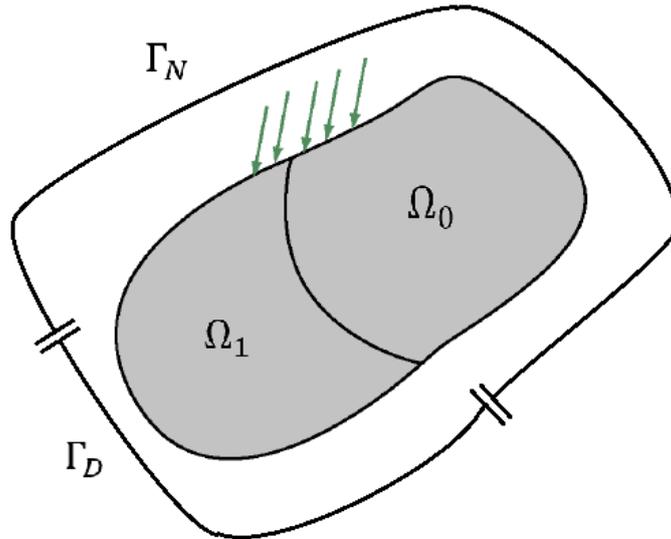


Figure 4: domain Ω divided in two subdomains: Ω_0 and Ω_1 , with both Dirichlet and Neumann boundary conditions. The green arrows show those Neumann boundary conditions different from 0.

As seen in Figure 4, the domain Ω is separated in two domains Ω_0 and Ω_1 , and the displacements corresponding to each domain are \mathbf{u}_0 and \mathbf{u}_1 respectively. The system is characterized by both Dirichlet and Neumann boundary conditions, where the former only contains nodes in the boundary of Ω_1 while the latter contains the boundaries in Ω_0 and Ω_1 .

2.2. Inverse elasticity problem

The continuous inverse problem in (5) and (6) assumes the knowledge of the displacements \mathbf{u}_0 and aims to find the traction field \mathbf{t} and the displacement field \mathbf{u}_1 . This problem is formally stated as:

$$\begin{aligned} \text{Given } \mathbf{u}_0 \in U_0(\Omega_0), \text{ find } \mathbf{t} \in T \subseteq L^2(\Gamma_N) \text{ and } \mathbf{u}_1 \quad \forall \mathbf{v} \in V \quad (7) \\ \in U_1(\Omega_1) \text{ s. t. } \bar{b}(\mathbf{t}, \mathbf{v}) \\ = \bar{a}(\mathbf{u}_1, \mathbf{v}) + \bar{c}(\mathbf{v}), \end{aligned}$$

where the forms $\bar{a}(\mathbf{u}_1, \mathbf{v})$, $\bar{b}(\mathbf{t}, \mathbf{v})$ and $\bar{c}(\mathbf{v})$ are given by:

$$\begin{aligned} \bar{a}(\mathbf{u}_1, \mathbf{v}) &:= \int_{\Omega_1} \boldsymbol{\sigma}(\mathbf{u}_1) : \boldsymbol{\varepsilon}(\mathbf{v}) d\Omega, \\ \bar{b}(\mathbf{t}, \mathbf{v}) &:= \int_{\Gamma_N} \mathbf{v} \cdot \mathbf{t} d\Gamma, \\ \bar{c}(\mathbf{v}) &:= \int_{\Omega_0} \boldsymbol{\sigma}(\mathbf{u}_0) : \boldsymbol{\varepsilon}(\mathbf{v}) d\Omega. \end{aligned}$$

The reference geometry and boundary conditions are be those depicted in Figure 5. The domain is an elastic gel where the Dirichlet known boundary conditions are $\mathbf{u} = \mathbf{0}$ at the gel bottom. \mathbf{u}_0 is experimentally read on some of the top and side boundaries, and \mathbf{T} is set to be satisfying mechanical equilibrium and defined on different boundaries that do not coincide with the Dirichlet ones.

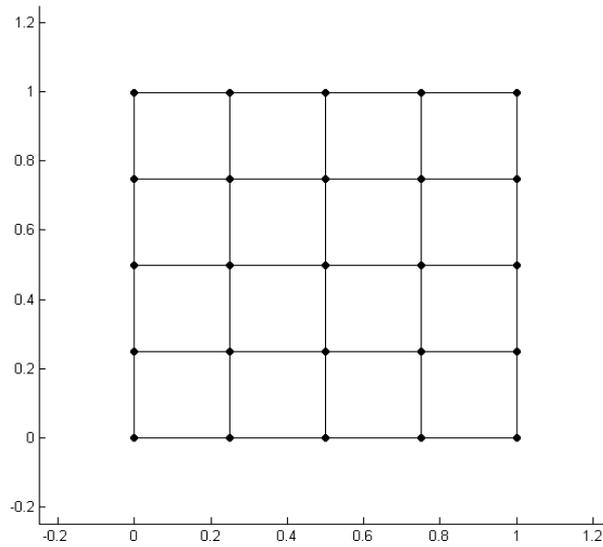


Figure 5: 2D square domain with quadrilateral elements.

The existence and uniqueness of the solution of (7) cannot be guaranteed due to the experimental errors of the experimental values of \mathbf{u}_0 and the finite element discretisation employed. Therefore, the methodology presented in the next chapter aims to find a discrete traction and displacement fields that minimize the error $\bar{a}(\mathbf{u}_1, \mathbf{v}) + \bar{c}(\mathbf{v}) - \bar{b}(\mathbf{t}, \mathbf{v})$ for an arbitrary test function \mathbf{v} .

3. Methodology

3.1. Discrete non-regularized inverse problem

Analysing \mathbf{u}_0 as a set of n_0 points $X = \{\mathbf{x}_1, \dots, \mathbf{x}_{n_0}\}$ of Ω_0 . The operator that extracts the values of a continuous field \mathbf{u}_0 on the set X can be denoted as O . That is, $O\mathbf{u}_0 = \{\mathbf{u}_0(\mathbf{x}_1), \dots, \mathbf{u}_0(\mathbf{x}_{n_0})\}$. Then, the inverse problem (7) is modified by defining the following functional:

$$\tilde{J}_0(\mathbf{t}, \mathbf{u}_1) := \|O\mathbf{u}[\mathbf{t}, \mathbf{u}_1] - O\mathbf{u}_0\|^2, \quad (8)$$

where $\|\cdot\|$ is the standard Euclidean norm in $\mathbb{R}^{3 \times n_0}$.

The discretized form of (8) reads:

$$\tilde{J}_0^h(\mathbf{t}, \mathbf{u}_1) := \|\mathbf{K}_0^{-1}(\mathbf{A}\mathbf{t} - \mathbf{K}_1\mathbf{u}_1) - \mathbf{u}_0\|^2, \quad (9)$$

Here, \mathbf{K}_0 and \mathbf{K}_1 are the standard stiffness matrices of the degrees of freedom with imposed displacements and the unknown displacements respectively. The matrix \mathbf{A} projects the boundary loads on nodal contributions.

The minimization of $\tilde{J}_0^h(\mathbf{t}, \mathbf{u}_1)$ would require to compute the inverse matrix \mathbf{K}_0^{-1} . Therefore, it is convenient to reinterpret the equation (9) using a different metric of the vector space. So, the problem can be transformed into minimising the following functional:

$$J_0^h(\mathbf{t}, \mathbf{u}_1) := \|\mathbf{K}_0\mathbf{u}_0 + \mathbf{K}_1\mathbf{u}_1 - \mathbf{A}\mathbf{t}\|^2, \quad (10)$$

Now, the minimization of $J_0^h(\mathbf{t}, \mathbf{u}_1)$, which corresponds to a least-square problem, leads to the following normal equations:

$$[\mathbf{A} \quad -\mathbf{K}_1]^T [\mathbf{A} \quad -\mathbf{K}_1] \begin{Bmatrix} \mathbf{t} \\ \mathbf{u}_1 \end{Bmatrix} = [\mathbf{A} \quad -\mathbf{K}_1]^T \mathbf{K}_0\mathbf{u}_0 \quad (11)$$

In order to rewrite the problem as the solution of \mathbf{t} and \mathbf{u}_1 in a partitioned manner and considering \mathbf{M} is positive definite (and thus, invertible), it is convenient to use the relation $\mathbf{A} = \widehat{\mathbf{A}}\mathbf{M}$ and $\tilde{\mathbf{I}} = \mathbf{I} - \widehat{\mathbf{A}}(\widehat{\mathbf{A}}^T\widehat{\mathbf{A}})^\dagger\widehat{\mathbf{A}}^T$, where $(\widehat{\mathbf{A}}^T\widehat{\mathbf{A}})^\dagger$ denotes the pseudo-inverse of $\widehat{\mathbf{A}}^T\widehat{\mathbf{A}}$, which is equal to the inverse when the matrix is invertible. Here, the matrix \mathbf{M} is the stiffness matrix associated to the boundary Γ_N . Then, (11) can be rewritten as:

$$\mathbf{K}_1^T \tilde{\mathbf{I}} \mathbf{K}_1 \mathbf{u}_1 = -\mathbf{K}_1^T \tilde{\mathbf{I}} \mathbf{K}_0 \mathbf{u}_0, \quad (12)$$

$$\mathbf{t} = \mathbf{M}^{-1} (\widehat{\mathbf{A}}^T \widehat{\mathbf{A}})^\dagger \widehat{\mathbf{A}}^T (\mathbf{K}_0 \mathbf{u}_0 + \mathbf{K}_1 \mathbf{u}_1), \quad (13)$$

This new form allows to compute \mathbf{u}_1 from (12) and then obtain \mathbf{t} using (13).

