DISCONTINUOUS GALERKIN APPROXIMATION ON POLYHEDRAL MESH FOR THE ELASTOPLASTIC PROBLEM

Michele Precuzzi
Luca Caivano

July 7, 2021

Contents

1. Preface

1.1. Introduction to Discontinuous Galerkin

The idea of Discontinuous Galerkin methods is almost the same as the classic Galerkin methods but the solution is found in a discrete space made of polynomials completely discontinuous across mesh elements. Despite this method was originally proposed for hyperbolic problems (like the wave equation), today it is used in a large variety of situations, including elliptic problems, where usually the solution is searched in $H^1(\Omega)$, which is a space of virtually continuous functions.
Allowing discontinuity across mesh elements doubles the degrees of freedom on the internal edges, however there are many good features, like the more flexibility of the mesh, where usually the continuity assumption leads to some constrictions. Another advantage compared to continuous methods is that if we have a discontinuity in the solution, solving the Discontinuous Galerkin formulation of the problem allows to capture such behaviour.
Moreover polyhedric elements give us even more flexibility in the construction of the computational domain:

- since there are no restrictions about the mesh complexity and how the elements must join, almost every kind of grid is allowed
- in case of hanging nodes between two elements, these can be treated as standard nodes for both of them since there are no restriction in the number of edges of the elements
- even on simple geometries, the average number of elements needed to discretize complicated domains is smaller without enforcing any domain approximation

Notice that, due to the fact that the discrete space is constructed based on employing piecewise discontinuous polynomials, DG methods are naturally suited to robustly support polytopic meshes. Finally the discontinuity of the solution and the development of efficient quadrature rules on polytopic elements allow a massively-parallel implementation of such family of methods on parallel architectures. For a further discussion about this last topic we refer to [5].

We start with a brief introduction to the Discontinuous Galerkin method and the required functional analysis results. As a starting point, for an easier introduction, we apply it to the 2D Poisson problem, then we move to the quasi-static 2D elastic problem and, finally, we generalize the problem including the plastic response. In particular the main focus of this project is the implementation in Matlab of the Discontinuous Galerkin Finite Element method for the elastoplastic problem.

1.2. Preliminary results of functional analysis

All the following results will be presented in 2D since this is the case we are interested in.

Theorem 1.1 (Trace theorem). *Let $\Omega \subset \mathbb{R}^2$ be a bounded Lipschitz domain and let $\Gamma_0 \subset \partial\Omega$ be open. There exists a linear operator (the trace operator) $\tau_0 : H^1(\Omega) \to H^{\frac{1}{2}}(\Gamma_0)$ such that:*

- $\forall \varphi \in \mathcal{D}(\bar{\Omega})\ \ \tau_0\varphi = \varphi|_{\Gamma_0}$
- $\exists C = C(\Omega)$ *s.t.* $||\tau_0 u||_{H^{\frac{1}{2}}(\Gamma_0)} \leqslant C||u||_{H^1(\Omega)}$

Moreover there exists a second linear operator $\tau_1 : H^2(\Omega) \to H^{\frac{1}{2}}(\Gamma_0)$ such that:

- $\forall \varphi \in \mathcal{D}(\bar{\Omega})\ \ \tau_1\varphi = \nabla\varphi \cdot \mathbf{n}|_{\Gamma_0}$
- $\exists \tilde{C} = \tilde{C}(\Omega)$ *s.t.* $||\tau_1 u||_{H^{\frac{1}{2}}(\Gamma_0)} \leqslant \tilde{C}||u||_{H^2(\Omega)}$

Where we used $\mathcal{D}(\bar{\Omega})$ to indicate the space $C_0^\infty(\bar{\Omega})$ equipped with a specific notion of convergence that we don't show here.

Theorem 1.3 (Sobolev embedding theorem). *Let $\Omega \subset \mathbb{R}^2$ be a bounded Lipschitz domain and let $m > 1$, then $H^m(\Omega) \overset{C}{\hookrightarrow} C^{k,\alpha}(\bar{\Omega})$ where $k \in \mathbb{N}, 0 \leqslant k < m - 1$ and $\alpha \in (0, 1)$.*
In particular this last result tells us that in 2D every equivalence class in $H^2(\Omega)$ admits a continuous representative.

Theorem 1.4(Extension operator).*There exist an operator $\mathcal{E} : H^s(\Omega) \to H^s(\mathbb{R}^2)$ which is continuous and s.t. $\mathcal{E}(u)|_\Omega = u$ and $||\mathcal{E}(u)||_{H^s(\mathbb{R}^2)} \leqslant C||u||_{H^s(\Omega)}$*
For more details about the extension operator we remind to [3]

1.3. Properties of the polytopic mesh

In this section we state some properties that a mesh may have and that we will need in order to state a general convergence result for the DGFEM method.

Definition 1.1(Polytopic-regular mesh). *A mesh $\mathcal{T}_h$ is said to be polytopic-regular if for any $K \in \mathcal{T}_h$ there exists a set of non-overlapping (not necessarily shape-regular) d-dimensional simplices $\{S_K^F\}_{F\subset\partial K}$ contained in K, such that for all faces $F \subset \partial K$, the following condition holds:*

$$h_K \leqslant C\frac{d|S_K^F|}{|F|}$$

where the constant C is independent of the discretization parameters, the number of faces of the element, and the face measure.

Moreover we will need the following assumption:

Assumption 1.1 *There exists a covering $\mathcal{T}_\# = \{T_K\}$ of $\mathcal{T}_h$ (i.e. a set of shape-regular 2-dimensional simplices T_K s.t. $\forall K \in \mathcal{T}_h\ \exists T_K \in \mathcal{T}_\#$ s.t. $K \subsetneq T_K$) and a positive constant C_Ω, independent of the mesh parameters, such that:*

$$\max_{K\in\mathcal{T}_h} |\{K' \in \mathcal{T}_h : K' \cap T_K \neq \emptyset,\ T_K \in \mathcal{T}_\# \text{ s.t. } K \subset T_K\}| \leqslant C_\Omega$$

and $h_{T_K} \leqslant C h_K$ for each pair $K \in \mathcal{T}_h$ and $T_K \in \mathcal{T}_\#$ with $K \subset T_K$
This implies that, when the computational mesh $\mathcal{T}_h$ is refined, the amount of overlap present in the covering $\mathcal{T}_\#$ remains bounded.

2. The Poisson problem

Let $\Omega \subset \mathbb{R}^2$ be a bounded Lipschitz domain and let's consider the following Poisson problem with homogeneous Dirichlet boundary conditions:

$$\begin{cases} -\Delta u = f & \text{in } \Omega \subset \mathbb{R}^2 \\ u = 0 & \text{on } \partial\Omega \end{cases} \tag{2.1}$$

Let $\mathcal{T}_h = \{K_i\}_{i=1}^N$ be a triangulation of Ω, define $\Omega_h = \cup_{i=1}^N K_i$ and let $v \in W^1(\mathcal{T}_h) = \{v \in L^2(\Omega_h) : v|_k \in H^1(K)\, \forall K \in \mathcal{T}_h\}$. Let's now multiply the first equation of (2.1) by v, integrate on a generic element K_i, integrate by parts and sum over all the elements of the triangulation to finally obtain: find $u \in W^1(\mathcal{T}_h)$ s.t.

$$\sum_{i=1}^N \int_{K_i} \nabla u \cdot \nabla v ds \; - \; \sum_{i=1}^N \int_{\partial K_i} \nabla u \cdot n_k v dl \; = \; \int_{\Omega_h} f v ds \tag{2.2}$$

$\forall v \in W^1(\mathcal{T}_h)$

2.1. Useful notations and the Magic Formula

Before proceeding lets' introduce the following notations:

- We define as $\mathcal{F}_h^i$ the set of all the internal boundaries, namely $\mathcal{F}_h^i = \left(\cup_{i=1}^N \partial K_i\right) \backslash \partial\Omega_h$
- We define as $\mathcal{F}_h$ the set of the external boundaries, namely $\mathcal{F}_h = \partial\Omega_h$
- Finally we define $\mathcal{F}_h = \mathcal{F}_h^i \cup \mathcal{F}_h$

Let's now consider a piece of boundary $\Gamma \in \mathcal{F}_h^i$ shared by two elements K^+ and K^- , let n^+ and n^- be the outer normal directions of K^+ and K^- respectively. Let's consider a function $v \in W^1(\mathcal{T}_h)$ and, since in general v is not continuous, we define v^+ as the trace of $v|_{K^+}$ restricted on Γ and v^- as the trace of $v|_{K^-}$ restricted on Γ. Then we define:

- $\{\{v\}\} = \frac{v^+ + v^-}{2}$
- $[[v]] = \begin{cases} v^+ n^+ + v^- n^- & \text{if v is a scalar function} \\ v^+ \otimes n^+ + v^- \otimes n^- & \text{if v is a vector function} \end{cases}$

If $\Gamma \in \mathcal{F}_h$ with outer normal direction n then we define:

- $\{\{v\}\} = v$

- $[[v]] = \begin{cases} vn & \text{if v is a scalar function} \\ v \otimes n & \text{if v is a vector function} \end{cases}$

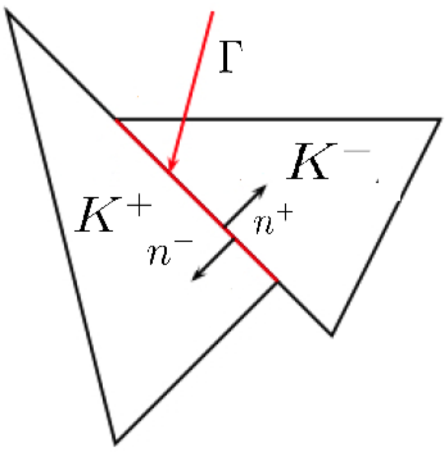


Fig1. Representation of an interface between two elements and the normal outer directions

We are now ready to state the following result that we are going to use in order to develop the complete weak form of the Discontinuous Galerkin approximation for the Poisson problem:

Theorem 2.1(Magic Formula). *$\forall \tau$ vector function the following result holds:*

$$\sum_{K \in \mathcal{T}_h} \int_{\partial K} \tau \cdot n_k d\gamma \; = \; \sum_{F \in \mathcal{F}_h} \int_F \{\{\tau\}\} \cdot [[v]]\, d\gamma \; + \; \sum_{F \in \mathcal{F}_h^i} \int_F [[\tau]] \cdot \{\{v\}\} d\gamma$$

Finally we state a regularity theorem for the Laplace problem:

Theorem 2.2(Elliptic regularity). *Let $\Omega \subset \mathbb{R}^2$ be a bounded and Lipschitz domain and let u be the solution of* (2.2), then:

- *If $f \in L^p(\Omega), 1 \leqslant p < \infty$ then $u \in \mathcal{W}^{2,p}(\Omega) = \{u \in L^p(\Omega) : D^\beta u \in L^p(\Omega)\, \forall \beta : |\beta| \leqslant 2\}$*
- *If $f \in L^\infty(\Omega)$ then $u \in C^{1,\alpha}(\Omega)$ $\forall \alpha$ s.t. $0 \leqslant \alpha < 1$*
- *If $f \in C^{0,\alpha}(\Omega)$ then $u \in C^{2,\alpha}(\Omega)$ $\forall \alpha$ s.t. $0 < \alpha < 1$*

2.2. Weak formulation

Thanks to the **Magic Formula** we can rewrite (2.2) as:
find $u \in W^1(\mathcal{T}_h)$ s.t.

$$\sum_{i=1}^{N} \int_{K_i} \nabla u \cdot \nabla v ds - \sum_{F \in \mathcal{F}_h} \int_F \{\{\nabla u\}\} \cdot [[v]]\, dl - \sum_{F \in \mathcal{F}_h^i} \int_F [[\nabla u]] \cdot \{\{v\}\} dl = \int_{\Omega_h} f v ds \;\; \forall v \in W^1(\mathcal{T}_h)$$

From now on we assume that $f \in L^2(\Omega)$, so that, by elliptic regularity, we have that $u \in \mathcal{W}^{2,2} \equiv H^2(\Omega)$ and, thanks to Sobolev embedding, this implies that u is continuous and so $[[u]] = 0$, more precisely $u \in C^{0,\alpha}$ and it can be shown that this implies also $[[\nabla u]] = 0$.
Thanks to these two observations we can rewrite the weak formulation as the following expression:
find $u \in W^1(\mathcal{T}_h)$ s.t.

$$\sum_{i=1}^{N} \int_{K_i} \nabla u \cdot \nabla v ds - \sum_{\Gamma \in \mathcal{F}_h} \int_\Gamma \{\{\nabla u\}\} \cdot [[v]]\, dl - \theta \sum_{\Gamma \in \mathcal{F}_h} \int_\Gamma [[u]] \cdot \{\{\nabla v\}\} dl$$
$$+ \sum_{\Gamma \in \mathcal{F}_h} \int_\Gamma \gamma\, [[u]] \cdot [[v]]\, dl = \int_{\Omega_h} f v ds \quad \forall v \in W^1(\mathcal{T}_h)$$

Note that this formulation is consistent since we removed and added only terms that are equal to zero, however, passing to the discrete formulation, they will not be zero anymore since we are looking for discontinuous solutions.
Note that no boundary conditions are imposed since they will be satisfied in a weak sense (like Neumann boundary conditions for the continuous Galerkin methods), we will see some more details later about that.
The coefficient γ is for stabilization and it is also called "penalty coefficient". Usually, if we are integrating on an edge Γ shared by two elements K^+ and K^- whose sizes are h^+ and h^- respectively, one can set $p|_F = \alpha \frac{p^2}{h}$ where $p = \max\{p(K^+), p(K^-)\}, h = \min\{h^+, h^-\}$ and α is a given coefficient that is very case-sensitive and must be given or tuned. About θ we have three choices and each one leads to a different method:

- $\theta = 1 \rightarrow$ "Symmetric Interior Penalty" (SIP). The name comes from the fact that the associated bilinear form and the matrix associated to the linear system are symmetric.
- $\theta = -1 \rightarrow$ "Non-Symmetric Interior Penalty" (NIP).
- $\theta = 0 \rightarrow$ "Incomplete Interior Penalty" (IIP).

2.3. Space discretization

The discrete test space will be $V_h = \{v_h \in L^2(\Omega_h) : v_h|_K \in \mathbb{P}^p(K)\, \forall K \in \mathcal{T}_h\} \nsubseteq H_0^1(\Omega_h)$ where in general p can depend on K. This choice is very common and allows to build a basis and a related algebraic system in the very same way as it is done for the Continuous Galerkin methods. However there is a crucial difference: considering an internal edge F shared by two elements K^+ and K^-, we have to take into account the basis functions related to both elements which are independent since we are dealing with a (discrete) space of discontinuous functions and this also heavily increases the number of degrees of freedom compared a Continuous Galerkin method.