As stated in [1], equations (12) and (13) can be written as:

$$\mathbf{K}_1^T \tilde{\mathbf{I}}_0 \mathbf{K}_1 \mathbf{u}_1 = -\mathbf{K}_1^T \tilde{\mathbf{I}}_0 \mathbf{K}_0 \mathbf{u}_0, \quad (14)$$

$$\mathbf{t} = \mathbf{M}^{-1} \widehat{\mathbf{A}}^T (\mathbf{K}_0 \mathbf{u}_0 + \mathbf{K}_1 \mathbf{u}_1), \quad (15)$$

when $m \leq n_0$, with $\tilde{\mathbf{I}}_0 = \mathbf{I} - \widehat{\mathbf{A}} \widehat{\mathbf{A}}^T$.

It is shown in [1] that the system of equations is well-posed, and thus solvable when $m \leq n_0$. In this reference it was assumed that the domain of the traction field is included in the domain of the experimental displacements \mathbf{u}_0 . However, this requirement is not necessary. It has been numerically test in this work that, in fact, the equations above are solvable (that is, matrix on the left hand side of (14) is not singular) regardless of the locations of \mathbf{u}_0 and \mathbf{T} , as far as $m \leq n_0$. In this case, the matrix $\mathbf{A}^T \mathbf{A}$ is the identity matrix, and thus $\mathbf{K}_1^T \tilde{\mathbf{I}}_0 \mathbf{K}_1$ can be inverted.

Although it will not test it this project, the traction field denoted by \mathbf{T} may also represent internal forces or body forces. This opens the door to consider inverse problems for analysing active loads present in biological tissues.

3.2. Regularised inverse problem

The non-regularised inverse problem is only applicable when $m \leq n_0$. For $m > n_0$, the problem becomes ill-posed [1], and the pseudo-inverse $(\widehat{\mathbf{A}}^T \widehat{\mathbf{A}})^\dagger$ must be employed. The latter has a high computational cost and $\mathbf{K}_1^T \tilde{\mathbf{I}}_0 \mathbf{K}_1$ becomes close to singular, what leads to the loss of uniqueness of the solution for \mathbf{T} .

In order to deal with this situation, one of the most commonly used methods is the Tikhonov regularization [2]. This technique consists on introducing an additional term in the functional in (8), linearly dependent on λ , such that it turns into:

$$\tilde{J}_0(\mathbf{t}, \mathbf{u}_1) := \|\mathcal{O}\mathbf{u}[\mathbf{t}, \mathbf{u}_1] - \mathcal{O}\mathbf{u}_0\|^2 + \lambda \|\mathbf{t}\|^2, \quad (16)$$

In this situation, the new discretized expressions results to be:

$$J_0^h(\mathbf{t}, \mathbf{u}_1) := \|\mathbf{K}_0 \mathbf{u}_0 + \mathbf{K}_1 \mathbf{u}_1 - \mathbf{A} \mathbf{t}\|^2 + \lambda \|\mathbf{t}\|^2, \quad (17)$$

Applying the same proceedings as done in the previous section, the resulting expressions to be implemented in order to solve the problem are the same, but with a different expression for $\tilde{\mathbf{I}}_0$, which becomes $\tilde{\mathbf{I}} = \mathbf{I} - \widehat{\mathbf{A}}(\widehat{\mathbf{A}}^T \widehat{\mathbf{A}} + \lambda \mathbf{I})^{-1} \widehat{\mathbf{A}}^T$.

The introduction of a fictional parameter in the system leads to a worse approximation of the mechanical equilibrium, but solves at the same time the non-uniqueness of the solution. This two contrary characteristics can be measured by means of $\|\mathbf{K}_0 \mathbf{u}_0 + \mathbf{K}_1 \mathbf{u}_1 - \mathbf{A} \mathbf{t}\|$ and $\|\mathbf{t}\|$ respectively.

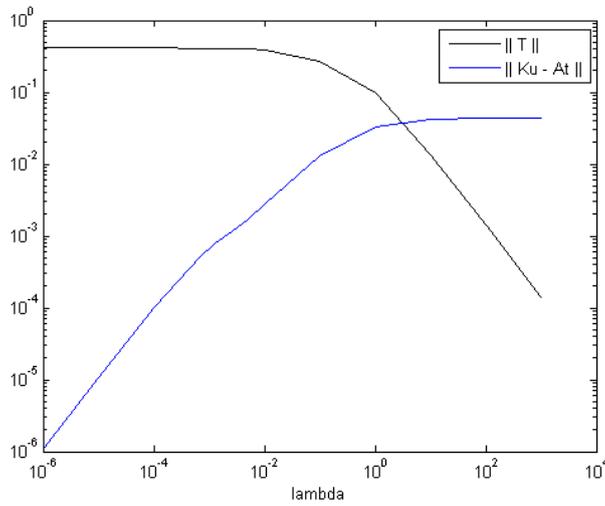


Figure 6: norms of T and mechanical equilibrium with respect to λ .

In Figure 6, there is an example of the evolution of these two parameters with respect to λ . It can be observed how the mechanical equilibrium is lost when λ increases, while the norm of the solution T decreases, since the functional tends to solely minimise the norm of the tractions. When regularisation is needed, that is when $m > n_0$ in the present case, the high values of $\|T\|$ for small λ represent solutions that belong to the null space of the system matrix, which are included in the solution and can be arbitrary large. Plotting one with respect to the other leads to the typical L-shape curve, shown in figure 7.

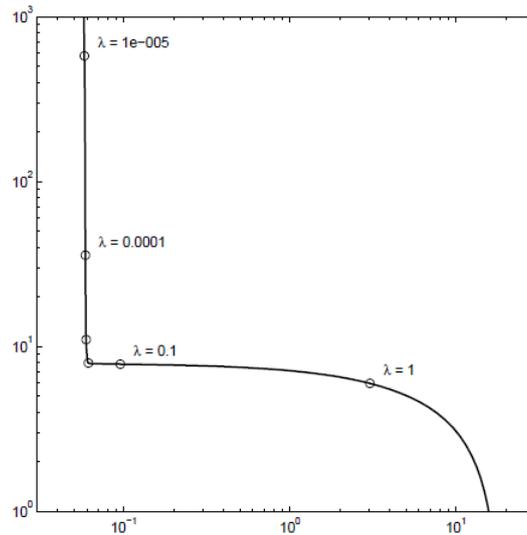


Figure 7: L-curve showing the norm of \mathbf{T} with respect to the mechanical equilibrium for different values of λ . [2]

This L-curve is really a trade-off between both quantities, which both should be controlled. In order to select a suitable value for λ that minimizes the error committed when obtaining the mechanical equilibrium and, at the same time, reduce the norm of the solution, a common method is to select the one that minimizes both with an acceptable error. This value of λ coincides with the vertex seen in Figure 7, when there is a sudden change in the behaviour of the L-curve. For small values of λ , the norm of the solution decreases rapidly when λ increases, until a point where the norm stabilizes and the residual starts to grow very fast when λ keeps increasing. Thus, selecting the value of λ that leads to this change in behaviour leads to a good compromise between both values.

4. Results

Several numerical computations have been done in order to check all the qualitatively different possible combinations of Ω_{u_0} and Ω_T . So, there have been proposed 9 different situations, explained in Figure 8. These situations will be tested in order to study the viability of the inverse problem and the Tikhonov regularization method for combinations of Ω_{u_0} and Ω_T when one is not a subset of the other.

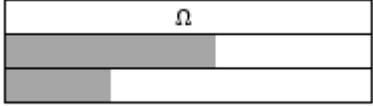
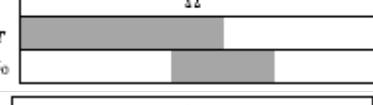
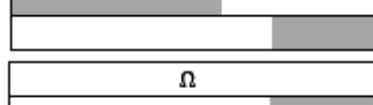
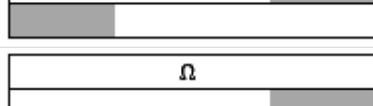
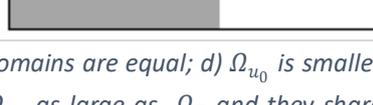
a	$\Omega_T \subset \Omega_{u_0} \ \& \ \Omega_T \cap \Omega_{u_0} \neq \emptyset$	$\left\{ \begin{array}{l} \Omega_T \\ \Omega_{u_0} \end{array} \right\}$ 
b	$\Omega_{u_0} \subset \Omega_T \ \& \ \Omega_T \cap \Omega_{u_0} \neq \emptyset$	$\left\{ \begin{array}{l} \Omega_T \\ \Omega_{u_0} \end{array} \right\}$ 
c	$\Omega_T = \Omega_{u_0}$	$\left\{ \begin{array}{l} \Omega_T \\ \Omega_{u_0} \end{array} \right\}$ 
d	$\Omega_{u_0} \not\subset \Omega_T \ \& \ \Omega_T \not\subset \Omega_{u_0} \ \& \ \Omega_T \cap \Omega_{u_0} \neq \emptyset \ \& \ \dim(T) > \dim(u_0) \neq \emptyset$	$\left\{ \begin{array}{l} \Omega_T \\ \Omega_{u_0} \end{array} \right\}$ 
e	$\Omega_{u_0} \not\subset \Omega_T \ \& \ \Omega_T \not\subset \Omega_{u_0} \ \& \ \Omega_T \cap \Omega_{u_0} \neq \emptyset \ \& \ \dim(T) = \dim(u_0) \neq \emptyset$	$\left\{ \begin{array}{l} \Omega_T \\ \Omega_{u_0} \end{array} \right\}$ 
f	$\Omega_{u_0} \not\subset \Omega_T \ \& \ \Omega_T \not\subset \Omega_{u_0} \ \& \ \Omega_T \cap \Omega_{u_0} \neq \emptyset \ \& \ \dim(T) < \dim(u_0) \neq \emptyset$	$\left\{ \begin{array}{l} \Omega_T \\ \Omega_{u_0} \end{array} \right\}$ 
g	$\Omega_T \cap \Omega_{u_0} = \emptyset \ \& \ \dim(T) > \dim(u_0) \neq \emptyset$	$\left\{ \begin{array}{l} \Omega_T \\ \Omega_{u_0} \end{array} \right\}$ 
h	$\Omega_T \cap \Omega_{u_0} = \emptyset \ \& \ \dim(T) = \dim(u_0) \neq \emptyset$	$\left\{ \begin{array}{l} \Omega_T \\ \Omega_{u_0} \end{array} \right\}$ 
i	$\Omega_T \cap \Omega_{u_0} = \emptyset \ \& \ \dim(T) < \dim(u_0) \neq \emptyset$	$\left\{ \begin{array}{l} \Omega_T \\ \Omega_{u_0} \end{array} \right\}$ 