2.4. Well-posedness of the problem

In the following section we are going to show some results about the well-posedness of the problem for all the three versions of the method (SIP, NIP and IIP). Although these results concern the Poisson problem, an extension is possible for a generic Advection-Diffusion-Reaction equation and for a further discussion about this topic we remind to [1].
As a first step, since we are dealing with discontinuous functions, we define the following norms:

- $||v||^2_{H^s(\mathcal{T}_h)} = \sum_{K\in\mathcal{T}_h} ||v||^2_{H^s(K)}$
- $||v||^2_{L^2(\mathcal{F}_h)} = \sum_{F\in\mathcal{F}_h} ||v||^2_{L^2(F)}$
- $||v||^2_{DG} = ||\nabla_h v||^2_{L^2(\Omega)} + ||\gamma^{\frac{1}{2}}\, [[v]]\, ||^2_{L^2(\mathcal{F}_h)}$
- $|||v|||^2_{DG} = ||v||^2_{DG} + ||\gamma^{-\frac{1}{2}}\{\{\nabla_h v\}\}||^2_{L^2(\mathcal{F}_h)}$

Where $\nabla_h v$ is defined such that $(\nabla_h v)\,|_K = \nabla\,(v|_K)$
Moreover we define the following bilinear form:

$$\mathcal{A}_\theta(u,v) = \sum_{i=1}^{N}\int_{K_i} \nabla u\cdot\nabla v ds - \sum_{F\in\mathcal{F}_h}\int_F \{\{\nabla u\}\}\cdot[[v]]\, dl - \theta\sum_{F\in\mathcal{F}_h}\int_F [[u]]\cdot\{\{\nabla v\}\} dl + \sum_{F\in\mathcal{F}_h}\int_F \gamma\,[[u]]\cdot[[v]]\, dl$$

Theorem 2.3. *The bilinear form $\mathcal{A}_\theta$ is continuous in the sense that $\exists C > 0$:*
$|\mathcal{A}_\theta(v, w_h) \leqslant C|||v|||_{DG}||w_h||_{DG}\ \forall v \in W^2(\Omega) = \{v \in L^2(\Omega_h) : v|_k \in H^2(K)\, \forall K \in \mathcal{T}_h\},\ \forall w_h \in V_h$.
Moreover for the NIP version it is also coercive: $\exists\bar{C} > 0 : \mathcal{A}_\theta(v_h, v_h) \geqslant \bar{C}||v_h||^2_{DG}$
The same result can be obtained for SIP and IIP if the penalty coefficient α is large enough.

It is possible to show that this result is sufficient in order to prove the well-posedness of the problem.

2.5. Convergence results

Theorem 2.4. *Let $u \in H^{s+1}(\Omega)$ be the exact solution to the problem and let u_h be the solution of the discretized weak problem. Assume also that, in case of SIP or IIP, the penalty coefficient is large enough to ensure the well-posedness of the problem, moreover let h be a representative of the mesh size and, for simplicity, let's assume that the polynomial degree p is the same for all the elements of the mesh. Then the following convergence estimates hold:*

- $\exists C > 0 : ||u - u_h||_{DG} \leqslant C \frac{h^{\min\{p,s\}}}{p^{s-\frac{1}{2}}} ||u||_{H^{s+1}(\mathcal{T}_h)}$
- $\exists \bar{C} > 0 : ||u - u_h||_{L^2(\mathcal{T}_h)} \leqslant \bar{C} \frac{h^{\min\{p,s\}+1}}{p^{s+\frac{1}{2}}} ||u||_{H^{s+1}(\mathcal{T}_h)}$ *for SIP*
- $\exists \bar{C} > 0 : ||u - u_h||_{L^2(\mathcal{T}_h)} \leqslant \bar{C} \frac{h^{\min\{p,s\}}}{p^{s-\frac{1}{2}}} ||u||_{H^{s+1}(\mathcal{T}_h)}$ *for IIP and NIP*

Moreover, if $u \in H^s(\Omega)\, \forall s > 0$ then there is an exponential decay of the error in the polynomial degree.

2.6. Generic boundary conditions

2.6.1. Non-homogeneous Dirichlet boundary conditions

Let's consider the following problem:

$$\begin{cases} -\Delta u = f & \text{in } \Omega \subset \mathbb{R}^2 \\ u = g \not\equiv 0 & \text{on } \partial\Omega \end{cases}$$

In this case, if F is an external boundary (i.e. $F \subset \partial\Omega_h$) we define $[[u]] = u - g$, so that the bilinear form $\mathcal{A}_\theta$ remains the same, while the right-hand side presents an additional term, indeed the problem now reads:
find $u \in W^1(\mathcal{T}_h)$ s.t.

$$\mathcal{A}_\theta(u, v) = \int_{\Omega_h} fv ds + \sum_{F \in \mathcal{F}_h} \int_F g\{\{\nabla_h v\}\} dl \qquad \forall v \in V_h$$

2.6.2. Mixed boundary conditions

In case of Neumann boundary conditions the **Magic Formula** doesn't apply on the edges on which such conditions are imposed since the terms associated to them are known. To state a general result, consider the following space-discretized mixed problem:

$$\begin{cases} -\Delta u = f & \text{in } \Omega_h \\ u = g & \text{on } \Gamma_D \\ \nabla u \cdot n = h & \text{on } \Gamma_N \end{cases}$$

where $\Gamma_D \cap \Gamma_N = \varnothing$ and $\Gamma_D \cup \Gamma_n = \partial\Omega_h$. We now define:

- $\mathcal{F}_h^D = \{F \in \mathcal{F}_h : \text{Dirichlet boundary condition is imposed on } F\}$
- $\mathcal{F}_h^N = \mathcal{F}_h \backslash \mathcal{F}_h^D$
- $\mathcal{A}_\theta : W^1 \times W^1 \to \mathbb{R} : \mathcal{A}_\theta(u,v) = \sum_{i=1}^N \int_{K_i} \nabla u \cdot \nabla v ds \; - \; \sum_{F \in \mathcal{F}_h^i \cup \mathcal{F}_h^D} \int_F \{\{\nabla u\}\} \cdot [[v]]\, dl$ $- \theta \sum_{F \in \mathcal{F}_h^i \cup \mathcal{F}_h^D} \int_F [[u]] \cdot \{\{\nabla v\}\} dl \; + \; \sum_{F \in \mathcal{F}_h^i \cup \mathcal{F}_h^D} \int_F \gamma\, [[u]] \cdot [[v]]\, dl$
- $F : W^1 \to \mathbb{R} : Fv = \int_{\Omega_h} f v_h ds + \sum_{F \in \mathcal{F}_h^D} \int_F g \{\{\nabla_h v_h\}\} dl + \sum_{F \in \mathcal{F}_h^N} \int_F hv dl$

then, following the same argument for the previous derivation it is possible to show that the weak formulation of the problem reads:
find $u \in W^1(\mathcal{T}_h)$ s.t.

$$\mathcal{A}_\theta(u,v) = Fv \quad \forall v \in W^1(\mathcal{T}_h)$$

3. The Elastic problem

3.1. Description of the problem

Let $\Omega \subset \mathbb{R}^2$ be an open bounded set. We are going to solve the linear time-independent elastic problem, which formulates as the following:

$$\begin{cases} -\nabla \cdot \sigma\left(u\left(x,y\right)\right) = f\left(x,y\right) & \text{in } \Omega \subset \mathbb{R}^2 \\ \text{boundary conditions} & \end{cases}$$

where $f \in L^2(\Omega)$ denotes the external loading and $\sigma : \Omega \to \mathbb{S} = \{3 \times 3 \text{ symmetric tensors}\}$ is the stress tensor, for which we assume the Hook's constitutive law: $\sigma(u) = \mathcal{D}\varepsilon(u)$ where the fourth order stiffness tensor $\mathcal{D} : \mathbb{S} \to \mathbb{S}$ is defined such that $\mathcal{D}\tau = 2\mu\tau + \lambda Tr(\tau)\mathbb{I}_3 \ \forall \tau \in \mathbb{S}$ and $\varepsilon(u) = \frac{1}{2}\left(\nabla u + (\nabla u)^T\right)$ is the symmetric part of the gradient of u. Here, $\mathbb{I}_3$ is the 3×3 identity tensor, $Tr(\cdot)$ represents the trace operator, and μ, $\lambda \in L^\infty(\Omega)$ are the first and the second Lamè parameters, respectively. We assume that $\mathcal{D}$ is symmetric, positive definite and uniformly bounded over Ω.

3.2. Weak formulation of the space-discretized problem

We introduced the possibility to set different boundary conditions for each component, in particular we will treat Dirichlet and Neumann boundary conditions. Under this hypothesis, the most general problem is in the following form:

$$\begin{cases} -\nabla \cdot \sigma\left(u\left(x,y\right)\right) = f\left(x,y\right) & \text{in } \Omega \subset \mathbb{R}^2 \\ u \cdot n = u_{D,n} & \text{on } \Gamma_{D,n} \\ u \cdot t = u_{D,t} & \text{on } \Gamma_{D,t} \\ n^T \sigma(u)\, n = h_n & \text{on } \Gamma_{N,n} \\ t^T \sigma(u)\, n = h_t & \text{on } \Gamma_{N,t} \end{cases}$$

where $u = [u_x, u_y]$, n is the normal outer direction and t is s.t. $||t|| = 1$ and $t \cdot n = 0$, moreover $\Gamma_{D,n} \cup \Gamma_{N,n} = \Gamma_{D,t} \cup \Gamma_{N,t} = \partial\Omega$

The possibility to set independent types of boundary condition for each component allow to study also situations described in *Fig2*, where Neumann boundary conditions are imposed on the horizontal edges for both components and on the vertical edges in the normal direction while Dirichlet boundary conditions are imposed on the vertical edges in the tangent direction.

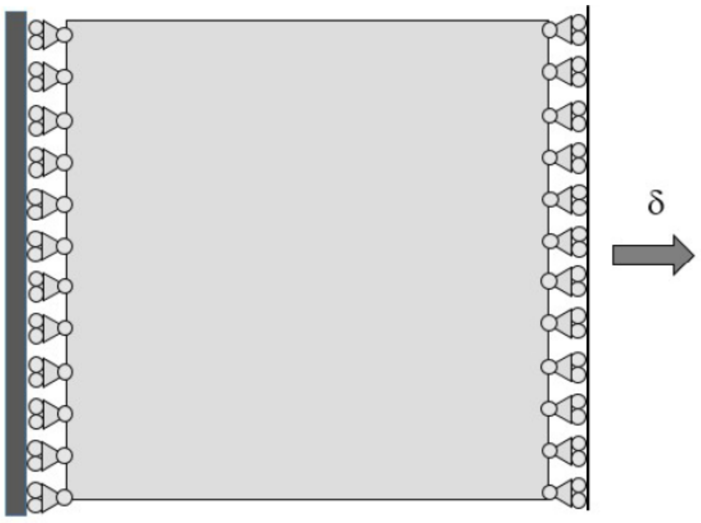


Fig2. Example of problem with mixed boundary conditions

We now introduce the following definitions:

- $\mathcal{F}_{h,n}^{D} = \{F \in \mathcal{F}_h : \text{Dirichlet boundary condition on the normal direction is imposed on } F\}$
- $\mathcal{F}_{h,n}^{N} = \mathcal{F}_h \backslash \mathcal{F}_{h,n}^{D}$
- $\mathcal{F}_{h,t}^{D} = \{F \in \mathcal{F}_h : \text{Dirichlet boundary condition on the tangent direction is imposed on } F\}$
- $\mathcal{F}_{h,t}^{N} = \mathcal{F}_h \backslash \mathcal{F}_{h,t}^{D}$
- $\mathcal{F}_{h}^{i,D} = \mathcal{F}_{h,n}^{D} \cup \mathcal{F}_{h,t}^{D} \cup \mathcal{F}_{h}^{i}$
- $\mathcal{F}_{h}^{i,N} = \mathcal{F}_{h,n}^{N} \cup \mathcal{F}_{h,t}^{N} \cup \mathcal{F}_{h}^{i}$
- $W_h^{DG} = \{w \in [L^2(\Omega)]^2 : w|_K \in [\mathbb{P}^{p_k}(K)]^2 \, \forall K \in \mathcal{T}_h\}$
- $\mathcal{A}_\theta : W_h^{DG} \times W_h^{DG} \to \mathbb{R} : \mathcal{A}_\theta(u,v) = \sum_{i=1}^{N} \int_{K_i} \sigma(u) : \varepsilon(v)\, ds \;-\; \sum_{F \in \mathcal{F}_h^{i,D}} \int_F \{\{\sigma(u)\}\} : [[\tilde{v}]]\, dl$ $-\;\theta \sum_{F \in \mathcal{F}_h^{i,D}} \int_F [[u]] : \{\{\widetilde{\sigma(v)}\}\} dl \;+\; \sum_{F \in \mathcal{F}_h^{i,D}} \int_F \gamma\, [[u]] : [[\tilde{v}]]\, dl$
- $F : W_h^{DG} \to \mathbb{R} : Fv = \int_{\Omega_h} fv ds + \sum_{F \in \left(\mathcal{F}_{h,n}^{D} \cup \mathcal{F}_{h,t}^{D}\right)} \int_F g \{\{\widetilde{\sigma(v)}\}\} dl + \sum_{F \in \left(\mathcal{F}_{h,n}^{N} \cup \mathcal{F}_{h,t}^{N}\right)} \int_F h \tilde{v} dl$

Where $g = u_{D,n}\mathbf{n} + u_{D,t}\mathbf{t}, \quad h = h_n\mathbf{n} + h_t\mathbf{t}$ and:

$$\begin{cases} \tilde{v} = (v \cdot \mathbf{n})\,\mathbf{n} & \text{on } \Gamma_{D,n} \backslash \Gamma_{D,t} \\ \tilde{v} = (v \cdot \mathbf{t})\,\mathbf{t} & \text{on } \Gamma_{D,t} \backslash \Gamma_{D,n} \\ \tilde{v} = v & \text{otherwise} \end{cases} \qquad \begin{cases} \widetilde{\sigma} = \left(\mathbf{n}^T \sigma \mathbf{n}\right)\mathbf{n} & \text{on } \Gamma_{N,n} \backslash \Gamma_{N,t} \\ \widetilde{\sigma} = \left(\mathbf{t}^T \sigma \mathbf{n}\right)\mathbf{t} & \text{on } \Gamma_{N,t} \backslash \Gamma_{N,n} \\ \widetilde{\sigma} = \sigma & \text{otherwise} \end{cases}$$

Following the same argument presented in the previous sections (taking care of the presence of two components) we can obtain the weak formulation which reads as follows:
Find $u_h \in W_h^{DG}$ such that:

$$\mathcal{A}_\theta(u_h, v_h) = F v_h \qquad \forall v_h \in W_h^{DG}$$

3.3. Well-posedness and error analysis

In this section we briefly present some theoretical results that guarantee the well-posedness of the problem and also provide a convergence result in terms of polynomials degrees and elements sizes for all the 3 possible values for θ.
Let's start specifying that for the elastic problem the penalization coefficient γ is defined in the following way:

$$\begin{cases} \gamma = \alpha \bar{\mathcal{D}}_K \max_{K \in \{K^+, K^-\}} \frac{p_K^2}{h_K} & \text{on } F \subset \partial K^+ \cap \partial K^- \\ \gamma = \alpha \bar{\mathcal{D}}_K \frac{p_K^2}{h_K} & \text{on } F \subset \partial K \cap \left(\mathcal{F}_{h,n}^D \cup \mathcal{F}_{h,t}^D\right) \end{cases}$$

where $\bar{\mathcal{D}}_K = |\left(\mathcal{D}|_K\right)^{\frac{1}{2}}|_2^2 \; \forall K \in \mathcal{T}_h$ (here $|\cdot|_2$ is the operator norm induced by the l_2 norm on $\mathbb{R}^3$), and α is a positive parameter.

We now introduce the following norm for the space W_h^{DG}:

$$||v||_{DG}^2 = ||\mathcal{D}^{\frac{1}{2}} \varepsilon\left(v\right)||_{L^2(\Omega)}^2 + ||\gamma^{\frac{1}{2}} \left[\left[v\right]\right]||_{L^2\left(\mathcal{F}_h^{i,D}\right)}^2 \qquad \forall v \in W_h^{DG}$$

From now on we assume that we are working on a polytopic-regular mesh (**Definition 1.1**), moreover we work also under the **Assumption 1.1**.
Under this hypothesis the following lemmas are valid:

Lemma 3.1(Continuity and Coercivity). *For the NIP DGFEM formulation of the problem the following relations holds for every value of α:*

$$\mathcal{A}_\theta\left(v,v\right) \geqslant C_1 ||v||_{DG}^2 \quad \forall v \in W_h^{DG} \qquad\qquad \mathcal{A}_\theta\left(v,w\right) \leqslant C_2 ||v||_{DG} ||w||_{DG} \quad \forall v, w \in W_h^{DG}$$

moreover, if α is sufficiently large, they hold also for the SIP and IIP formulations.

Lemma 3.2 $\forall w \in W_h^{DG}$ *we have that:*

$$||\gamma^{-\frac{1}{2}}\{\{w\}\}||_{L^2\left(\mathcal{F}_h^{i,D}\right)}^2 \leqslant \frac{C}{\alpha} ||w||_{L^2(\Omega)}$$

Lemma 3.3 $\forall w \in W_h^{DG}$ *we have that:*

$$\sum_{F \in \mathcal{F}_h^{i,D}} \int_F \left[\left[v\right]\right] : \left[\left[v\right]\right] dl \leqslant \frac{C}{\alpha} ||\gamma^{-\frac{1}{2}} \left[\left[w\right]\right]||_{L^2\left(\mathcal{F}_h^{i,D}\right)}^2$$

We are finally ready to state the main result about convergence:

Theorem 3.1 *Let $\mathcal{T}_h$ be polytopic-regular and such that the* **Assumption 3.1** *is valid and let $\mathcal{E}$ be the extension operator defined in* **Theorem 1.4.** *Assume that the exact solution u is sufficiently regular, let's say $u \in H^{r_K}(T_K)$, $\sigma(u) \in \mathcal{H}^{r_K}(T_K) = [H^{r_K}(T_K)]^{2\times 2}_{sym}$ $\forall K \in \mathcal{T}_h$. Let $u_h \in W_h^{DG}$ be the DGFEM solution obtained with a penalty parameter α sufficiently large. Then the following bounds hold:*

- $||u - u_h||^2_{DG} \leqslant C \sum_{K \in \mathcal{T}_h} \frac{h_K^{2(s_K-1)}}{p_K^{2(r_K-3/2)}} \left(||\mathcal{E}u||^2_{H^{r_K}(T_K)} + ||\mathcal{E}\sigma(u)||^2_{\mathcal{H}^{r_K}(T_K)} \right)$
- $||u - u_h||^2_{L^2} \leqslant C \sum_{K \in \mathcal{T}_h} \frac{h_K^{2s_K}}{p_K^{2(r_K-1/2)}} \left(||\mathcal{E}u||^2_{H^{r_K}(T_K)} + ||\mathcal{E}\sigma(u)||^2_{\mathcal{H}^{r_K}(T_K)} \right)$ for SIP
- $||u - u_h||^2_{L^2} \leqslant C \sum_{K \in \mathcal{T}_h} \frac{h_K^{2(s_K-1)}}{p_K^{2(r_K-3/2)}} \left(||\mathcal{E}u||^2_{H^{r_K}(T_K)} + ||\mathcal{E}\sigma(u)||^2_{\mathcal{H}^{r_K}(T_K)} \right)$ for NIP and IIP

with $s_K = \min(p_K + 1, r_K)$ $\forall K \in \mathcal{T}_h$. The constant C depends on the material parameters and the shape-regularity of T_K , but is independent of h_K, p_K and the number of elements faces.
As we discussed above, in case of smooth solution we observe an exponential decay of the errors w.r.t. the polynomials degree p.

3.4. Linear system

In this section we derive the linear system obtained from the DGFEM scheme, needed in order to perform the numerical simulation.
We first stress the fact that the basis functions in our case are monomials. The idea is to construct a base $\{\Phi_i\}_{i=1}^{N_h}$ for a single component and then the complete base will be $\{[\Phi_1, 0], [\Phi_2, 0], \dots,$ $\dots, [\Phi_{N_h}, 0], [0, \Phi_1], [0, \Phi_2], \dots, [0, \Phi_{N_h}]\}$ so that $u(x,y) = \left[\sum_{i=1}^{N_h} u_i^x \Phi_i(x,y),\ \sum_{i=1}^{N_h} u_i^y \Phi_i(x,y)\right]$.
Once we have done it we define the following $2N_h \times 2N_h$ matrices $\mathbf{V}, \mathbf{IT}$ and $\mathbf{S}$ and the $2N_h$-dimensional vectors $\mathbf{f}$ and $\mathbf{u}$ where:

- $\mathbf{V}$ collects the contribution of $\sum_{i=1}^{N} \int_{K_i} \sigma(u) : \varepsilon(v)\, ds$
- $\mathbf{IT}$ collects the contribution of $\sum_{F \in \mathcal{F}_h^{i,D}} \int_F \{\{\sigma(u)\}\} : [[\tilde{v}]]\, dl$, so that (exploiting the linearity of the problem) $\mathbf{IT}^T$ represents $\sum_{F \in \mathcal{F}_h^{i,D}} \int_F [[u]] : \{\{\widetilde{\sigma(v)}\}\} dl$
- $\mathbf{S}$ collects the contribution of $\sum_{F \in \mathcal{F}_h^{i,D}} \int_F \gamma\, [[u]] : [[\tilde{v}]]\, dl$
- $\mathbf{f}$ represents Fv
- finally $\mathbf{u} = \left[u_1^x, u_2^x, \dots, u_{N_h}^x, u_1^y, u_2^y, \dots, u_{N_h}^y\right]^T$

Let's inspect the matrix $\mathbf{V}$ to see how to handle the multiple dimension, the argument will be the same for the other matrices and $\mathbf{f}$:
$\mathbf{V}$ is divided in 4 $N_h \times N_h$ sub-matrices $\mathbf{V} = \begin{bmatrix} \mathbf{V1} & \mathbf{V2} \\ \mathbf{V3} & \mathbf{V4} \end{bmatrix}$ with the following roles:

- $\mathbf{V1}(i,j) = \sum_{i=1}^{N} \int_{K_i} \sigma([\Phi_j, 0]) : \varepsilon([\Phi_i, 0])\, ds$
- $\mathbf{V2}(i,j) = \sum_{i=1}^{N} \int_{K_i} \sigma([0, \Phi_j]) : \varepsilon([\Phi_i, 0])\, ds$
- $\mathbf{V3}(i,j) = \sum_{i=1}^{N} \int_{K_i} \sigma([\Phi_j, 0]) : \varepsilon([0, \Phi_i])\, ds$
- $\mathbf{V4}(i,j) = \sum_{i=1}^{N} \int_{K_i} \sigma([0, \Phi_j]) : \varepsilon([0, \Phi_i])\, ds$

All the integrals are computed using a quadrature method.
In this way the algebraic formulation of the problem reads:

$$\text{find } \mathbf{u} \in \mathbb{R}^{2N_h} \text{ s.t. } \mathbf{Ku} = \mathbf{f}$$

where $\mathbf{K} = \mathbf{V} - \mathbf{IT} - \theta \cdot \mathbf{IT}^T + \mathbf{S}$

3.5. Numerical results

In this section we present the solutions obtained with our code on some benchmark problems.

3.5.1. Smooth solution with regular geometry and mesh

The first problem we are considering is the following:

$$\begin{cases} -\nabla \cdot \sigma(u(x,y)) = f(x,y) & \text{in } \Omega = [0,1]^2 \\ u \cdot n = u_{D,n} & \text{on } \Gamma_{D,n} = \partial\Omega \backslash \{0.1 \leqslant x \leqslant 1, \quad y = 1\} \\ n^T \sigma(u)\, n = h_n & \text{on } \Gamma_{N,n} = \{y = 1, 0.1 \leqslant x \leqslant 1\} \\ t^T \sigma(u)\, n = h_t & \text{on } \Gamma_{N,t} = \partial\Omega \end{cases}$$

$$\text{where: } \begin{cases} u_{D,n} = \sin(\pi x)\sin(\pi y) \\ h_n = -8.78 * 10^6 \pi \sin(\pi x) & \text{on } \{0.1 \leqslant x \leqslant 1, \quad y = 1\} \\ h_n = -3.38 * 10^6 \pi \sin(\pi y) & \text{on } \{x = 0, \quad 0 \leqslant y \leqslant 1, \} \\ h_n = 8.78 * 10^6 \pi \sin(\pi x) & \text{on } \{0 \leqslant x \leqslant 1, \quad y = 0\} \\ h_n = 3.38 * 10^6 \pi \sin(\pi y) & \text{on } \{x = 1, \quad 0 \leqslant y \leqslant 1, \} \\ h_t = -3.38 * 10^6 \pi \sin(\pi x) & \text{on } \{0.1 \leqslant x \leqslant 1, \quad y = 1\} \\ f_x(x,y) = f_y(x,y) = 9.12 * 10^7 \pi^2 \sin(\pi x)\sin(\pi y) - 8.45 * 10^7 \pi^2 \cos(\pi x)\cos(\pi y) \end{cases}$$

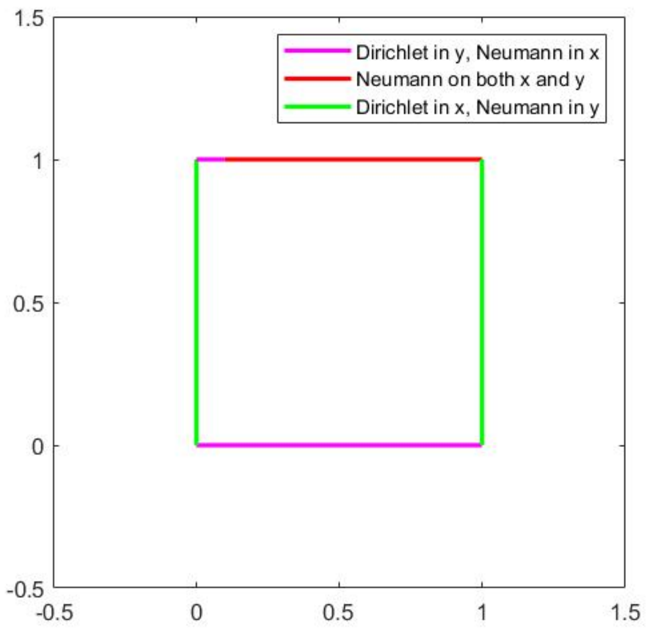


Fig3. Representation of mixed boundary conditions

With this data the exact solution is $u_{ex}(x,y) = [\sin(\pi x)\sin(\pi y)\,,\ \sin(\pi x)\sin(\pi y)]$.
We solve this problem through the SIP DGFEM formulation, with penalty coefficient equals to 10, on a simple structured mesh with square elements. The aim of this problem is mainly to test the implementation of mixed boundary conditions for the two components, verifying the accordance between theoretical and experimental convergence rates w.r.t. the mesh size and the polynomial degree of the basis functions.
The solution is C^{∞}, so, by **Theorem 3.1**, fixing a polynomial degree p we expect a convergence w.r.t. the mesh size h of order $p+1$ for the L^2-norm of the error and of order p for the H^1-norm, while, fixing the mesh size, we expect an exponential decay of the error w.r.t. p.

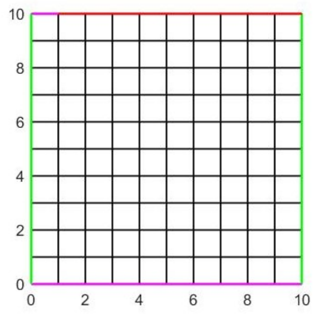


Fig4. Structured square mesh with 100 elements

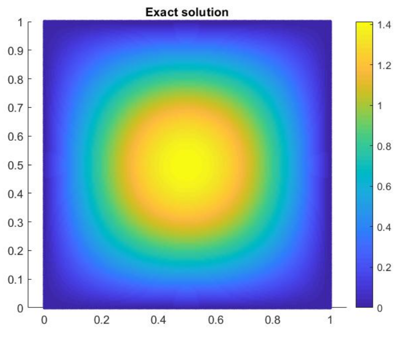


Fig5. Total displacement of the exact solution

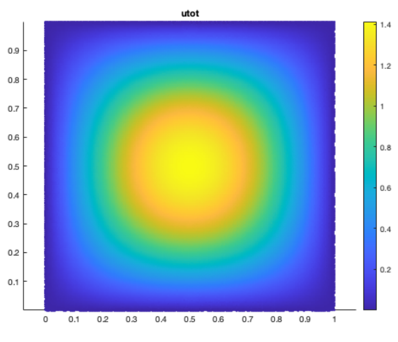


Fig6. Total displacement of the numerical solution

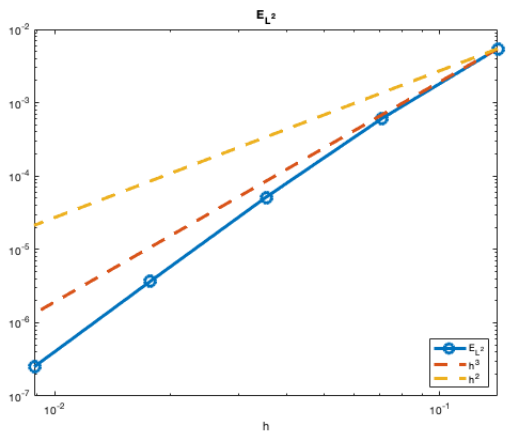


Fig7. Convergence in h of
$||u_{ex} - u_h||_{L^2}$ *with* $p = 2$

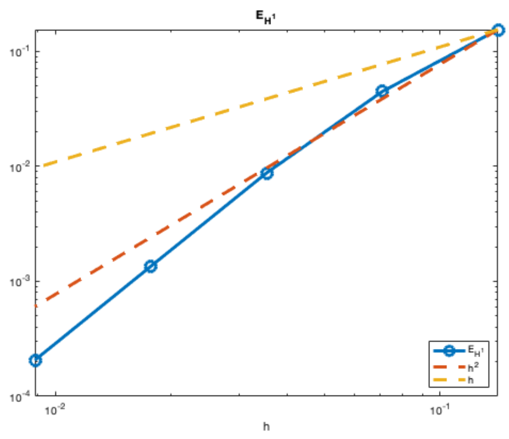


Fig8. Convergence in h of
$||u_{ex} - u_h||_{H^1}$ *with* $p = 2$

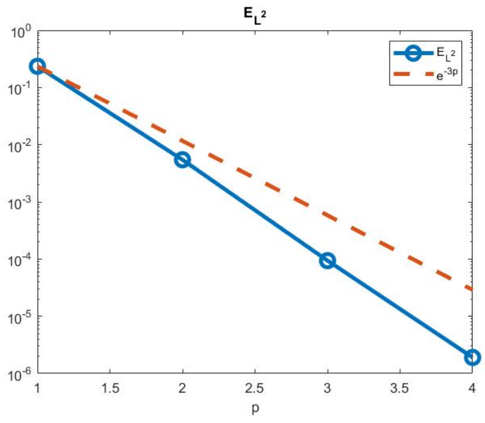


Fig9. Convergence in p of
$||u_{ex} - u_h||_{L^2}$ *with* $N = 100$

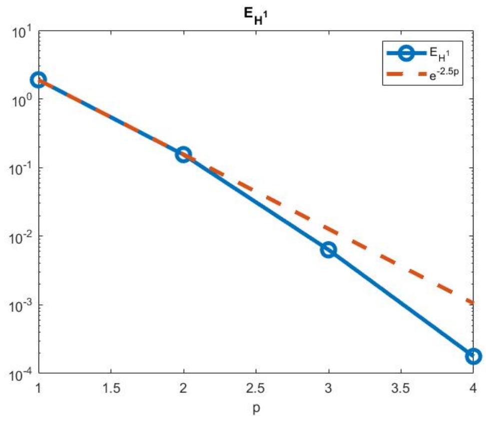


Fig10. Convergence in p of
$||u_{ex} - u_h||_{H^1}$ *with* $N = 100$

The figures above confirm our argument. In particular we find orders of convergence even a little higher then the theoretical estimates but this is fine since the latter is only a lower bound.
Note that figures 5 and 6 show the total displacements, for instance in figure 5 the colors represent the quantity $\sqrt{u_{x,ex}^2 + u_{y,ex}^2}$.
Finally observe that for figures 7 and 8 both axis are in logarithm scale, while for figures 9 and 10, in order to better observe the exponential behaviour, we set only the y axis in logarithm scale.