Figure 8: a) Ω_T is a subset of Ω_{u_0} ; b) Ω_{u_0} is a subset of Ω_T ; c) both domains are equal; d) Ω_{u_0} is smaller than Ω_T and they share some, but not all, degrees of freedom; e) Ω_{u_0} as large as Ω_T and they share some, but not all, degrees of freedom; f) Ω_{u_0} is larger than Ω_T and they share some, but not all, degrees of freedom; g) Ω_{u_0} is smaller than Ω_T and they don't share any degree of freedom; h) Ω_{u_0} is as large as Ω_T and they don't share any degree of freedom; i) Ω_{u_0} is larger than Ω_T and they don't share any degree of freedom.

The numerical computations have been tested for all the cases explained above for two different domains. There have been two sets of computations: one with a square domain composed by 16 elements and another with 225 elements. These two sets have been performed in order to check the influence of the element size on the results. In both cases, the Young modulus and the viscosity used to compute the mass matrix were dimensionless, with values $E = 1$ and $\nu = 0.3$. All the values of the imposed displacements have been set to 1.

The degrees of freedom set in Ω_T and Ω_{u_0} vary from case to case:

- a) Ω_T is a subset of Ω_{u_0} :
 - a. Ω_T contains the left half of the top nodes in its horizontal direction.
 - b. Ω_{u_0} contains the left $\frac{3}{4}$ of the top nodes in its horizontal direction.
- b) Ω_{u_0} is a subset of Ω_T :
 - a. Ω_T contains the left $\frac{3}{4}$ of the top nodes in its horizontal direction.
 - b. Ω_{u_0} contains the left half of the top nodes in its horizontal direction.
- c) Both domains are equal, containing the left $\frac{3}{4}$ in its horizontal direction.
- d) Ω_{u_0} is smaller than Ω_T and they share some, but not all, degrees of freedom:
 - a. Ω_T contains the right $\frac{3}{4}$ of the top nodes in its horizontal direction and the first $\frac{1}{4}$ upper nodes of the right boundary in its vertical direction.
 - b. Ω_{u_0} contains the left half of the top nodes in its horizontal direction.
- e) Ω_{u_0} as large as Ω_T and they share some, but not all, degrees of freedom.
 - a. Ω_T contains the right $\frac{2}{3}$ of the top nodes in its horizontal direction.
 - b. Ω_{u_0} contains the left $\frac{2}{3}$ of the top nodes in its horizontal direction.
- f) Ω_{u_0} is larger than Ω_T and they share some, but not all, degrees of freedom.
 - a. Ω_T contains the left half of the top nodes in its horizontal direction
 - b. Ω_{u_0} contains the right $\frac{3}{4}$ of the top nodes in its horizontal direction and the first $\frac{1}{4}$ upper nodes of the right boundary in its vertical direction.
- g) Ω_{u_0} is smaller than Ω_T and they don't share any degree of freedom.
 - a. Ω_T contains the left $\frac{3}{4}$ of the top nodes in its horizontal direction.
 - b. Ω_{u_0} contains the left half of the top nodes in its vertical direction.
- h) Ω_{u_0} is as large as Ω_T and they don't share any degree of freedom.
 - a. Ω_T contains the left half of the top nodes in its horizontal direction.
 - b. Ω_{u_0} contains the left half of the top nodes in its vertical direction.
- i) Ω_{u_0} is larger than Ω_T and they don't share any degree of freedom.

- a. Ω_T contains the left half of the top nodes in its vertical direction.
- b. Ω_{u_0} contains the left $\frac{3}{4}$ of the top nodes in its horizontal direction.

4.1. Set 1

The results of the first set of numerical computations are shown in Table 1, seen below:

Set 1						
Case	κ	Mec. Equilibrium	Regularized ME	Solution norm	Regularized SN	λ
a	1,70E+03	3,17E-02	3,17E-02	4,02E-01	4,00E-01	3,85E-03
b	1,15E+16	1,99E-15	3,09E-04	1,35E+01	4,13E-01	2,17E-03
c	1,72E+03	2,35E-15	4,32E-04	4,28E-01	4,23E-01	3,03E-03
d	3,14E+17	1,06E-13	9,80E-03	9,37E+00	5,70E-01	1,70E-02
e	1,48E+07	1,30E-13	1,73E-02	1,27E+01	1,66E+00	6,68E-03
f	1,91E+05	1,84E-01	1,85E-01	1,50E+01	5,94E-01	6,76E-03
g	2,00E+16	2,57E-13	2,84E-01	6,93E+02	1,26E+01	7,03E-03
h	1,47E+05	7,09E-13	3,37E-01	2,46E+02	5,23E+00	1,38E-02
i	1,45E+05	1,42E-02	2,25E-02	1,44E+00	2,14E+00	1,99E-03

Table 1: results of the first set of computations, comparing the results of the regularized and non-regularized solutions.

As an example, the results of the case d) are shown below. It can be seen how the displacements change sensibly from the non-regularized solution to the regularized one in Figure 9 and 10, in a similar fashion as tractions in Figure 11 and 12. By the other hand, the L-curve from this case is shown in figure 13, where the selected value of λ is marked with a black dot.

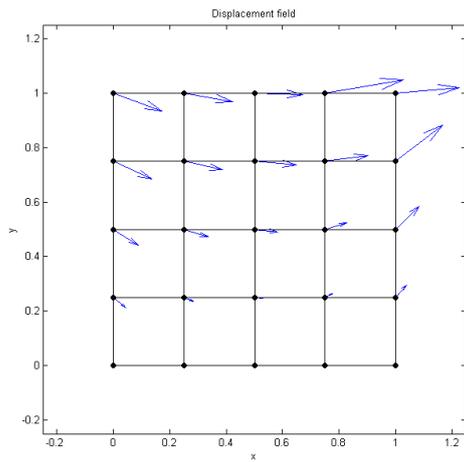


Figure 9: displacements of case d) without regularization.

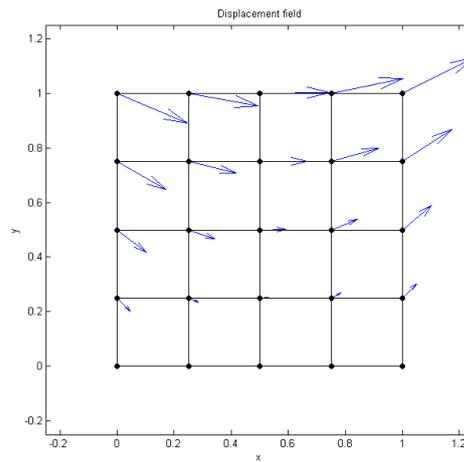


Figure 10: displacements of case d) with regularization.

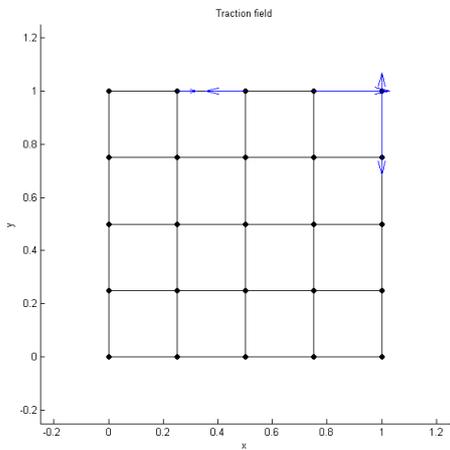


Figure 11: tractions of case d) without regularization.

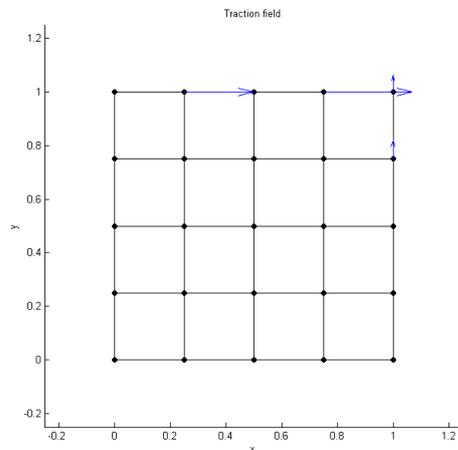


Figure 12: tractions of case d) with regularization.

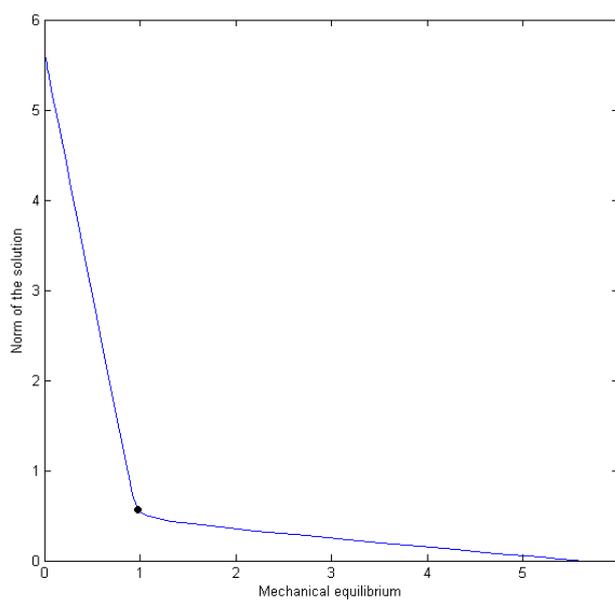


Figure 13: L-curve of case d). The optimal selected value of λ is marked with the black dot.

4.2. Set 2

The results of the first set of numerical computations are shown in table 2, seen below:

Set 2						
Case	κ	Mec. Equilibrium	Regularized ME	Solution norm	Regularized SN	λ
a	2,39E+05	1,54E-02	1,54E-02	2,43E-01	2,30E-01	2,16E-02
b	3,67E+18	1,22E-13	2,05E-03	8,80E+01	5,62E-01	1,78E-02
c	2,75E+05	1,75E-13	4,19E-03	9,23E-01	6,34E-01	3,91E-02
d	4,68E+18	9,57E-11	5,63E-03	7,90E+02	5,00E-01	2,80E-02
e	1,00E+16	1,57E-06	1,04E-02	2,08E+03	4,23E-01	3,52E-02
f	3,86E+11	2,00E-02	5,83E-02	2,10E+05	4,32E-01	4,17E-02
g	4,95E+16	4,05E-12	9,54E-02	5,43E+03	3,37E+00	2,29E-02
h	6,73E+06	2,60E-12	1,08E-01	9,14E+02	3,89E-01	1,31E-01
i	2,45E+06	1,21E-02	1,50E-02	3,04E+00	5,82E-01	1,75E-02

Table 2: results of the second set of computations, comparing the results of the regularized and non-regularized solutions.

The results of case d), as before, show the displacement field in Figure 14 and 15, the traction field in Figure 16 and 17, and finally the L-curve in Figure 18, with the optimal λ value marked with a black dot.

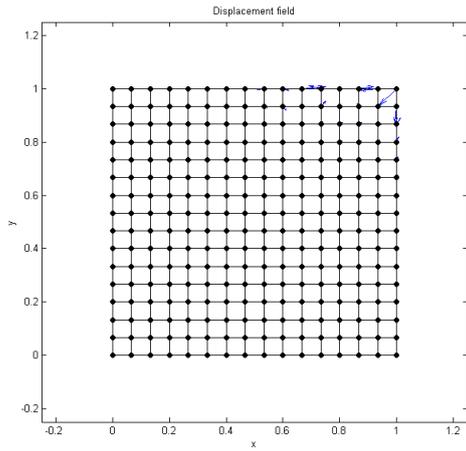


Figure 14: displacements of case d) without regularization.

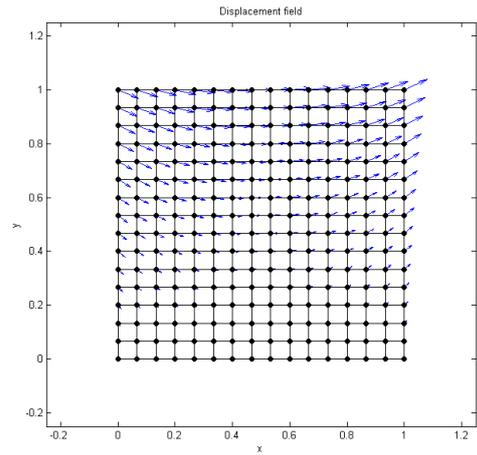


Figure 15: displacements of case d) with regularization.

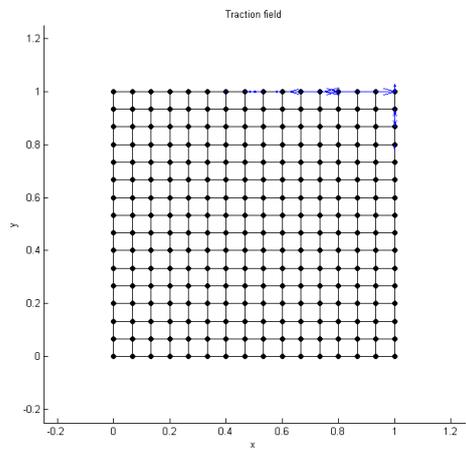


Figure 16: traction field of case d) without regularization.

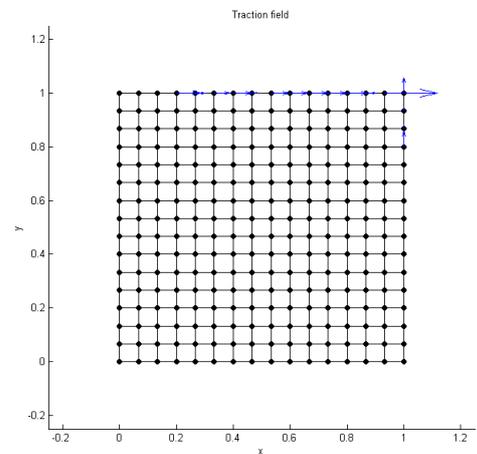


Figure 17: traction field of case d) with regularization.

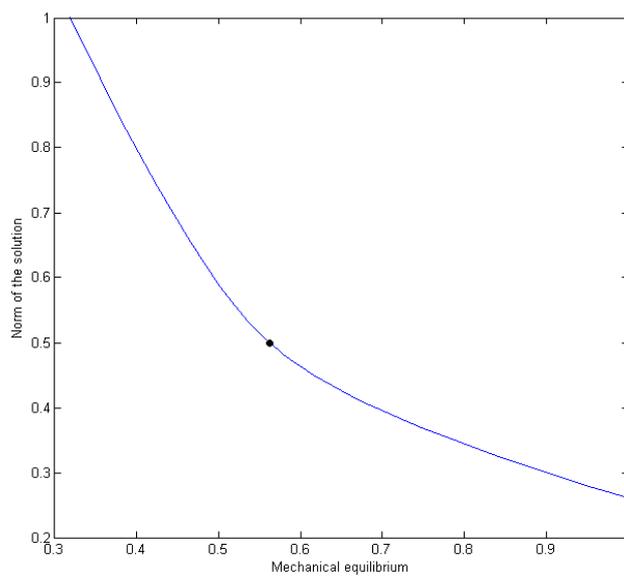


Figure 18: L-curve of case d). The optimal selected value of λ is marked with the black dot.



In this project, the optimal value of λ has been determined through the norm between the two quantities of interest and using a suitable scaling factor to make them comparable. Optimization algorithms are envisaged for future work in order to find λ in a more robust way. A stability study of the equations is also set as future work, although preliminary conclusions can be conjectured from the two sets of numerical computations.

5. Conclusions

The present work has verified some previous theoretical results [1], that is:

- Regularisation is not required when $m \leq n_0$ and $\Omega_T \subseteq \Omega_{u_0}$.
- The least-square inverse problem is ill-posed when $m > n_0$. Regularisation is required in these cases.
- The condition of the problem increases proportionally to the mesh size.
- The Tikhonov regularization method is a suitable regularization method for the least-square inverse problem [2].

Additionally, it has been shown that regularisation may be applied also when $\Omega_T \not\subseteq \Omega_{u_0}$ nor $\Omega_{u_0} \not\subseteq \Omega_T$. It has been shown that for different values of lambda, plotting $\|\mathbf{T}\|$ versus $\|\mathbf{Ku} - \mathbf{AT}\|$ follows the so-called L-curve only when the non-regularized problem is ill-conditioned, while this qualitative shape is lost when the system is well posed.

Regarding the results shown in the previous chapter, and relying on the verified previous theoretical results, it is straightforward to see that, when regularization is required, the optimal value of λ increases when the mesh becomes finer. Although the mechanical equilibrium seems to degrade following the data shown in Table a and 2 when applying regularization, the regularized solution shows a much more coherent solution. This is due to the non-regularized solution is obtained through the computation of \mathbf{T} , which represents a solution that belongs to the null space of the system matrix.

It has been left for future work the implementation of the problem in a 3-dimensional domain, an optimization algorithm to determine the optimal value of λ in a more precise and robust way, relying to the shape of the L-curve and an attempt to achieve an analytical solution of the experimental results obtained during this project.

6. Bibliography

[1] – J. Muñoz. "Non-regularised Inverse Finite Element Analysis for 3D Traction Force Microscopy". *International Journal of Numerical Analysis and Modelling*. 13 (5): 763-781, 2016.

[2] – P. C. Hansen. "The L-curve and its use in the numerical treatment of inverse problems". Department of Mathematical Modelling, Technical University of Denmark.

[3] – R. W. Style, R. Boltyanskiy, G. K. German, C. Hyland, C. W. MacMinn, A. F. Mertz, L. A. Wilen, Y. Xu, E. R. Dufrense. "Traction Force Microscopy in physics and biology". 10.1039/C4SM00264D, 2014.