3.5.2. Smooth solution on a perforated plate

The second benchmark case that we present is a rectangular plate with a quarter of circle missing in the bottom left corner. For this problem we adopt a non trivial polyhedric mesh, as it is shown in the following figure.

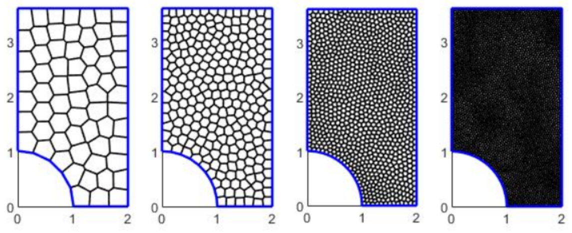


Fig11. Perforated plate with 4 mesh refinements

We want the exact solution to be $u_{ex}(x,y) = [\exp(-2x)\cos(y),\ \exp(-3y)\sin(x)]$, so we set:

$$\begin{cases} f_x(x,y) = 2.53 * 10^8 \exp -3y\cos(x) - 3.48 * 10^8 \exp -2x\cos(y) \\ f_y(x,y) = -1.69 * 10^8 \exp -2x\sin(y) - 7.87 * 10^8 \exp -3y\sin(x) \end{cases}$$

and we impose on all the boundary on both directions a Dirichlet boundary condition equal to u_{ex}.

For this problem, as in the previous case, we set the penalty coefficient equal to 10.

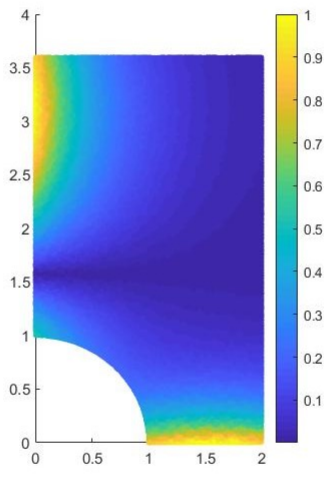


Fig12. Total displacement of the exact solution

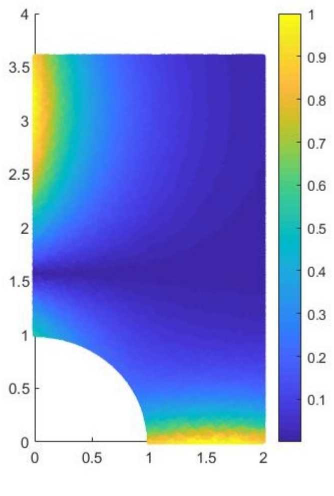


Fig13. Total displacement of the numerical solution

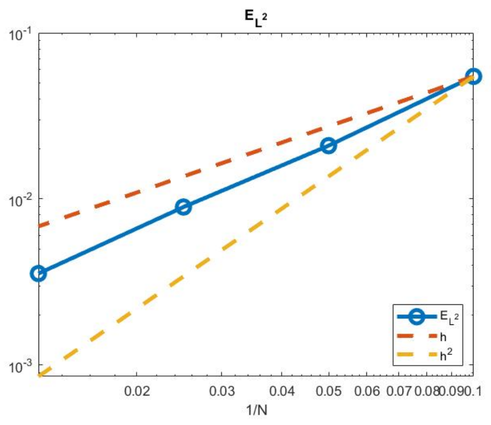


Fig14. Convergence in h of
$||u_{ex} - u_h||_{L^2}$ *with* $p = 1, \ \theta = 0$

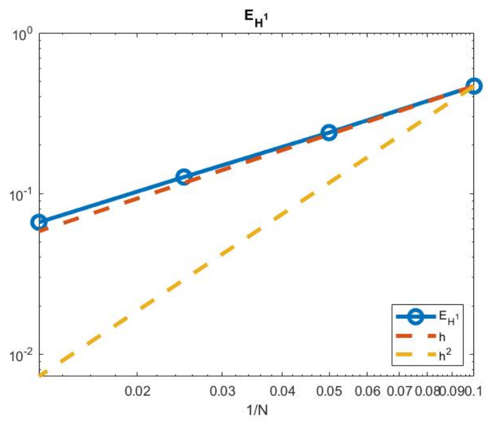


Fig15. Convergence in h of
$||u_{ex} - u_h||_{H^1}$ *with* $p = 1, \ \theta = 0$

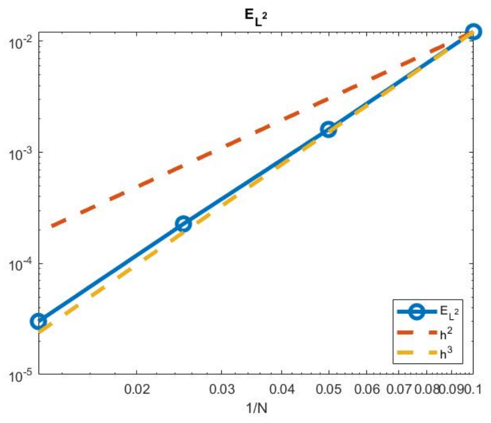


Fig16. Convergence in h of
$||u_{ex} - u_h||_{L^2}$ *with* $p = 2, \ \theta = 1$

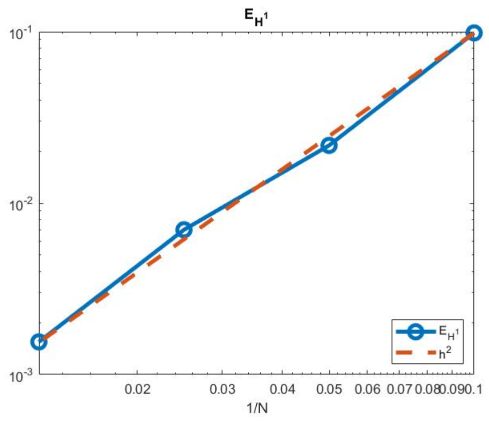


Fig17. Convergence in h of
$||u_{ex} - u_h||_{H^1}$ *with* $p = 2, \ \theta = 1$

Figures 12 and 13 show the total displacement of the exact and the numerical solution respectively, the latter was obtained solving the SIP SGFEM formulation on the most refined grid setting $p = 2$. Figures 14 and 15 show the convergence result for the IIP DGFEM formulation of the problem ($\theta = 0$) solved on the four meshes. Note that for the IIP formulation $||u_{ex}-u_h||_{L^2}$ and $||u_{ex}-u_h||_{H^1}$ have the same order of convergence as we could predict by **Theorem 3.1**.
For figures 16 and 17 we set again $\theta = 1$ and we increase the polynomial degree up to $p = 2$ and once again the results match the theoretical estimates.

3.5.3. Pulling force on a wheel

As a simple application we show the solution of the SIP DGFEM formulation of the elastic problem on a circular domain with a hole in the center (once again the penalty coefficient is set equal to 10).

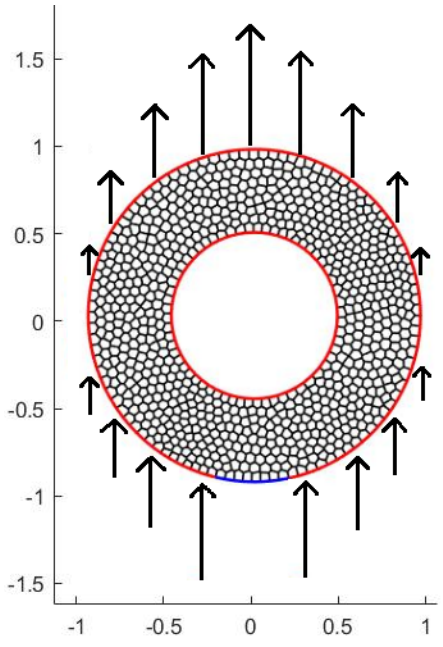


Fig18. Computational mesh and boundary conditions

Figure 18 shows the computational mesh employed to solve the problem and the boundary conditions: on the whole boundary a Neumann condition is imposed (except for a small portion of the bottom part of the external boundary), homogeneous on the internal boundary and equal to $6 * 10^4\,(x^2 - 16)$ on the external boundary directed as illustrated in the figure. In order to have a well-posed problem we impose an homogeneous Dirichlet boundary condition on the bottom of the domain: indeed if we impose Neumann everywhere then the solution is not unique anymore (uniqueness holds up to a constant displacement, so a translation)

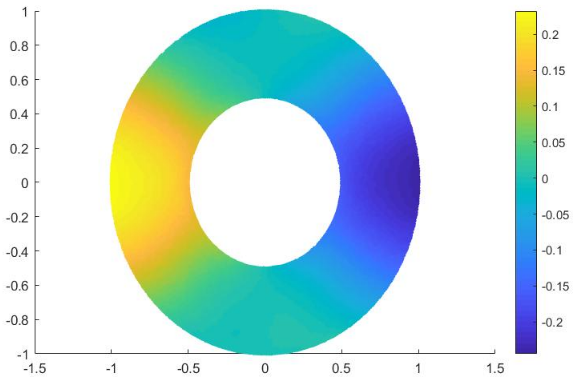


Fig19. Displacement in the x direction

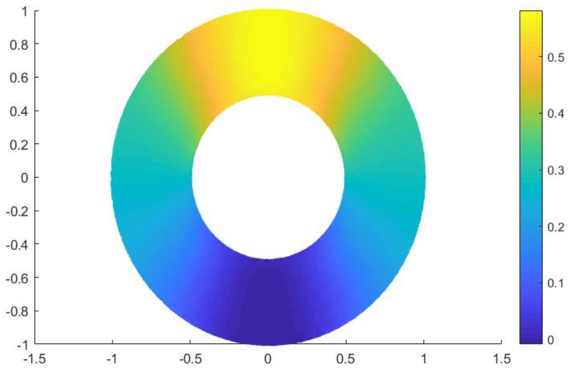


Fig20. Displacement in the y direction

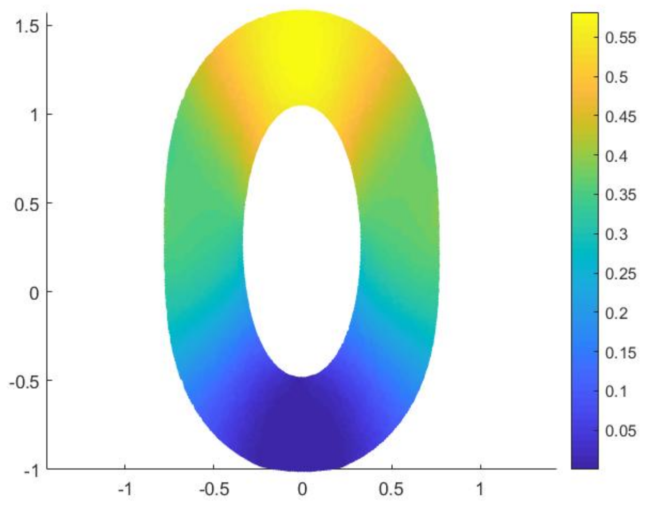


Fig21. Solution (displacement) applied
to the original domain

Figures 19 and 20 show respectively the x and the y component of the solution and the plots, together with the final result in figure 21, show something that we could expect: the pulling force imposed on the top part makes the wheel squeeze in the vertical direction. Note that the bottom part perfectly matches the Dirichlet boundary condition.

4. The elastoplastic problem

Until now we have considered the pure elastic problem, characterised by a linear relation $\sigma(u) = \mathcal{D}\varepsilon(u)$ with $\mathcal{D} : \mathbb{S} \to \mathbb{S}$ defined such that $\mathcal{D}\tau = 2\mu\tau + \lambda Tr(\tau)\mathbb{I}_3 \ \forall\tau \in \mathbb{S}$ (Hook's constitutive law). In this way we studied the elastic response of a material without considering the plastic one, which may occur in correspondence of specific loads or after a threshold is reached in terms of applied forces. In essence , plastic behaviour is characterised by an irreversible straining $\varepsilon^{pl}(u)$ which can only be sustained once a certain level of stress has been reached, while the elastic strain $\varepsilon^{el}(u)$ (which can still be obtained through the Hook's law) is still present but it goes back to the initial value after the unloading phase. In order to take into account both the elastic and the plastic response we need to correct the fourth order tensor $\mathcal{D}$, leading to a nonlinear relation between u and σ. From now on we call $\varepsilon(u) = \varepsilon^{el}(u) + \varepsilon^{pl}(u)$ the total strain.

In order to formulate a theory which models elastoplastic material deformation three requirements have to be met:

- An explicit relationship between stress and strain must be formulated to describe material behaviour under elastic conditions i.e. before the onset of plastic deformation (Hook's constitutive law)
- A yield criterion indicating the stress level at which plastic flow commences must be postulated
- A relationship between stress and strain must be developed for post-yield behaviour, i.e. when the deformation is made up of both elastic and plastic components

4.1. The Drucker-Prager yield criterion

The yield criterion determines the stress level at which plastic deformation begins and can be written in the general form

$$\Psi(\sigma) = 0$$

where Ψ is some function that can involve material parameters to be determined experimentally. Any yield criterion should be independent of the orientation of the coordinate system employed and therefore it should be a function of the three stress invariants only.

For our problem we adopted the Drucker-Prager yield criterion, which reads as follows:

$$\Psi(\sigma) = \sqrt{\frac{1}{2}}|\mathcal{I}_D\sigma| + \frac{\eta}{3}\mathbb{I}_3 : \sigma - c$$

where $\mathcal{I}_D = \mathcal{I} - \frac{1}{3}\mathbb{I}_3 \otimes \mathbb{I}_3$ and $\mathcal{I}\tau = \tau \ \forall\tau \in \mathbb{R}^{3\times3}_{sym}$, i.e. $\mathcal{I}_D\sigma$ is the deviatoric part of σ.