[4] – Mase, G. Thomas; Mase, George E. *Continuum mechanics for engineers*. 3rd ed. Boca Raton: CRC, 2010.

Annexes

A. Code

A.1. main.m

```
% MECHANICAL INVERSE ANALYSIS
%
% ux = Matrix with x displacements.
%     ux(i,j)=displacement at node in
%         vertical position i (y, botttom to up)
%         horizontal position j (x, left to right)
% uy = Matrix with y displacements
% uxM = matrix indicating known x displacements
%     uxM(i,j)=1 => displacement ux at node i,j is known (u0)
%     uxM(i,j)=0 => displacement ux at node i,j is unknown (u1)
% uyM = matrix indicating known y displacements
% TxM = matrix indicating where Tx must be computed. Same structure as
uxM
% TyM = matrix indicating where Tx must be computed. Same structure as
uxM
format short e
clear; close all; clc
% P = dinput('Plot results? y/[]','n');

Mesh.nx=4; Mesh.ny=4; Mesh.nz = 4;
Mesh.Lx=1; Mesh.Ly=1; Mesh.Lz = 1;
Mesh.dim=2;

Mat.E=1;
Mat.v=0.3;

if Mesh.dim == 2
    [C,X] = GenerateMesh (Mesh);
else
    [~,C,X] = GenerateMesh_Jose (Mesh);
end

[BCu,BCT,u0]=DefineBCRectangle (Mesh.nx, Mesh.ny);

% Experimental displacements
% u(1,:) node numbers
% u(2,i) dof of node (1,i)
% u(3,i) value of u at node (1,i) and dof (2,i)
% nums = size(BCu,1);
% u0 = [BCu ones (nums,1)];% 2*rand(nums,1)-1
[u,Tout]=Inverse (Mesh,Mat,u0,BCu,BCT,C,X); % t=nodal forces=Reactions

% if strcmpi(P,'y') == 1
%     Plotter
% end
% close all
```

A.2. AssembleA.m

```
function A = AssembleA(K,dofT)
%AssembleA Assembles matrix A through matrix K and dofT
% Detailed explanation goes here
n=length(dofT);
A = zeros(size(K,1),n);
for i=1:n
    A(dofT(i),i)=1;
end
end
```

A.3. bisection.m

```

function y = bisection(K0,K1,u0,M,A)
%BISECTION Finds the optimal value for lambda to apply in the l-curve
%regularization method for a particular case.
% The method used is a combination of logarithmic loop and a
bisection
% method to minimize the distance from the L-curve to
% [0,0]

tol = 1e-6; maxIts = 1e3;
xys = 100; % Number of y initial values
% ylist = ones(xys,1);
% ylist(1) = 1e-6;
% for i = 2:xys
%     ylist(i) = ylist(i-1)*10;
% end
ylist = (logspace(-10,5,xys))';
ylist = (logspace(-15,10,xys))'; %Nou, test
multi = 1e2;
% ylist = ([1e-3:1e-3:1e-2 2e-2:1e-2:5e-2])';
% xys = length(ylist);

% Logarithmic loop
Ts = zeros(xys,1);
KUATs = Ts;
dists = Ts;
for i = 1:xys
    Id=inv(A'*A+ylist(i)*eye(size(A,2)));
    I0 = eye(size(A,1))-A*Id*A';
    u1 = (K1'*I0*K1)\(-K1'*I0*K0*u0);
    T = M\ (Id*A'*(K0*u0+K1*u1));
    Ts(i) = norm( T );
    KUATs(i) = norm( K0*u0+K1*u1-A*M*T );
    dists(i) = norm([Ts(i) multi*KUATs(i)]);
end
resu = [ylist Ts KUATs dists];

% Bisection method
[~,pos] = min(dists);

if (pos>1 && pos<xys)
    y1 = ylist(pos-1);
    y3 = ylist(pos+1);
    d1 = dists(pos-1);
    d3 = dists(pos+1);
elseif pos == xys
    y1 = ylist(pos-1);
    y3 = ylist(pos);
    d1 = dists(pos-1);
    d3 = dists(pos);
elseif pos == 1
    y1 = ylist(1);
    y3 = ylist(2);
    d1 = dists(1);
    d3 = dists(2);
end
y2 = (y1+y3)/2;

```

```

it = 1;
Ts = zeros(maxIts,1);
KUATs = Ts;
dists = Ts;
ylist = Ts;

while it < maxIts
    Id=inv(A'*A+y2*eye(size(A,2)));
    I0 = eye(size(A,1))-A*Id*A';
    u1 = (K1'*I0*K1)\(-K1'*I0*K0*u0);
    T = M\ (Id*A'*(K0*u0+K1*u1));
    Ts(it) = norm( T ); % With y2
    KUATs(it) = norm( K0*u0+K1*u1-A*M*T );
    d2 = norm([Ts(it) multi*KUATs(it)]);    dists(it) = d2;
    ylist(it) = y2;
    [~,o] = min([d1 d2 d3]);
    if o == 1
        y3 = y2;
        d3 = d2;
    elseif o == 3
        y1 = y2;
        d1 = d2;
    elseif o == 2
        [~,o2] = min([d1 d3]);
        if o2 == 1
            y3 = y2;
            d3 = d2;
        elseif o2 == 2
            y1 = y2;
            d1 = d2;
        end
    end
    end
    if abs((y1+y3)/2-y2) < 1e-6
        y = y2;
        display(y)
        break
    end
    y2 = (y1+y3)/2;
    it = it+1;
end
if it<maxIts
    display('Optimal regularization parameter found')
else
    y = y2;
    display('Optimal regularization parameter NOT found')
end
display('100·Distance =')
display(d2)

resu2 = [ylist Ts KUATs dists];
resu2( ~any(resu2,2), : ) = []; %Remove empty rows
resu = [resu;resu2]; clear resu2;
[~,ord] = sort(resu(:,3));
resu(:,1) = resu(ord,1);
resu(:,2) = resu(ord,2);
resu(:,3) = resu(ord,3);
resu(:,4) = resu(ord,4);

figure;
plot(multi*resu(:,3),resu(:,2)); hold on
if it<maxIts

```

```

scatter(multi*KUATs(it),Ts(it),33,'k','fill'); hold on
else
scatter(multi*KUATs(end-1),Ts(end-1),33,'k','fill'); hold on
end
% title(strcat(['L-curve with lambda = ',num2str(y2)]));
% axis([5.8 6.1 0 1.2]);
xlabel('Mechanical equilibrium'); ylabel('Norm of the solution');
hold off

% figure;
% loglog(resu(:,1),resu(:,2),'k'); hold on
% loglog(resu(:,1),resu(:,3)); hold on
% xlabel('lambda');
% legend('||| T |||','||| Ku - At |||');
% hold off

end

```

A.4. ConvertDof.m

```

function [ out,value ] = ConvertDof( in,dim,dof )
%ConvertDof converts a way of expressing degrees of freedom to another
way
% If the input is a list of dof, the function translates it into a
table
% where:
% - Column 1 corresponds to n° of node
% - Column 2 corresponds to n° of dof (x=1, y=2, z=3)
% If the input is a table, the function translates it into a list
where
% each value is a corresponding dof between 1 and dim*nnodes
% NOTE: The function assumes the numeration of the nodes can be used
to
% find the dof number.

if (size(in,2) == 1 && ~exist('dof','var')) % Convert from vector to
table
    list = in;
    list = list/dim;
    table = zeros(length(list),2);
    table(:,1) = round(list);
    if dim == 2
        a = find(rem(list,1)~=0);
        b = setdiff(1:length(list),a);
        table(a,2) = 1;
        table(b,2) = 2;
    elseif dim == 3
        a = find(rem(list,1) < 0.5 && rem(list,1) > 0);
        b = find(rem(list,1) > 0.5);
        c = setdiff(1:length(list),unique(a,b));
        table(a,2) = 1;
        table(b,2) = 2;
        table(c,2) = 3;
    end
    out = table;

elseif size(in,2) == 1
    out = [ floor((dof-1)/dim)+1 rem(dof-1,dim)+1 in ];

elseif size(in,2) > 1 % Convert from table to vector
    table = in;
    list = dim*(table(:,1)-1)+table(:,2);
    [list,inds] = sort(list);
    out = list;
    if size(in,2) == 3
        value = table(:,end);
        value = value(inds);
    end

end

end

end

```

A.5. DefineBCRectangle.m

```
function [ BCu,BCt,u0] = DefineBCRectangle( nx,ny)
%DefineBCRectangle Summary of this function goes here
% BCu: Dirichlthe constraints
% BCt: dof with applied tractions
% u0 : dof with measured displacements
% Detailed explanation goes here
%     Ly=max(X(:,2));
%     dim=size(X,2);
%     nodeD = find(abs(X(:,2))<eps);
%     nodeT = find(abs(X(:,2)-Ly)<eps);
%     dofD = (nodeD-1)*dim+1; % constrained dof along x due to
Dirichlet
%     dofT = (nodeT-1)*dim+1; % dof with applied load along x
%     if dim == 3
%         dofD = [dofD (nodeD-1)*dim+2];
%         dofT = [dofT (nodeT-1)*dim+2];
%     end
%     dofD = [dofD' nodeD'*dim ]'; % constrained dof due to
Dirichlet
%     % dofT = [dofT' nodeT'*dim ]'; % dof with applied load
%
%     BCu = ConvertDof(dofD,dim);
%     BCt = ConvertDof(dofT,dim);

t0 = 'Which case do you want to study?\n';
t1 = '1: T as a subset of u0\n';
t2 = '2: u0 as a subset of T\n';
t21 = '21: u0 has 75% of dof w.r.t. case 2\n';
t3 = '3: dom(T) == dom(u0)\n';
t4 = '4: T and u0 share dof and dom(T) > dom(u0)\n';
t41 = '41: u0 has 1/3 of independent dof w.r.t. case 4\n';
t5 = '5: T and u0 share dof and dom(T) = dom(u0)\n';
t6 = '6: T and u0 share dof and dom(T) < dom(u0)\n';
t7 = '7: T and u0 empty intersect and dom(T) > dom(u0)\n';
t8 = '8: T and u0 empty intersect and dom(T) = dom(u0)\n';
t9 = '9: T and u0 empty intersect and dom(T) < dom(u0)\n';

text = strcat(t0,t1,t2,t21,t3,t4,t41,t5,t6,t7,t8,t9);
example = input(text);

pX = nx+1;
pY = ny+1;
switch example
case 1
    BCt = [(1:round(pX/2))' ones(round(pX/2),1)];
    u0 = [(1:round(3*pX/4))' ones(round(3*pX/4),1)];
case 2
    BCt = [(1:round(3*pX/4))' ones(round(3*pX/4),1)];
    u0 = [(1:round(pX/2))' ones(round(pX/2),1)];
case 21 % Amb 6 dof en comptes de 8
    BCt = [(1:round(3*pX/4))' ones(round(3*pX/4),1)];
    u0 = [ 1 1
           2 1
           3 1
           6 1
           7 1
           8 1 ];
```

```

case 3
    BCt = [(1:round(3*pX/4))' ones(round(3*pX/4),1)];
    u0 = BCt;
case 4
    BCt = [(round(pX/4):pX-1)' ones(pX-round(pX/4),1)];
    num = mod(round(pX*pY/4),10);
    % if num~=0
    %     BCt = [BCt ; (pX:pX:pY*num/4)'
2*ones(ceil(pY/(num*4)),1)];
    % else
    BCt = [BCt ; (pX:pX:round(pX*pY/4))'
2*ones(round(pY/4),1)];
    % end
    if BCt(1,1) == 1
        BCt = BCt(2:end,:);
    end
    i=BCt(:,2)==2;
    if sum(i) < 2
        BCt = [BCt; 2*pX 2];
    end
    u0 = [(1:round(pX/2))' ones(round(pX/2),1)];
case 41 % Amb només un dof diferent en comptes de 3
    BCt = [(round(pX/4):pX-1)' ones(pX-round(pX/4),1)];
    num = mod(round(pX*pY/4),10);
    if num~=0
        BCt = [BCt ; (pX:pX:pY*num/4)' 2*ones(ceil(pY/(num*4)),1)];
    else
        BCt = [BCt ; (pX:pX:round(pX*pY/4))'
2*ones(round(pY/4),1)];
    end
    if BCt(1,1) == 1
        BCt = BCt(2:end,:);
    end
    [~,i]=find(BCt(:,2)==2);
    if length(i) < 2
        BCt = [BCt; 2*pX 2];
    end
    u0 = [ 3 1
          4 1
          5 1
          6 1
          7 1
          8 1
          9 1
          10 1];
case 5
    BCt = [(round(pX/3)+1:pX)' ones(pX-round(pX/3),1)];
    u0 = [(1:round(2*pX/3))' ones(round(2*pX/3),1)];
case 6
    BCt = [(1:round(pX/2))' ones(round(pX/2),1)];
    u0 = [(round(pX/4):pX-1)' ones(pX-round(pX/4),1)];
    num = mod(round(pX*pY/4),10);
    % if num~=0
    %     u0 = [u0 ; (pX:pX:pY*num/4)' 2*ones(ceil(pY/(num*4)),1)];
    % else
    u0 = [u0 ; (pX:pX:round(pX*pY/4))' 2*ones(round(pY/4),1)];
    % end
    if u0(1,1) == 1
        u0 = u0(2:end,:);
    end
    i=u0(:,2)==2;

```

```

    if sum(i) < 2
        u0 = [u0; 2*pX 2];
    end
    case 7
        BCt = [(1:round(3*pX/4))' ones(round(3*pX/4),1)];
        u0 = [(1:round(pX/2))' 2*ones(round(pX/2),1)];
    case 8
        BCt = [(1:round(pX/2))' ones(round(pX/2),1)];
        u0 = [(1:round(pX/2))' 2*ones(round(pX/2),1)];
    case 9
        BCt = [(1:round(pX/2))' 2*ones(round(pX/2),1)];
        u0 = [(1:round(3*pX/4))' ones(round(3*pX/4),1)];
    otherwise
        error('Case not implemented');
    end

end

num = input('Prescribed displacements value (default = 1):\n');
if isempty(num) == 1
    num = 1;
end

BCu = ((nx+1)*ny+1:(nx+1)*(ny+1))';
BCu = [BCu ones(size(BCu,1),1)
       BCu 2*ones(size(BCu,1),1)];
u0 = [u0 num*ones(size(u0,1),1)]; % Applied unit displacement
display(BCt)
display(u0)

end

```

A.6. GenerateMesh.m

```
function [Bij,C,X]=GenerateMesh(Mesh)
% USAGE:
% [X,C]=GenerateMesh(dx,dy,nx,ny,dz,nz)
% INPUT:
% nx = number of elements along x (Vertical)
% dx = size of each element along x
% ny = number of elements along y (Horizontal)
% dx = size of each element along y
% nz = number of elements along z
% dz = size of each element along z (if dz==0, 2D)
% OUTPUT:
% X(:,i) =nodal coordiantes of node i
% C(:,e) =nodal connectivities of element e
% Bij = matrix with position of nodes at each grid point
%      2D: Bij(1,1) node at top left,
%          Bij(1,ny+1) node at top right,
%          Bij(nx+1,ny+1) node at bottom right
%          +---> ny
%          |
%          nx |
%          v
%      3D: Bij(1,1,1) node at bottom left on z=0,
%          Bij(ny+1,nx+1,nz+1) node at top right on z=nz*dx
dx = Mesh.Lx/Mesh.nx;
dy = Mesh.Ly/Mesh.ny;
dz = Mesh.Lz/Mesh.nz;
nx = Mesh.nx;
ny = Mesh.ny;
nz = Mesh.nz;

if abs(dz)<eps % 2D
    Bij=zeros(nx+1,ny+1);
    X=zeros((nx+1)*(ny+1),2);
    C=zeros(nx*ny,4);
    for j=1:ny+1 % horizontal (left to right)
        for i=1:nx+1 % Vertical (bottom to top)
            nn=(i-1)*(ny+1)+j;
            X(nn,:)=[(j-1)*dx,(i-1)*dy];
            Bij(nx+2-i,j)=nn;
            if i~=nx+1 && j~=ny+1
                ne=(nx-i)*ny+j;
                nbl=nn;
                bot=[nbl,nbl+1,nbl+ny+2,nbl+ny+1];
                C(ne,:)=bot;
            end
        end
    end
else
    Bij=zeros(nx+1,ny+1,nz+1);
    X=zeros((nx+1)*(ny+1)*(nz+1),3);
    C=zeros(nx*ny*nz,8);
    for k=1:nz+1
        for j=1:ny+1 % horizontal (left to right)
            for i=1:nx+1 % Vertical (bottom to top)
                nn=(k-1)*(nx+1)*(ny+1)+(i-1)*(ny+1)+j;
                X(nn,:)=[(j-1)*dx,(i-1)*dy,(k-1)*dz];
                Bij(nx+2-i,j,k)=nn;
                if k~=nz+1 && i~=nx+1 && j~=ny+1
```


A.7. Inverse.m

```

function [u,T]=Inverse (Mesh,Mat,u0,BCu,BCT,C,X)
% Mesh.nx;
% Mesh.ny;
% Mesh.Lx;
% Mesh.Ly;
% Mat.E;
% Mat.v;
% INPUT:
% ux = Matrix with displacements.
%     u(i,1) = nodes with read displacement
%     u(i,2) = dof at node (i) with read displacement
%     u(i,3) = value of the read displacement
% dofD = Dirichlet dof (prescribed with value u=0)
% dpfD0 = Read displacements (prescribed with value u<>0)
% dpfD1 = Unknown displacements (not prescribed)
nx = Mesh.nx;
ny = Mesh.ny;
Lx = Mesh.Lx;
Ly = Mesh.Ly;
E = Mat.E;
v = Mat.v;
dim = Mesh.dim;
% Build elemental matrices Ke, Ae and Me
% Build Xe
Xe = [ 0      0
      Lx/nx   0
      Lx/nx   Ly/ny
      0      Ly/ny ];
% Build Ge
Ge = [ E  v  2 ];

% Create Ke and Me
if dim==2
    Ke = keq4e(Xe,Ge);
    Me = Me2le(Xe);
elseif dim == 3
    Ke = keh8e(Xe,Ge);
    Me = Meq4e(Xe,Ge);
end
% Connectivity matrix
% Create matrix X and C

% Assemble matrices Ke,Ae and Me into K, A and M
K = Assemble(C,Ke,Mesh);
% A=nnod*dim x dof(t)
%
% Boundary conditions for u and loading t
dofD=ConvertDof(BCu,dim);
dofT=ConvertDof(BCT,dim);
[dofD0,u0] = ConvertDof(u0,dim);

A = AssembleA(K,dofT); % \hat A(i,j)=1 or 0
M = MassMatrix(C,dofT,Xe,Mesh);
% Extract matrices K0 and K1 and assign values of u0 from ux and uy

nnod = size(X,1);
dofD1 = Mask(dofD,dofD0,dim*nnod); ...