With the notation $|\tau|$ we indicate a norm of the tensor $\tau \in \mathbb{R}^{3\times 3}_{sym}$ (usually the L^2-norm) and η, c are material parameters that must be given.
Notice that the term $\mathbb{I}_3 : \sigma$ suggests that we are dealing with a 3-dimensional problem and this is actually the case: even if we want to study the elastoplastic response of a 2D body, we cannot ignore the strain and the stress that may occur in the third direction. However, differently from a real 3D problem, we can impose some constrains and in the following we are adopting the **plane-strain** approach:

- $\varepsilon^{el}_{xz} = \varepsilon^{el}_{yz} = 0$
- $\varepsilon^{pl}_{xz} = \varepsilon^{pl}_{yz} = 0$
- $\varepsilon_{zz} = \varepsilon^{el}_{zz} + \varepsilon^{pl}_{zz} = 0$

As a consequence $\sigma_{xz} = \sigma_{yz} = 0$ but, in general, $\sigma_{zz} \neq 0$
If we plot $\Psi(\sigma)$ in the space of the principal stresses we obtain a cone:

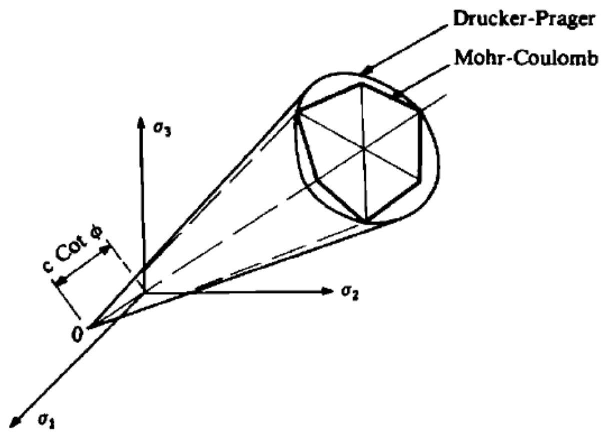


Fig2. Geometrical representation of the Mohr-Coulomb and DruckerPrager yield surfaces in principal stress space (This figure was taken from [2])

This cone in every section intersects the apices of the hexagon that represents the yield surface of another yield criterion: the Mohr-Coulomb yield criterion, a more complex version of the Drucker-Prager approach which includes some corrections. For a further discussion about it we refer to [2].

4.2. Time-discretization

Since the problem is no more linear ($\sigma(u) = \mathcal{D}\varepsilon(u)$ with $\mathcal{D} = \mathcal{D}(u)$) we cannot rely on the usual implementation, at least not before applying a suitable linearization.
The main idea is to divide the total load in an arbitrary number of steps, let's say N, and to use the solution u_i associated to the i-th load to compute the solution u_{i+1} associated to the following step using Newton algorithm. Note that, since we are using a discontinuous method (and so the Dirichlet boundary condition is imposed in a weak sense), the total load includes

both the external force f and all the boundary conditions, so it's represented by the functional F introduced in section 3.2.
We describe here more precisely the algorithm:

```
Result: $u_N$ (final displacement)
Input: N, F, MaxIt;
1. Set: $i = 0$;
while $i < N$    (loading phase)    do
    2. define $F_i = \frac{i}{N}F$;
    3. compute an initial guess $u_i^0$ for $u_i$ solution of $\mathcal{A}_\theta(u_i, v) = F_i v \quad \forall v \in W_h^{DG}$;
    4. Set: $k = 0$, it $= 0$
    while Not converging and it < MaxIt  (Newton algorithm)   do
        5. compute the constitutive law $\mathcal{D}\left(u_i^k\right)$;
        6. compute $u_i^{k+1}$ using the Newton update rule;
    end
end
```

Algorithm 1: Loading procedure with Newton algorithm

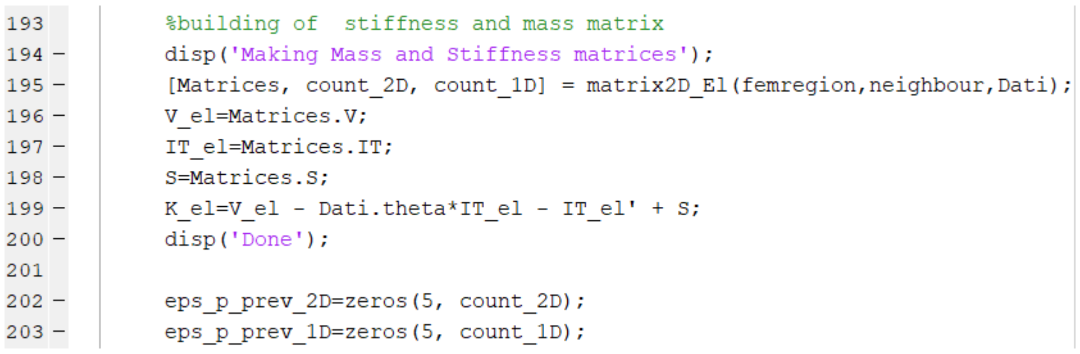
This procedure can be interpreted as a time discretization in N time steps, however the time derivative still doesn't enter in the equations, i.e. we are solving the quasi-static elastoplastic problem. A possible formulation is the following:

$$\begin{cases} -\nabla \cdot \sigma(u(x,y,t)) = f^T(x,y,t) & \text{in } \mathbb{R}^2 \times [0,T] \\ u \cdot n = u_{D,n}^T & \text{on } \Gamma_{D,n} \times [0,T] \\ u \cdot t = u_{D,t}^T & \text{on } \Gamma_{D,t} \times [0,T] \\ n^T \sigma(u)\, n = h_n^T & \text{on } \Gamma_{N,n} \times [0,T] \\ t^T \sigma(u)\, n = h_t^T & \text{on } \Gamma_{N,t} \times [0,T] \end{cases}$$

where $f^T(x,y,t) = \frac{t}{T} f(x,y)$, and similarly for $u_{D,n}^T$, $u_{D,t}^T$, h_n^T, and h_t^T.
We associate to this formulation a homogeneous time discretization $\{t_i\}_{i=0}^N$, $t_i = \frac{i}{N}T$ in order to be coherent with the algorithm described above.
Adopting this setting we have $u(x,y,t_i) = \left[\sum_{j=1}^{N_h} u_{ij}^x \Phi_j(x,y)\,,\ \sum_{j=1}^{N_h} u_{ij}^y \Phi_j(x,y)\right]$, so we define $\mathbf{u}_i = \left[u_{i1}^x, u_{i2}^x, \ldots, u_{iN_h}^x, u_{i1}^y, u_{i2}^y, \ldots, u_{iN_h}^y\right]^T$
The linear system is derived in the very same way but now, for $i = 1, \ldots, N$ we have to find $\mathbf{u_i}$ such that $\mathbf{K u_i} = \mathbf{f_i}$ where at every step i the constitutive law may vary.

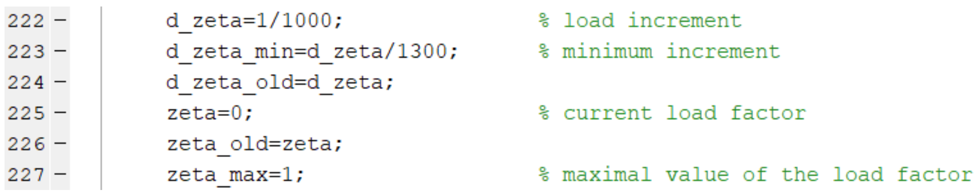
4.3. Code inspection

In this section we analyze the most important lines of the code in order to describe the implementation of the Newton algorithm.



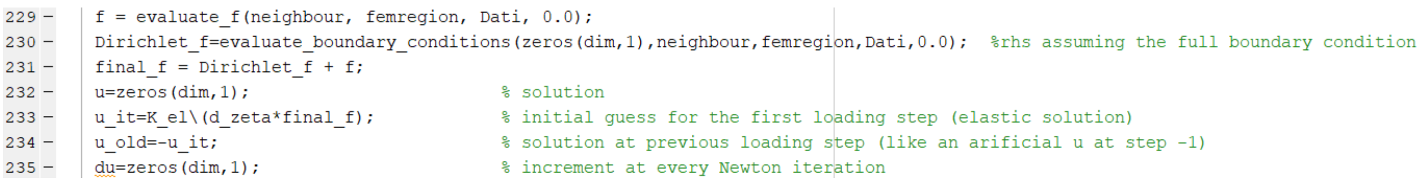
```
193       %building of  stiffness and mass matrix
194 -     disp('Making Mass and Stiffness matrices');
195 -     [Matrices, count_2D, count_1D] = matrix2D_El(femregion,neighbour,Dati);
196 -     V_el=Matrices.V;
197 -     IT_el=Matrices.IT;
198 -     S=Matrices.S;
199 -     K_el=V_el - Dati.theta*IT_el - IT_el' + S;
200 -     disp('Done');
201
202 -     eps_p_prev_2D=zeros(5, count_2D);
203 -     eps_p_prev_1D=zeros(5, count_1D);
```

Here we compute the matrices associated to the elastic regime. This is needed since as initial guess for the first iteration we choose the elastic response, which is close enough to the real solution so that the Newton algorithm can converge, moreover, as we will see later, the matrix **S** is reused also in case of plastic response.
In line 195 we get **count_2D** and **count_1D** which are, respectively, the total number of quadrature points for the 2D 1D integrations. We then define (lines 202 and 203) the plastic strain in every quadrature node, whose initial value is zero since the elastic response is considered as initial guess.



```
222 -     d_zeta=1/1000;              % load increment
223 -     d_zeta_min=d_zeta/1300;     % minimum increment
224 -     d_zeta_old=d_zeta;
225 -     zeta=0;                     % current load factor
226 -     zeta_old=zeta;
227 -     zeta_max=1;                 % maximal value of the load factor
```

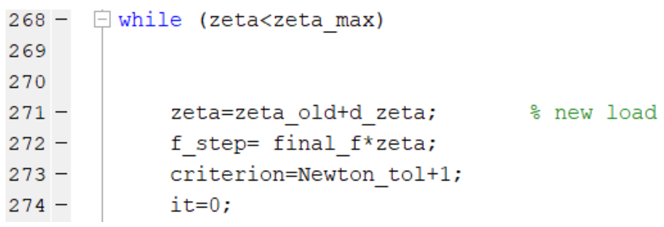
From this figure we can see the setting for the loading phase, divided in $N = 1000$ steps. At line 222 we define **d_zeta** as the fraction of the load increment applied in each step and (line225) **zeta** is the fraction of the total load currently used, so it's initially set to zero and it will increase until the value of 1 (**zeta_max**). The role of the other variables will be clear later.



```
229 -   f = evaluate_f(neighbour, femregion, Dati, 0.0);
230 -   Dirichlet_f=evaluate_boundary_conditions(zeros(dim,1),neighbour,femregion,Dati,0.0);  %rhs assuming the full boundary condition
231 -   final_f = Dirichlet_f + f;
232 -   u=zeros(dim,1);                       % solution
233 -   u_it=K_el\(d_zeta*final_f);           % initial guess for the first loading step (elastic solution)
234 -   u_old=-u_it;                          % solution at previous loading step (like an arificial u at step -1)
235 -   du=zeros(dim,1);                      % increment at every Newton iteration
```

At line 229 we compute the contribution of the right-hand side given by the full external force f and the full Neumann boundary condition. We complete the right-hand side at line 231, adding the contribution given by the full Dirichlet boundary condition.

At line 233 we compute the elastic response associated to the first load, indeed by linearity it's enough to multiply by **d_zeta** the right-hand side of the original problem. Moreover, at line 235, we initialize to zero the increment of the displacement, that will be useful for the Newton algorithm.



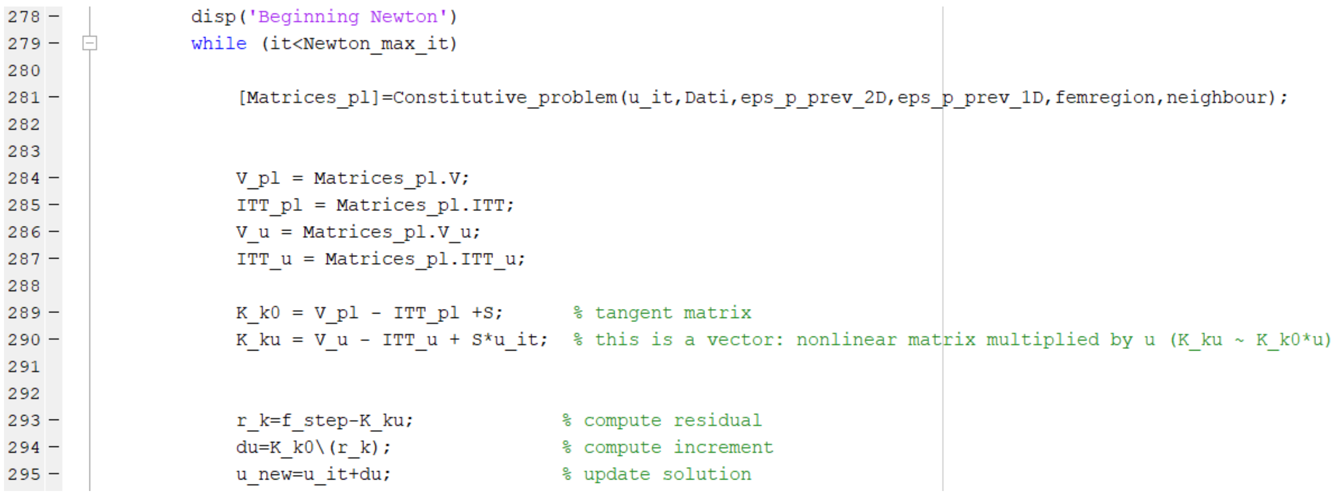
```
268 -  while (zeta<zeta_max)
269
270
271 -      zeta=zeta_old+d_zeta;       % new load
272 -      f_step= final_f*zeta;
273 -      criterion=Newton_tol+1;
274 -      it=0;
```

Here we update the loading fraction and the right-hand side before entering in the Newton algorithm. In particular at line 272 **f_step** represents what we defined as $\mathbf{f_i}$.

Now we enter in the most important section of the code: the Newton algorithm.
We recall that, in order to apply the Newton procedure, given the entering solution u_i^k (here represented by **u_it**) we need to compute both the real stress $T_k = \sigma\left(u_i^k\right) = \mathcal{D}\left(u_i^k\right)\varepsilon\left(u_i^k\right)$ and the linearization centered at u_i^k, represented by a fourth order tensor that we will call T_k^0 applied to $\varepsilon\left(u_i^k\right)$ In practice we must end up with two quantities:

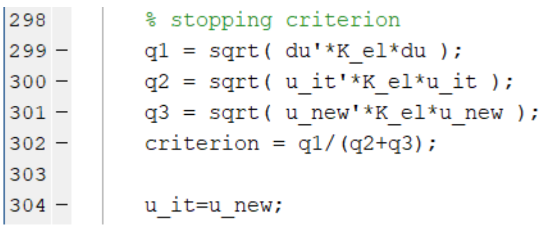
- the tangent matrix $\mathbf{K_k^0}$, which is an analogous of the usual stiffness matrix but imposing $\sigma(\tau) = T_k^0\varepsilon(\tau)$.
- the vector $\mathbf{K_{ku}}$, computed in the same way as $\mathbf{K_k^0}$ but now we impose $\sigma\left(u_i^k\right) = T_k$. Note that this time we use the whole solution u_i^k since, because of non-linearity of σ w.r.t. u_i^k, we cannot rely on the decomposition of u_i^k through the basis functions and this leads immediately to two consequences:
    1. $\mathbf{K_{ku}}$ is a vector since the basis functions are used only as test functions
    2. since for any test function Φ we cannot evaluate exact value of $\sigma(\Phi)$ in case of plasticity, we set $\theta = 0$ in order to remove this term from the problem.

Note that, if we find an elastic response on every quadrature node then $T_k^0 = \mathcal{D}^{el}$, so we get the Hook's constitutive law and $\mathbf{K_k^0} = \mathbf{K_{el}}$ (line 199), moreover it is possible to prove also that $\mathbf{K_{ku}} = \mathbf{K_k^0}u_i^k = \mathbf{K_{el}}u_i^k$. This is reasonable since for a complete elastic response we are dealing with a linear problem, so the linearized term and the exact one coincide.