```

```

% Convert Mask on grid to dof array. Check no duplicated dofs

% K0:
dofDF = [dofD0 ; dofD1]; % Free dof, measured u0 and unknown u1
K0 = K(dofDF,dofD0);
% K1 =
K1 = K(dofDF,dofD1);
% A =
A = A(dofDF, :);
% Solve system (eq. (23)). Assume TxM=uxM, TyM=uyM
[u1,T] = solver(K0,K1,u0,M,A);

% Comprovació directe-invers
rhs = K0*u0+K1*u1;
lhs = A*M*T;
display('Mechanical equilibrium norm =')
disp(norm(rhs-lhs))
display('Norm of the solution =')
disp(norm(T))

u0 = ConvertDof( u0,dim,dofD0 );
u1 = ConvertDof( u1,dim,dofD1 );
uDBC = ConvertDof(zeros(length(dofD),1),dim,dofD);
u = [u0;u1;uDBC];
u = sortrows(u,[1 2]);
T = ConvertDof(T,dim,dofT);

```

A.8. keh8e.m

```
function Ke = keh8e(Xe,Ge)
%*****
% Keh8E:
%   Creates the element stiffness matrix of elastic
%   hexahedra 8-node element.
% Syntax:
%   Ke = keh4e(Xe,Ge)
% Input:
%   Xe   : coordinates Xe = [x1 y1; x2 y2; x3 y3; x4 y4]
%   Ge   : element material data: [E , nu , (type)]
% Output:
%   Ke   : element stiffness matrix.
% Date:
%   Version 1.0   04.05.2014
%*****

% Gauss abscissae and weights.
r = [-1 1]/sqrt(3);
w = [ 1 1];

% Set isotropic elasticity matrix
E = Ge(1);
nu = Ge(2);
f=E/(1-2*nu)/(1+nu);

D(1:3,1:3) = f* ...
    [ 1-nu  nu    nu
      nu   1-nu  nu
      nu    nu   1-nu];
D(4,4)=f*(1-2*nu)/2;
D(5,5)=D(4,4);
D(6,6)=D(5,5);

% determine number of nodes per element
nnodes = size(Xe,1);

% Initialize stiffness matrix.
Ke = zeros(3*nnodes);

% Gauss integration of stiffness matrix.
for i = 1:2
    for j = 1:2
        for k = 1:2

            % Parametric derivatives:
            dN(1:3,1:4) = [ -(1-r(j))  (1-r(j))  (1+r(j))  -(1+r(j))
                          -(1-r(i))  -(1+r(i))  (1+r(i))  (1-r(i))
                          (1-r(i))*(1-r(j))  (1+r(i))*(1-r(j))  (1+r(i))*(1+r(j))
                          (1-r(i))*(1+r(j))] / 8;
            dN(3,5:8) = dN(3,1:4);
            dN(3,1:4) = -dN(3,1:4);
            dN(1:2,5:8) = dN(1:2,1:4)*(1+r(k));
            dN(1:2,1:4) = dN(1:2,1:4)*(1-r(k));
            % transform to global coordinates
            Jt = dN*Xe;
            dN = Jt\dN;
        end
    end
end
```

```

% set up 4 node part of the gradient matrix
B=zeros(6,3*nnodes);
B(1,1:3:24) =dN(1,:);
B(2,2:3:24) =dN(2,:);
B(3,3:3:24) =dN(3,:);
B(4,1:3:24) =dN(2,:);
B(4,2:3:24) =dN(1,:);
B(5,1:3:24) =dN(3,:);
B(5,3:3:24) =dN(1,:);
B(6,2:3:24) =dN(3,:);
B(6,3:3:24) =dN(2,:);

Ke = Ke + w(i)*w(j)*w(k)*( B'*D*B )*det(Jt);

    end
  end
end

```

A.9. ke4e.m

```

function Ke = keq4e(Xe,Ge)
%*****
% KeQ4E:
%   Creates the element stiffness matrix of elastic
%   quadrilateral 4-node element in plane stress or
%   plane strain.
%   If Xe contains 8 node coordinate set the stiff-
%   ness matrix of 8-node element with quadratic
%   shape functions is evaluated using reduced
%   integration.
% Syntax:
%   Ke = keq4e(Xe,Ge)
% Input:
%   Xe   : coordinates Xe = [x1 y1; x2 y2; x3 y3; x4 y4]
%   Ge   : element material data: [E , nu , (type)]
%          optional: type = 1 : plane stress (default)
%                  type = 2 : plane strain
% Output:
%   Ke   : element stiffness matrix.
% Date:
%   Version 1.0   04.05.95
%*****

% Gauss abscissae and weights.
r = [-1 1]/sqrt(3);
w = [ 1 1];

% Set isotropic elasticity matrix
E = Ge(1);
nu = Ge(2);

if size(Ge,2) == 2           % default: plane stress
    type = 1;
else
    type = Ge(3);
end

if type == 1                % plane stress
    D = E/(1-nu^2) ...
        * [ 1 nu 0
            nu 1 0
            0 0 (1-nu)/2 ];
else                         % plane strain
    D = E/((1+nu)*(1-2*nu)) ...
        * [ 1-nu nu 0
            nu 1-nu 0
            0 0 (1-2*nu)/2 ];
end

% determine number of nodes per element
nnodes = size(Xe,1);

% Initialize stiffness matrix.
Ke = zeros(2*nnodes);

% Gauss integration of stiffness matrix.
for i = 1:2

```

```

for j = 1:2

% Parametric derivatives:
dN = [ -(1-r(j)) (1-r(j)) (1+r(j)) -(1+r(j))
        -(1-r(i)) -(1+r(i)) (1+r(i)) (1-r(i)) ]/4;

if nnodes==8
% evaluate the quadratic terms for the midside nodes
dN8 = ...
[ -r(i)*(1-r(j)) 0.5*(1-r(j)^2) -r(i)*(1+r(j)) -0.5*(1-r(j)^2)
  -0.5*(1-r(i)^2) -r(j)*(1+r(i)) 0.5*(1-r(i)^2) -r(j)*(1-r(i))
];

% modify corner nodes
dN(:,1) = dN(:,1) - 0.5*dN8(:,1) - 0.5*dN8(:,4);
dN(:,2) = dN(:,2) - 0.5*dN8(:,2) - 0.5*dN8(:,1);
dN(:,3) = dN(:,3) - 0.5*dN8(:,3) - 0.5*dN8(:,2);
dN(:,4) = dN(:,4) - 0.5*dN8(:,4) - 0.5*dN8(:,3);

% expand gradient matrix
dN = [dN, dN8];
end

% transform to global coordinates
Jt = dN*Xe;
dN = Jt\dN;

% set up 4 node part of the gradient matrix
B = [ dN(1,1) 0 dN(1,2) 0 dN(1,3) 0 dN(1,4) 0
      0 dN(2,1) 0 dN(2,2) 0 dN(2,3) 0 dN(2,4)
      dN(2,1) dN(1,1) dN(2,2) dN(1,2) dN(2,3) dN(1,3) dN(2,4) dN(1,4)
];

if nnodes == 8
% set up gradient matrix for midside nodes
B8 = [ dN(1,5) 0 dN(1,6) 0 dN(1,7) 0 dN(1,8)
       0 dN(2,5) 0 dN(2,6) 0 dN(2,7) 0
       dN(2,5) dN(1,5) dN(2,6) dN(1,6) dN(2,7) dN(1,7) dN(2,8)
       dN(1,8) ];

% expand gradient matrix
B = [B,B8];
end

Ke = Ke + w(i)*w(j)*( B'*D*B )*det(Jt);

end
end

```

A.10. Mask.m

```
function [ dofD1 ] = Mask(dofD,dofD0,ndof)
% Convert Mask on grid to dof array. Check no duplicated dofs
% and assign values to u0 from (ux,uy,uz)
% dofD = Dirichlet dof (prescribed with value u=0)
% dofD0 = Read displacements (prescribed with value u<>0), according
to
%      Mask
% dofD1 = Unknown displacements (not prescribed)
% u0     = Matrix with displacement values u(i,3) on nodes u(i,1) and
dof
% (u(i,2)
% nodes numbered as (ATM)
%
% 1---2---3
% |   |   |
% 4---5---6
% | ....
% and dof numbered as
% (1,2[,3]) dof of node 1
% (3,4)   dof of node 2 (4,5,6 in 3D)
% ....
% COMMENTS:
% dofD0 : contains dof with 1 in Mask, and not in list dofD
(Dirichlet)
% dofD1 : contains dof with 0 in Mask, and not in list dofD
(Dirichlet)

% dofD1
dofD1 = 1:ndof;
dofD1 = (setdiff(dofD1,[dofD;dofD0]))';
```