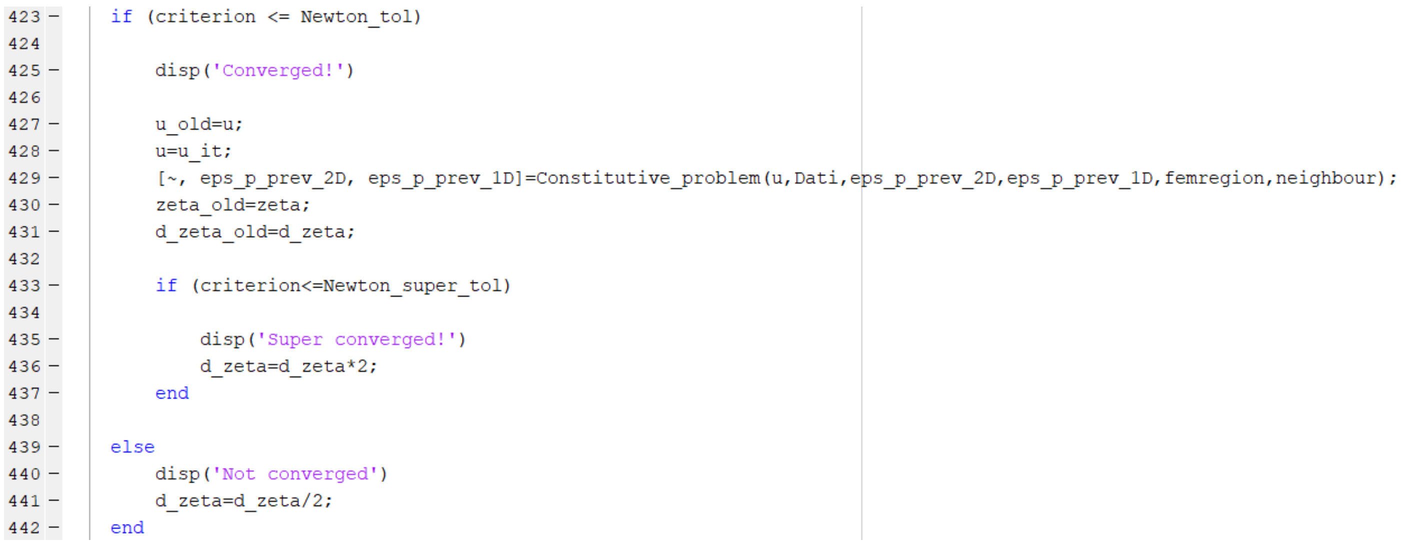
```
278 -          disp('Beginning Newton')
279 -          while (it<Newton_max_it)
280
281 -              [Matrices_pl]=Constitutive_problem(u_it,Dati,eps_p_prev_2D,eps_p_prev_1D,femregion,neighbour);
282
283
284 -              V_pl = Matrices_pl.V;
285 -              ITT_pl = Matrices_pl.ITT;
286 -              V_u = Matrices_pl.V_u;
287 -              ITT_u = Matrices_pl.ITT_u;
288
289 -              K_k0 = V_pl - ITT_pl +S;      % tangent matrix
290 -              K_ku = V_u - ITT_u + S*u_it;  % this is a vector: nonlinear matrix multiplied by u (K_ku ~ K_k0*u)
291
292
293 -              r_k=f_step-K_ku;             % compute residual
294 -              du=K_k0\(r_k);               % compute increment
295 -              u_new=u_it+du;               % update solution
```

At line 281 we compute the plastic versions of the same matrices computed above. Note that **S** doesn't change between elastic and plastic response but we could expect this since it represents the only term that doesn't involve any stress, so no constitutive law enters in its computation. The function **Constitutive_problem** is an extended version of **Matrix2D_El** (line 195) that will be discussed in the next section. $\mathbf{K_k^0}$ and $\mathbf{K_{ku}}$ are assembled at lines 289 and 290.
At lines 293-295 we compute the residual, the increment and we update the solution.



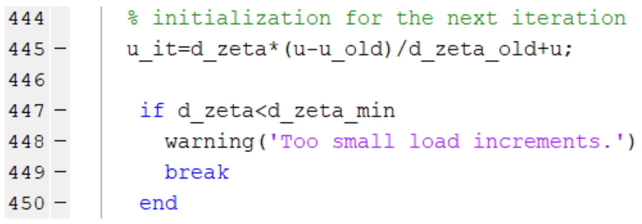
```
298        % stopping criterion
299 -      q1 = sqrt( du'*K_el*du );
300 -      q2 = sqrt( u_it'*K_el*u_it );
301 -      q3 = sqrt( u_new'*K_el*u_new );
302 -      criterion = q1/(q2+q3);
303
304 -      u_it=u_new;
```

At lines 299-302 a residual based on the increment is computed.



```
423 -    if (criterion <= Newton_tol)
424
425 -        disp('Converged!')
426
427 -        u_old=u;
428 -        u=u_it;
429 -        [~, eps_p_prev_2D, eps_p_prev_1D]=Constitutive_problem(u,Dati,eps_p_prev_2D,eps_p_prev_1D,femregion,neighbour);
430 -        zeta_old=zeta;
431 -        d_zeta_old=d_zeta;
432
433 -        if (criterion<=Newton_super_tol)
434
435 -            disp('Super converged!')
436 -            d_zeta=d_zeta*2;
437 -        end
438
439 -    else
440 -        disp('Not converged')
441 -        d_zeta=d_zeta/2;
442 -    end
```

If convergence is reached, at line 428 we update the real solution and, in the following line, we call again **Constitutive_problem** but with the only purpose of updating the plastic response in the quadrature nodes.
If the criterion based on the increment is much lower than the chosen tolerance we talk about "super convergence" and in this case, in the following load step, we double the load increment. Conversly, if convergence is not reached, the solution is not updated and in the following load step we halve the load increment.



```
444       % initialization for the next iteration
445 -     u_it=d_zeta*(u-u_old)/d_zeta_old+u;
446
447 -      if d_zeta<d_zeta_min
448 -        warning('Too small load increments.')
449 -        break
450 -      end
```

Line 445 computes the initial guess for the next load step. This initialization can be written also in the following way:
let $\delta U = U - U\_old$ (difference between the solution at the current and at the previous time step), then U_it= U+$\alpha * \delta$U where

$$\begin{cases} \alpha = \frac{1}{2} & \text{if convergence was not achieved} \\ \alpha = 1 & \text{if convergence was achieved but not super-convergence} \\ \alpha = 2 & \text{if super-convergence was achieved} \end{cases}$$

The idea behind this is a sort of interpolation that takes into account the evolution of the load increment.
Lines 447-450 are necessary in order to stop the algorithm in case the method is not converging.

4.4. An analytical solution for the Drucker-Prager yield criterion

We now address the problem of deriving a constitutive law for the elastoplastic problem given a specific displacement u, so, referring to the code, we now discuss the function **Constitutive_problem**, in particular we want to find the correct form of $T_k = \sigma(u)$ and the tangent fourth orther tensor T_k^0 for a general case of elastoplastic response.
This is a nontrivial problem consisting in a system of ODE whose solution is usually very complex and irregular.
However, luckily, for our case the operators T_k and T_k^0 can be found in closed forms (for more details on that topic we refer to [7] and [6]). To summarize their forms, we distinguish three different cases and define the following auxiliary notation:

- $\sigma_k^{tr} = \mathcal{D}^{el} \left(\varepsilon_k (u) - \varepsilon_k^{el} (u) \right)$
- $p_k^{tr} = \mathbb{I}_3 : \sigma_k^{tr} = Tr \left(\sigma_k^{tr} \right)$
- $s_k^{tr} = \mathcal{I}_D \varepsilon_k (u)$
- $\rho_k^{tr} = 2\mu || \varepsilon_k (u) ||_{L^2}$
- $n_k^{tr} = \frac{s_k^{tr}}{||\varepsilon_k (u)||_{L^2}}$

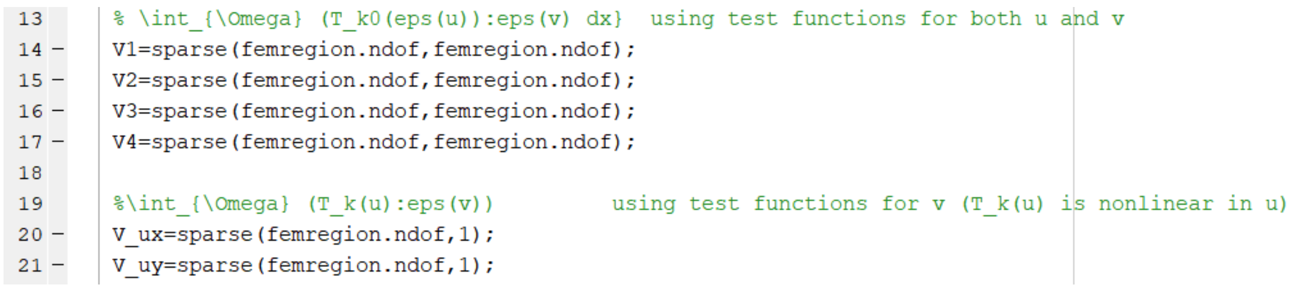
From now on we refer to the quantities just introduced as "trials" and the index k represents the k-th iteration of the Newton algorithm in a generic load step. Moreover we introduce the bulk modulus $K = \lambda + \frac{2\mu}{3}$

1. Elastic response occurs if $\Psi \left(\sigma_k^{tr} \right) \leqslant 0$. Then:
   - $T_k = \sigma_k^{tr}$
   - $T_k^0 = \mathcal{D}^{el}$
   - $\varepsilon_k^{pl} (u) = \varepsilon_{k-1}^{pl} (u)$
2. Return to the smooth portion of the yield surface occurs if $\Psi \left(\sigma_k^{tr} \right) > 0$ and $\eta \frac{p_k^{tr}}{3} - \frac{K\eta^2}{\mu\sqrt{2}} \rho_k^{tr} - c < 0$. Then:
   - $T_k = \sigma_k^{tr} - \frac{\Psi\left(\sigma_k^{tr}\right)}{\mu + K\eta^2} \left(\sqrt{2}\mu n_k^{tr} + K\eta \mathbb{I}_3 \right)$
   - $T_k^0 = \mathcal{D}^{el} - \frac{1}{\mu + K\eta^2} \left(\frac{2\sqrt{2}\mu^2 \Psi\left(\sigma_k^{tr}\right)}{\rho_k^{tr}} \left(\mathcal{I}_D - n_k^{tr} \otimes n_k^{tr} \right) + \left(\sqrt{2}\mu n_k^{tr} + K\eta \mathbb{I}_3 \right) \otimes \left(\sqrt{2}\mu n_k^{tr} + K\eta \mathbb{I}_3 \right) \right)$
   - $\varepsilon_k^{pl} (u) = \varepsilon_{k-1}^{pl} (u) + \frac{\Psi\left(\sigma_k^{tr}\right)}{\mu + K\eta^2} \left(\frac{n_k^{tr}}{\sqrt{2}} + \frac{\eta}{3} \mathbb{I}_3 \right)$
3. Return to the apex of the yield surface occurs if $\Psi \left(\sigma_k^{tr} \right) > 0$ and $\eta \frac{p_k^{tr}}{3} - \frac{K\eta^2}{\mu\sqrt{2}} \rho_k^{tr} - c \geqslant 0$. Then:
   - $T_k = \frac{c}{\eta} \mathbb{I}_3$
   - $T_k^0 = \mathbb{O}$
   - $\varepsilon_k^{pl} (u) = \varepsilon_{k-1}^{pl} (u) - \frac{c}{3K\eta} \mathbb{I}_3$

     Where $\mathbb{O}$ is the zero fourth order tensor, i.e. $\mathbb{O}\tau : \xi = 0 \ \forall \tau, \xi \in \mathbb{R}_{sym}^{3\times 3}$

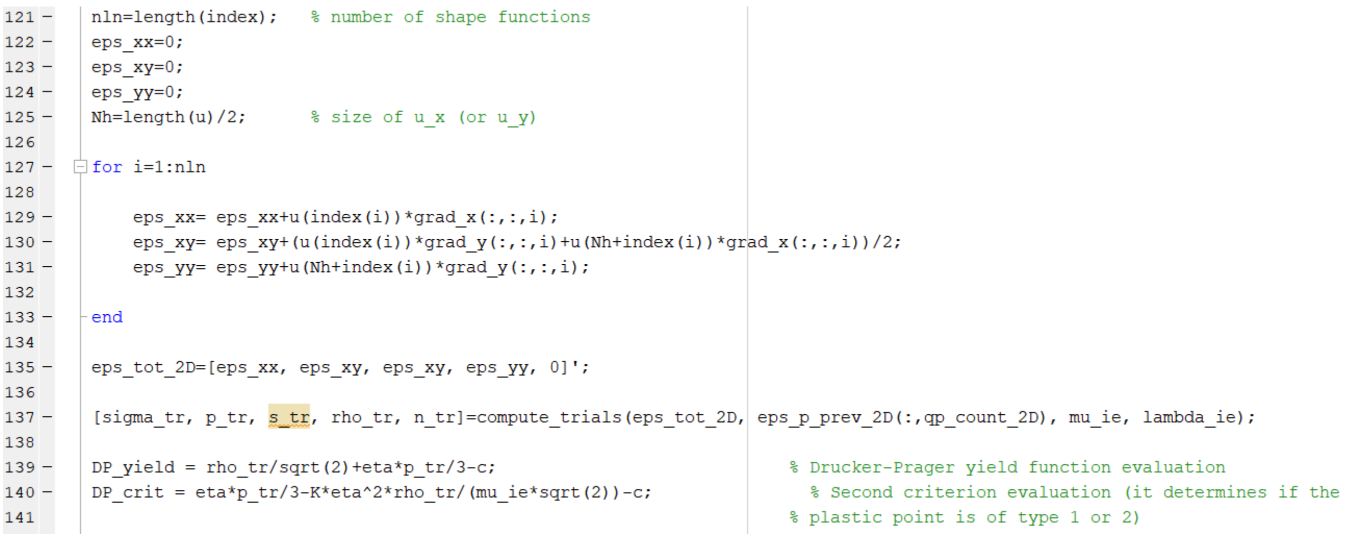
For more details see [4]
We now enter into the details of **Constitutive_problem** to see how to manage the implementation of such solution in all the three possible cases.