A.11. MassMatrix.m

```

function M = MassMatrix(C,dofT,Xe,Mesh)
%Build elemental Mass matrices and Assembles global M
nnod=(Mesh.ny+1)*(Mesh.nx+1);
nele=size(C,1);
dim=Mesh.dim;
if dim==3
    nnod=nnod*(Mesh.ny+1);
end

dofTi = zeros(1,dim*nnod);
dofTi(dofT) = 1;
% Build dofTu vector with correspondance u dof <-> t dof
% dofTu(i)=traction dof for displacement dof=i.
r=zeros(nnod*dim*nele,1);
c=r;
v=c;
k=0;
% K = zeros( dim*nnod , dim*nnod );
% Create matrix K
if dim == 2
    localEdges = [ 1 2
                  2 3
                  3 4
                  4 1 ];
elseif dim == 3
    localEdges = [ 1 4 3 2
                  1 2 6 5
                  5 6 7 8
                  8 7 3 4
                  4 1 5 8
                  2 3 6 7 ];
end

for e = 1:nele
    n = C(e,:);
    for i = 1:size(localEdges,1) % Loop on element edges
        % Check if element has 2 nodes with dof in dofT
        node = n(localEdges(i,:));
        % (max(dofT==nLoop(i))~=0 && max(dofT==nLoop(i+1))~=0)
        if min( dofTi((node-1)*dim+1) ) > 0 ||...
            min( dofTi((node-1)*dim+2) ) > 0 ...
        % Assumed all tractions on x also on y and z
        % Xe = %2 Nodes at boundary with traction
        if dim == 2
            Me = Me2le(Xe);
        elseif dim == 3
            Me = Meq4e(Xe,Tz);
        end%MassMatrix(Xe);
        % Assemble Me
        indI = 0;
        for nod = 1:length(node)
            ni = node(nod);
            indJ = 0;
            indI = indI+1;
            for j = 1:length(node)
                nj = node(j);

```

```

indJ = indJ+1;
for di=1:dim
    for dj=1:dim
        k=k+1;
        r(k)=dim*(ni-1)+di;
        c(k)=dim*(nj-1)+dj;
        v(k)=Me(dim*(indI-1)+di,dim*(indJ-1)+dj);
    end
end
end
end
end
end
end
end

M = sparse(r(1:k),c(1:k),v(1:k),dim*nnod,dim*nnod);

M = M(dofT,dofT);

end

```

A.12. Me2le.m

```

function Me = Me2le(Xe)
%*****
% AeQ4E:
%   Creates the element mass matrix of elastic
%   linear 2-node element
% Syntax:
%   Me = Me2le(Xe,Ge)
% Input:
%   Xe   : coordinates Xe = [x1 y1; x2 y2]
%   Tz   : =0, z tractions assumed zero, Tz=0; if >0, Tz included.
% Output:
%   Ae   : element traction matrix.
% Date:
%   Version 1.0   04.05.13
%*****

% Gauss abscissae and weights.
r = [-1 1]/sqrt(3);
w = [ 1 1];

% determine number of nodes per element
dimT=2;
% Initialize stiffness matrix.
M=zeros(2,2);
I=eye(dimT);
% Gauss integration of stiffness matrix.
for i = 1:2
    % Function values
    N= [ (1-r(i))
        (1+r(i)) ]/2;
    % Parametric derivatives:
    dN = [ -1  1 ]/2;
    % transform to global coordinates
    Jt = dN*(Xe(:,1:2))'; % dx/dxi
    M = M + w(i)*(N*N')*norm(Jt);
end
Me=zeros(dimT*2,dimT*2);
for i=1:2
    for j=1:2
        Me((i-1)*dimT+1:i*dimT,(j-1)*dimT+1:j*dimT)=M(i,j)*I;
    end
end
end

```

A.13. Meq4e.m

```

function Me = Meq4e(Xe,Tz)
%*****
% AeQ4E:
%   Creates the element mass matrix of elastic
%   quadrilateral 4-node element such that
% Syntax:
%   Me = Meq4e(Xe,Ge)
% Input:
%   Xe   : coordinates Xe = [x1 y1; x2 y2; x3 y3; x4 y4]
%   Tz   : =0, z tractions assumed zero, Tz=0; if >0, Tz included.
% Output:
%   Ae   : element traction matrix.
% Date:
%   Version 1.0   04.05.13
%*****

% Gauss abscissae and weights.
r = [-1 1]/sqrt(3);
w = [ 1 1];

% determine number of nodes per element
dimT=2;
if Tz
    dimT=3;
end
% Initialize stiffness matrix.
M=zeros(4,4);
I=eye(dimT);
% Gauss integration of stiffness matrix.
for i = 1:2
    for j = 1:2

        % Function values
        N= [ (1-r(i))*(1-r(j))
              (1+r(i))*(1-r(j))
              (1+r(i))*(1+r(j))
              (1-r(i))*(1+r(j)) ]/4;
        % Parametric derivatives:
        dN = [ -(1-r(j))  (1-r(j))  (1+r(j))  -(1+r(j))
                -(1-r(i))  -(1+r(i))  (1+r(i))  (1-r(i)) ]/4;

        % transform to global coordinates
        Jt = dN*Xe(:,1:2);

        M = M + w(i)*w(j)*(N*N')*det(Jt);

    end
end
Me=zeros(dimT*4,dimT*4);
for i=1:4
    for j=1:4
        Me((i-1)*dimT+1:i*dimT,(j-1)*dimT+1:j*dimT)=M(i,j)*I;
    end
end
end

```

A.14. Plotter.m

```

%Plotter.m plots u and Tout.
% 'u' must be in the format: [node dof u]
%   and include all the nodes and dof of the domain.
% 'Tout' must be in the format: [node dof T]
% The domain must be rectangular and ordered as:
% | 1   ... m   |
% | m+1 ... 2*m |
% | ...   ...   |
% | m+n ... m*n |
% dim == 2

% Plot displacements.
ndof = size(u,1);
dofx = find(u(:,2) == 1);
dofy = find(u(:,2) == 2);
ux = u(dofx,3);
uy = u(dofy,3);

figure(100)
quiver(X(:,1),X(:,2),ux,uy,1); hold on
scatter(X(:,1),X(:,2),25,'k','fill'); hold on
xlabel('x'); ylabel('y');
title('Displacement field');
for i = min(X(:,1)):Mesh.Lx/Mesh.nx:max(X(:,1))
    x1 = [i,i];    y1 = [min(X(:,2)) max(X(:,2))];
    plot(x1,y1,'k'); hold on
end
for j = min(X(:,2)):Mesh.Ly/Mesh.ny:max(X(:,2))
    x2 = [min(X(:,1)) max(X(:,1))];    y2 = [j,j];
    plot(x2,y2,'k'); hold on
end
axis([min(X(:,1))-0.25 max(X(:,1))+0.25 min(X(:,2))-0.25...
    max(X(:,2))+0.25]);
hold off

% Plot forces
nodes = Tout(:,1);
dofs  = Tout(:,2);
vals  = Tout(:,3);
pX = Mesh.nx;
pY = Mesh.ny;
h = Mesh.Ly;
b = Mesh.Lx;

xNod = X(nodes,1);
yNod = X(nodes,2);

i = dofs==1;
j = dofs==2;
Tx = vals.*i;
Ty = vals.*j;

figure(200)
scatter(X(:,1),X(:,2),25,'k','fill'); hold on
for i = min(X(:,1)):Mesh.Lx/Mesh.nx:max(X(:,1))
    x1 = [i,i];    y1 = [min(X(:,2)) max(X(:,2))];
    plot(x1,y1,'k'); hold on

```

```
end
for j = min(X(:,2)):Mesh.Ly/Mesh.ny:max(X(:,2))
    x2 = [min(X(:,1)) max(X(:,1))];    y2 = [j,j];
    plot(x2,y2,'k'); hold on
end
quiver(xNod,yNod,Tx,Ty); hold on
axis([min(X(:,1))-0.25 max(X(:,1))+0.25 min(X(:,2))-0.25 ...
max(X(:,2))+0.25]);
xlabel('x'); ylabel('y');
title('Traction field');
hold off
```

A.15. Solver.m

```
function [ u1,T ] = solver(K0,K1,u0,M,A)
%SOLVER Checks if the system requires regularization or not and solves
the
%system applying the most appropriate method.
% In case the system requires regularization, a l-curve
regularization
% method is applied.

tol = 1e6;
Mi = inv(M);
Id=eye(size(A,2));
I0 = eye(size(A,1))-A*Id*A';
CN = cond(K1'*I0*K1);
vapMinAbs = min(abs(eig(K1'*I0*K1)));
display(CN)
display(vapMinAbs)
% if CN>tol
    display('CN is high. Stabilization is recommended'); beep
    stab = input('Use stabilization method? [0,1]\n Default: 1\n');
    if isempty(stab) == 1
        stab = 1;
    end
% end

% method = input(...)
%'Method to compute lambda:\n 1- Bisection\n 2- High res.\n Default:
1\n');
% if isempty(method) == 1
    method = 1;
% end
switch method
    case 1

        if stab == 1%(CN > tol & stab == 1)
            y = bisection(K0,K1,u0,M,A);
            Id=inv(A'*A+y*eye(size(A,2)));
            I0 = eye(size(A,1))-A*Id*A';
            u1 = (K1'*I0*K1)\(-K1'*I0*K0*u0);
            T = M\ (Id*A'*(K0*u0+K1*u1));
        else
            u1 = (K1'*I0*K1)\(-K1'*I0*K0*u0);
            T = M\ (A'*(K0*u0+K1*u1));
        end

    case 2

        if stab == 1%(CN > tol & stab == 1)
            [y0,y1] = logRank(K0,K1,u0,M,A);
            resu = refine(K0,K1,u0,M,A,y0,y1); % Column 1: y
                                                % Column 2: "dist"
                                                % Column 3: || Ku-AT

        ||
                                                % Column 4: || T ||

            [~,i] = min(resu(:,2));
            y = resu(i,1);
            Id=inv(A'*A+y*eye(size(A,2)));

```



```
I0 = eye(size(A,1))-A*Id*A';  
u1 = (K1'*I0*K1)\(-K1'*I0*K0*u0);  
T = M\ (Id*A'*(K0*u0+K1*u1));
```

```
L_Plotter(resu,i)
```

```
else
```

```
u1 = (K1'*I0*K1)\(-K1'*I0*K0*u0);  
T = M\ (A'*(K0*u0+K1*u1));
```

```
end
```

```
end  
end
```