4.4.1. Implementation of the integral on the surface



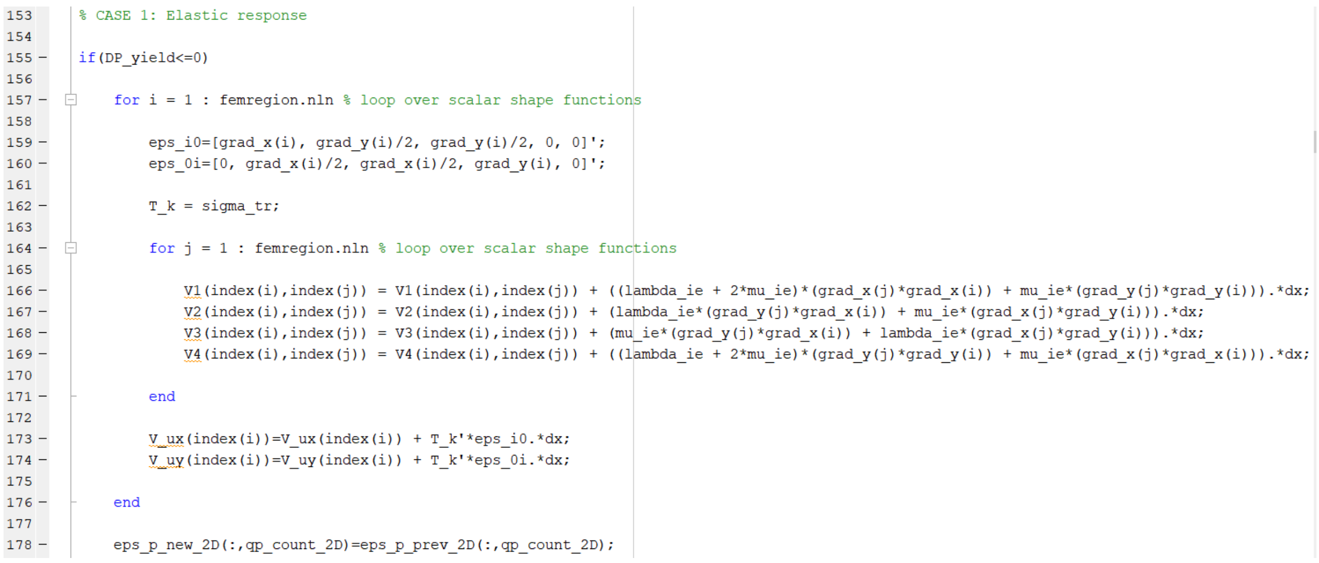
```
13     % \int_{\Omega} (T_k0(eps(u)):eps(v) dx}  using test functions for both u and v
14 -   V1=sparse(femregion.ndof,femregion.ndof);
15 -   V2=sparse(femregion.ndof,femregion.ndof);
16 -   V3=sparse(femregion.ndof,femregion.ndof);
17 -   V4=sparse(femregion.ndof,femregion.ndof);
18
19     %\int_{\Omega} (T_k(u):eps(v))         using test functions for v (T_k(u) is nonlinear in u)
20 -   V_ux=sparse(femregion.ndof,1);
21 -   V_uy=sparse(femregion.ndof,1);
```

In the following we analyze the integral $\sum_{i=1}^{N}\int_{K_i} T_k : \varepsilon\left(v\right) ds$, represented by the vector $[\mathbf{V\_ux}, \mathbf{V\_uy}]^T$ (lines 20-21), and it's linearized form $\sum_{i=1}^{N}\int_{K_i} T_k^0 \varepsilon\left(u\right) : \varepsilon\left(v\right) ds$, represented by the matrix $\mathbf{V} = \begin{bmatrix} \mathbf{V1} & \mathbf{V2} \\ \mathbf{V3} & \mathbf{V4} \end{bmatrix}$ (lines 14-17)



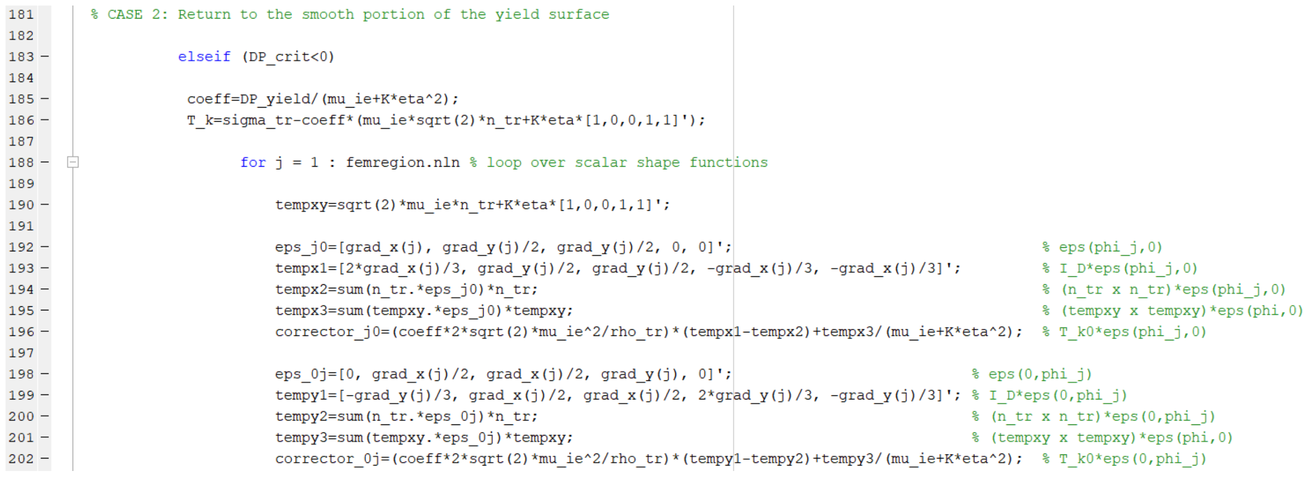
```
121 -   nln=length(index);    % number of shape functions
122 -   eps_xx=0;
123 -   eps_xy=0;
124 -   eps_yy=0;
125 -   Nh=length(u)/2;       % size of u_x (or u_y)
126
127 -   for i=1:nln
128
129 -       eps_xx= eps_xx+u(index(i))*grad_x(:,:,i);
130 -       eps_xy= eps_xy+(u(index(i))*grad_y(:,:,i)+u(Nh+index(i))*grad_x(:,:,i))/2;
131 -       eps_yy= eps_yy+u(Nh+index(i))*grad_y(:,:,i);
132
133 -   end
134
135 -   eps_tot_2D=[eps_xx, eps_xy, eps_xy, eps_yy, 0]';
136
137 -   [sigma_tr, p_tr, s_tr, rho_tr, n_tr]=compute_trials(eps_tot_2D, eps_p_prev_2D(:,qp_count_2D), mu_ie, lambda_ie);
138
139 -   DP_yield = rho_tr/sqrt(2)+eta*p_tr/3-c;                   % Drucker-Prager yield function evaluation
140 -   DP_crit = eta*p_tr/3-K*eta^2*rho_tr/(mu_ie*sqrt(2))-c;     % Second criterion evaluation (it determines if the
141                                                               % plastic point is of type 1 or 2)
```

First we need to compute the trials and for this purpose we need the total strain of the current solution, computed at lines 127-135 and then we rely on the function **Compute_trials**.
Note that the strain matrix is reorganized in a vectorial form in the way described at line 135. In this code every matrix is managed in such way, so, for instance, the identity matrix is represented by the vector $[1, 0, 0, 1, 1]^T$.
At line 139 $\Psi\left(u\right)$ is computed and, at line 140 , we define and evaluate **DP_crit**, which is the discriminant between the two types of plastic response.



```
153  % CASE 1: Elastic response
154
155  if(DP_yield<=0)
156
157      for i = 1 : femregion.nln % loop over scalar shape functions
158
159          eps_i0=[grad_x(i), grad_y(i)/2, grad_y(i)/2, 0, 0]';
160          eps_0i=[0, grad_x(i)/2, grad_x(i)/2, grad_y(i), 0]';
161
162          T_k = sigma_tr;
163
164          for j = 1 : femregion.nln % loop over scalar shape functions
165
166              V1(index(i),index(j)) = V1(index(i),index(j)) + ((lambda_ie + 2*mu_ie)*(grad_x(j)*grad_x(i)) + mu_ie*(grad_y(j)*grad_y(i))).*dx;
167              V2(index(i),index(j)) = V2(index(i),index(j)) + (lambda_ie*(grad_y(j)*grad_x(i)) + mu_ie*(grad_x(j)*grad_y(i))).*dx;
168              V3(index(i),index(j)) = V3(index(i),index(j)) + (mu_ie*(grad_y(j)*grad_x(i)) + lambda_ie*(grad_x(j)*grad_y(i))).*dx;
169              V4(index(i),index(j)) = V4(index(i),index(j)) + ((lambda_ie + 2*mu_ie)*(grad_y(j)*grad_y(i)) + mu_ie*(grad_x(j)*grad_x(i))).*dx;
170
171          end
172
173          V_ux(index(i))=V_ux(index(i)) + T_k'*eps_i0.*dx;
174          V_uy(index(i))=V_uy(index(i)) + T_k'*eps_0i.*dx;
175
176      end
177
178      eps_p_new_2D(:,qp_count_2D)=eps_p_prev_2D(:,qp_count_2D);
```

In case of elastic response the matrix **V** is assembled in the very same way as in **Matrix2D_El**, relying on the Hook's constitutive law (lines 166-169).
At lines 159 and 169 we find respectively $\varepsilon\left([\Phi_i, 0]\right)$ and $\varepsilon\left([0, \Phi_i]\right)$.
At line 162 T_k is assembled, lines 173 and 174 compute the contribution of the non-linearized integral and line 178 updates the plastic strain on the current quadrature node.

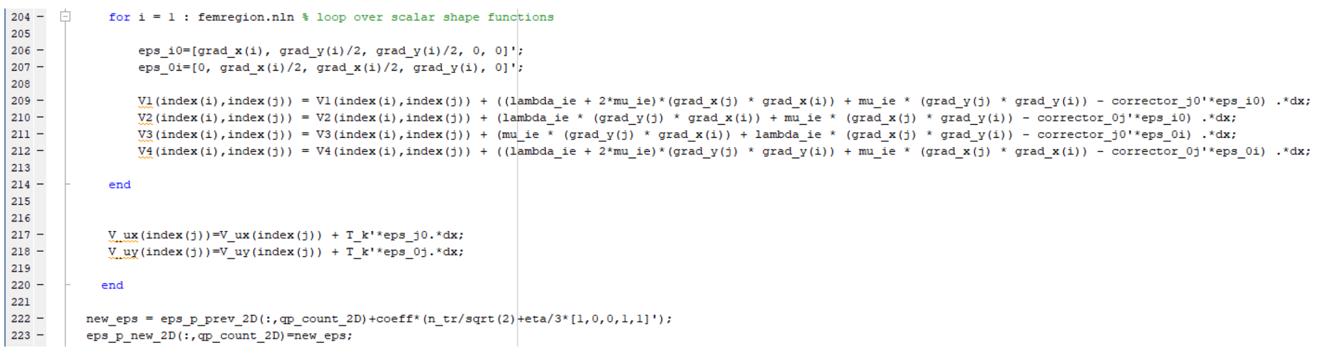


```
181  % CASE 2: Return to the smooth portion of the yield surface
182
183              elseif (DP_crit<0)
184
185               coeff=DP_yield/(mu_ie+K*eta^2);
186               T_k=sigma_tr-coeff*(mu_ie*sqrt(2)*n_tr+K*eta*[1,0,0,1,1]');
187
188                  for j = 1 : femregion.nln % loop over scalar shape functions
189
190                      tempxy=sqrt(2)*mu_ie*n_tr+K*eta*[1,0,0,1,1]';
191
192                      eps_j0=[grad_x(j), grad_y(j)/2, grad_y(j)/2, 0, 0]';                                  % eps(phi_j,0)
193                      tempx1=[2*grad_x(j)/3, grad_y(j)/2, grad_y(j)/2, -grad_x(j)/3, -grad_x(j)/3]';         % I_D*eps(phi_j,0)
194                      tempx2=sum(n_tr.*eps_j0)*n_tr;                                                        % (n_tr x n_tr)*eps(phi_j,0)
195                      tempx3=sum(tempxy.*eps_j0)*tempxy;                                                    % (tempxy x tempxy)*eps(phi,0)
196                      corrector_j0=(coeff*2*sqrt(2)*mu_ie^2/rho_tr)*(tempx1-tempx2)+tempx3/(mu_ie+K*eta^2);  % T_k0*eps(phi_j,0)
197
198                      eps_0j=[0, grad_x(j)/2, grad_x(j)/2, grad_y(j), 0]';                              % eps(0,phi_j)
199                      tempy1=[-grad_y(j)/3, grad_x(j)/2, grad_x(j)/2, 2*grad_y(j)/3, -grad_y(j)/3]'; % I_D*eps(0,phi_j)
200                      tempy2=sum(n_tr.*eps_0j)*n_tr;                                                    % (n_tr x n_tr)*eps(0,phi_j)
201                      tempy3=sum(tempxy.*eps_0j)*tempxy;                                                % (tempxy x tempxy)*eps(phi,0)
202                      corrector_0j=(coeff*2*sqrt(2)*mu_ie^2/rho_tr)*(tempy1-tempy2)+tempy3/(mu_ie+K*eta^2);  % T_k0*eps(0,phi_j)
```

The second case is for sure the most delicate and it needs some preprocess. In this case, to simplify the computation, we used the following identity:

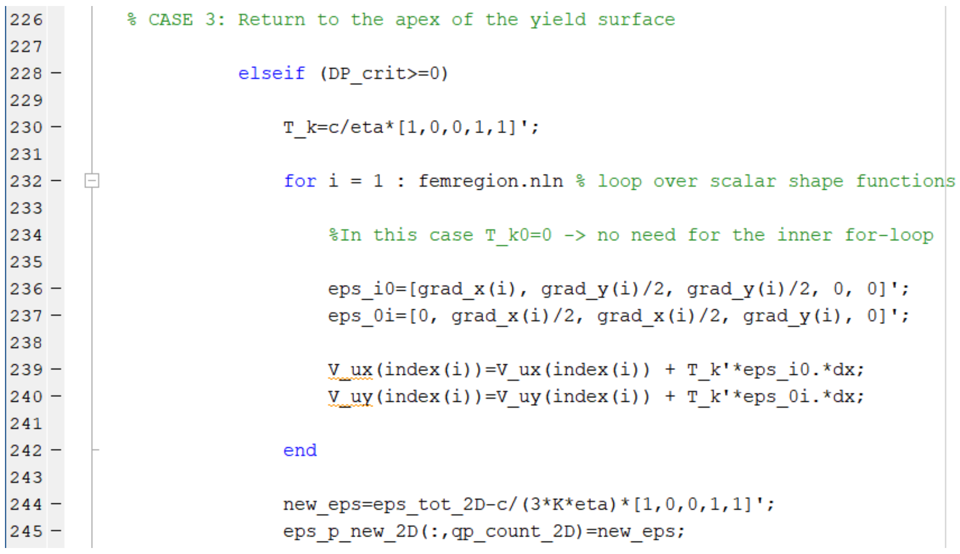
Proposition 4.1 *Let* $A, B \in \mathbb{R}^{n\times n}$, *then* $(A \otimes A)\, B = (A : B)\, A$

Such identity allows us to avoid dealing directly with fourth order tensors.
Lines 192-202 implement step by step the correction that we have to apply to the Hook's constitutive law in order to get the law described above for the case 2. Lines 192-196 do it for a generic basis function in the form $[\phi_i, 0]$ and lines 198-202 do it for the other type of basis functions, in particular at line 196 **corrector_j0** represents the quantity $\left(\mathcal{D}^{el} - T_k^0\right)\varepsilon\left([\Phi_j, 0]\right)$ while, at line 202, **corrector_0j** represents the quantity $\left(\mathcal{D}^{el} - T_k^0\right)\varepsilon\left([0, \Phi_j]\right)$



```
        for i = 1 : femregion.nln % loop over scalar shape functions

            eps_i0=[grad_x(i), grad_y(i)/2, grad_y(i)/2, 0, 0]';
            eps_0i=[0, grad_x(i)/2, grad_x(i)/2, grad_y(i), 0]';

            V1(index(i),index(j)) = V1(index(i),index(j)) + ((lambda_ie + 2*mu_ie)*(grad_x(j) * grad_x(i)) + mu_ie * (grad_y(j) * grad_y(i)) - corrector_j0'*eps_i0) .*dx;
            V2(index(i),index(j)) = V2(index(i),index(j)) + (lambda_ie * (grad_y(j) * grad_x(i)) + mu_ie * (grad_x(j) * grad_y(i)) - corrector_0j'*eps_i0) .*dx;
            V3(index(i),index(j)) = V3(index(i),index(j)) + (mu_ie * (grad_y(j) * grad_x(i)) + lambda_ie * (grad_x(j) * grad_y(i)) - corrector_j0'*eps_0i) .*dx;
            V4(index(i),index(j)) = V4(index(i),index(j)) + ((lambda_ie + 2*mu_ie)*(grad_y(j) * grad_y(i)) + mu_ie * (grad_x(j) * grad_x(i)) - corrector_0j'*eps_0i) .*dx;

        end


        V_ux(index(j))=V_ux(index(j)) + T_k'*eps_j0.*dx;
        V_uy(index(j))=V_uy(index(j)) + T_k'*eps_0j.*dx;

       end

    new_eps = eps_p_prev_2D(:,qp_count_2D)+coeff*(n_tr/sqrt(2)+eta/3*[1,0,0,1,1]');
    eps_p_new_2D(:,qp_count_2D)=new_eps;
```

Note that in this case the matrix **V** contains a correction w.r.t. the previous case, given by the term **corrector*eps**

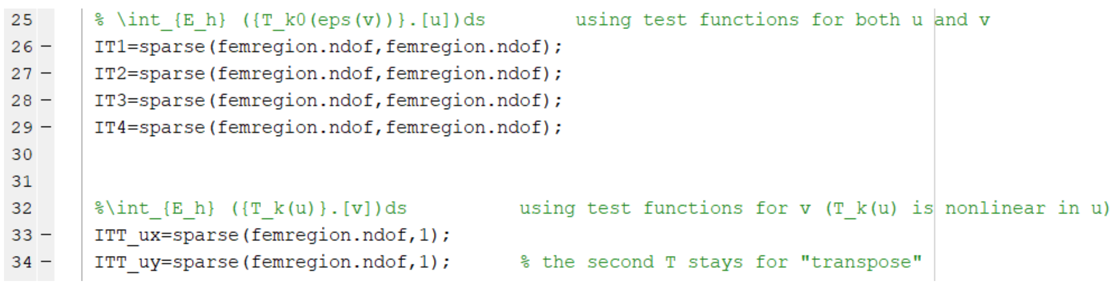


```
% CASE 3: Return to the apex of the yield surface

          elseif (DP_crit>=0)

              T_k=c/eta*[1,0,0,1,1]';

              for i = 1 : femregion.nln % loop over scalar shape functions

                  %In this case T_k0=0 -> no need for the inner for-loop

                  eps_i0=[grad_x(i), grad_y(i)/2, grad_y(i)/2, 0, 0]';
                  eps_0i=[0, grad_x(i)/2, grad_x(i)/2, grad_y(i), 0]';

                  V_ux(index(i))=V_ux(index(i)) + T_k'*eps_i0.*dx;
                  V_uy(index(i))=V_uy(index(i)) + T_k'*eps_0i.*dx;

              end

              new_eps=eps_tot_2D-c/(3*K*eta)*[1,0,0,1,1]';
              eps_p_new_2D(:,qp_count_2D)=new_eps;
```

In this last lines we finally see the implementation of the third case. Note that there is no contribution for the matrix **V** since in this last case $T_k^0 = \mathbb{O}$.

4.4.2. Implementation of the integral on the interfaces

This is the most tricky part and it requires some analytical considerations first.



```
25    % \int_{E_h} ({T_k0(eps(v))}.[u])ds        using test functions for both u and v
26 -  IT1=sparse(femregion.ndof,femregion.ndof);
27 -  IT2=sparse(femregion.ndof,femregion.ndof);
28 -  IT3=sparse(femregion.ndof,femregion.ndof);
29 -  IT4=sparse(femregion.ndof,femregion.ndof);
30
31
32    %\int_{E_h} ({T_k(u)}.[v])ds         using test functions for v (T_k(u) is nonlinear in u)
33 -  ITT_ux=sparse(femregion.ndof,1);
34 -  ITT_uy=sparse(femregion.ndof,1);       % the second T stays for "transpose"
```

with complete analogy to the previous case we have to compute both $\sum_{F\in\mathcal{F}_h^{i,D}}\int_F \{\{T_k^0\varepsilon(u)\}\} : [[v]]\, dl$, represented by $\mathbf{IT}^T$ (lines 26-29) and $\sum_{F\in\mathcal{F}_h^{i,D}}\int_F \{\{T_k\}\} : [[v]]\, dl$, represented by the vector **ITT_u** (lines 33-34).

First notice that, instead of assembling the matrix of the real linearization, we compute its transpose, this is done in order to exploit as much as possible the implementation of the same term in the elastic case, moreover note that, thanks to linearity, **IT** is actually the transpose of what we are looking for.
Linearity doesn't hold for the second integral, so in this case we have to directly compute the real quantity and the second "**T**" in **ITT_u**, which stands for "transpose", highlights this fact.
We now make some analytical considerations: consider an interface Γ shared by two elements K^+, K^- and suppose that we are inspecting Γ as an element of the set of edges of K^+. Take two functions $f, g \in W_h^{DG}$ and consider the quantity

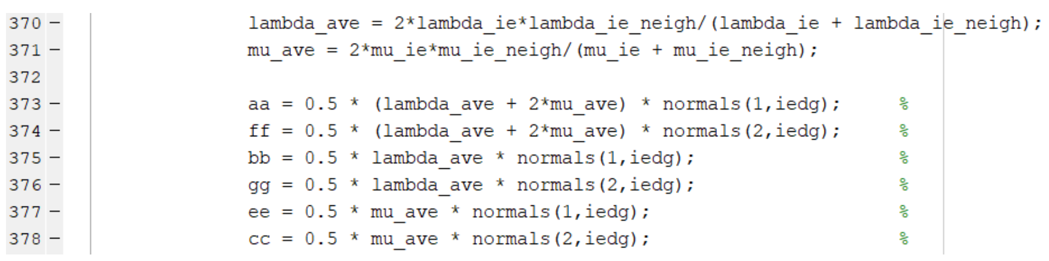
$$\int_\Gamma \{\{\sigma(f)\}\} : [[g]]\, dl = \underbrace{\frac{1}{2}\int_\Gamma \sigma\left(f^+\right) : [[g]]\, dl}_{\text{A}} + \underbrace{\frac{1}{2}\int_\Gamma \sigma\left(f^-\right) : [[g]]\, dl}_{\text{B}}$$

Since $\Gamma \subset \partial K^+$ and $\Gamma \subset \partial K^-$, the interface is visited twice and so we can compute just A, while B will be computed when the code inspects again Γ as an edge of K^- and this is done automatically by symmetry (f^+ and f^- swap the roles).
Similarly we can write

$$\text{A} \; = \underbrace{\frac{1}{2}\int_\Gamma \sigma\left(f^+\right) : \left(g^+ \otimes n^+\right) dl}_{\text{A}^+} + \underbrace{\frac{1}{2}\int_\Gamma \sigma\left(f^+\right) : \left(g^- \otimes n^-\right) dl}_{\text{A}^-}$$

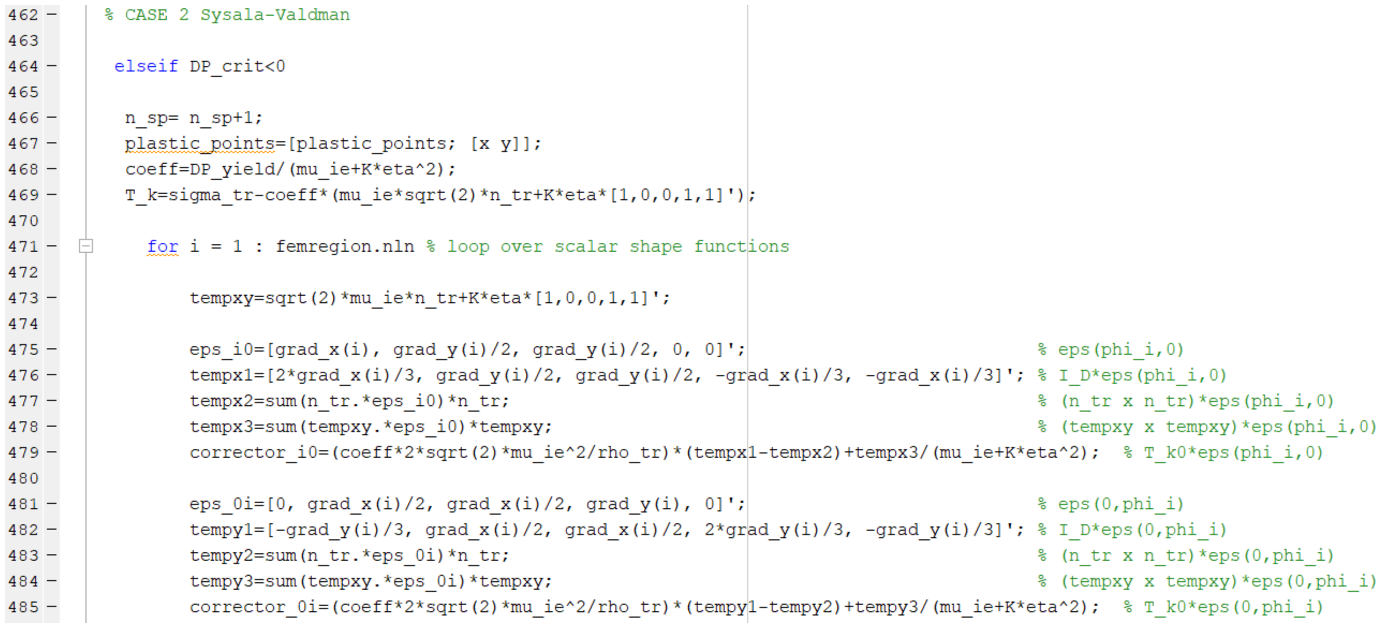
Note that, when we are looping on the edges of K^+, we know the global indexes of the basis functions associated to K^+, so, since in the code g^- is replaced by a test function associated to K^-, the contribution of A^- doesn't enter directly in the matrix **IT** and in the vector **ITT_u** but it is stored in a third-order tensor **ITN** and in a second-order tensor **ITTN_u** (the **N** stands for "neighbour") and it is assembled trough two special functions that we will see later.

We are now ready to see in details the implementation.


```
370 -            lambda_ave = 2*lambda_ie*lambda_ie_neigh/(lambda_ie + lambda_ie_neigh);
371 -            mu_ave = 2*mu_ie*mu_ie_neigh/(mu_ie + mu_ie_neigh);
372
373 -            aa = 0.5 * (lambda_ave + 2*mu_ave) * normals(1,iedg);     %
374 -            ff = 0.5 * (lambda_ave + 2*mu_ave) * normals(2,iedg);     %
375 -            bb = 0.5 * lambda_ave * normals(1,iedg);                  %
376 -            gg = 0.5 * lambda_ave * normals(2,iedg);                  %
377 -            ee = 0.5 * mu_ave * normals(1,iedg);                      %
378 -            cc = 0.5 * mu_ave * normals(2,iedg);                      %
```

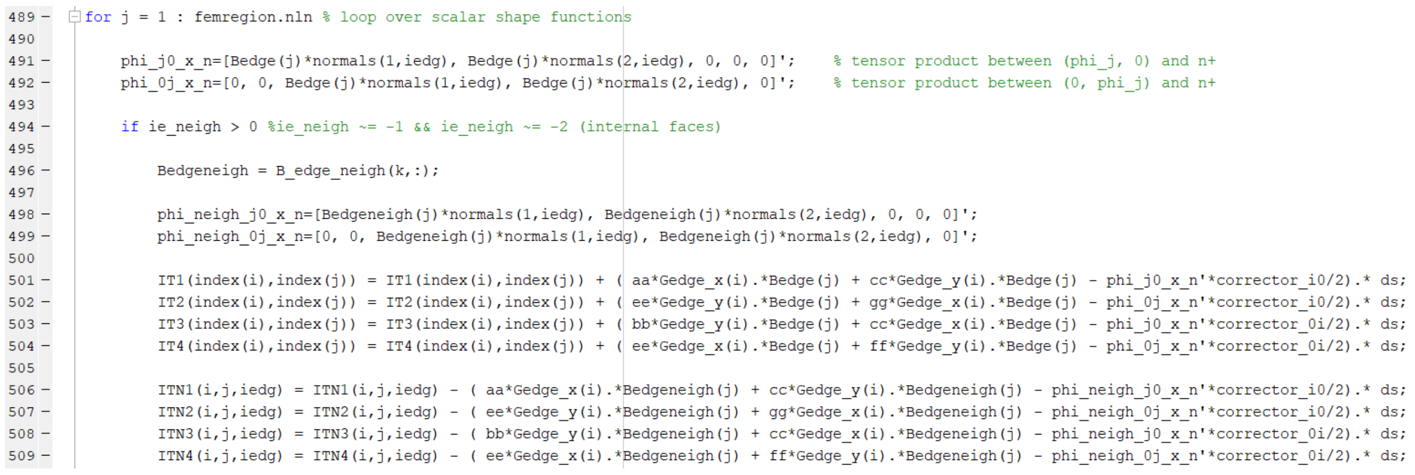
Since an interface can be shared by two elements, we need to average all the parameters using the values associated with both elements (lines 370-371) (if the interface belongs to the boundary then simply set **lambda_ie_neigh = lambda_ie** and **mu_ie_neigh = mu_ie**).
Lines 373-378 define some coefficients useful to compute the quantity $\sigma(f) : (g \otimes n)$ when f and g are basis functions.

Since the analytical solution for the Drucker-Prager yield criterion is still valid on an interface the subdivision in the three cases and the corresponding values of T_k and T_k^0 are the same as the ones shown above. We then inspect just one of the three cases to show how to manage the splitting of the contributions discussed above and we choose the case 2 for completeness.



```
% CASE 2 Sysala-Valdman

 elseif DP_crit<0

  n_sp= n_sp+1;
  plastic_points=[plastic_points; [x y]];
  coeff=DP_yield/(mu_ie+K*eta^2);
  T_k=sigma_tr-coeff*(mu_ie*sqrt(2)*n_tr+K*eta*[1,0,0,1,1]');

    for i = 1 : femregion.nln % loop over scalar shape functions

        tempxy=sqrt(2)*mu_ie*n_tr+K*eta*[1,0,0,1,1]';

        eps_i0=[grad_x(i), grad_y(i)/2, grad_y(i)/2, 0, 0]';                         % eps(phi_i,0)
        tempx1=[2*grad_x(i)/3, grad_y(i)/2, grad_y(i)/2, -grad_x(i)/3, -grad_x(i)/3]'; % I_D*eps(phi_i,0)
        tempx2=sum(n_tr.*eps_i0)*n_tr;                                                % (n_tr x n_tr)*eps(phi_i,0)
        tempx3=sum(tempxy.*eps_i0)*tempxy;                                            % (tempxy x tempxy)*eps(phi_i,0)
        corrector_i0=(coeff*2*sqrt(2)*mu_ie^2/rho_tr)*(tempx1-tempx2)+tempx3/(mu_ie+K*eta^2);  % T_k0*eps(phi_i,0)

        eps_0i=[0, grad_x(i)/2, grad_x(i)/2, grad_y(i), 0]';                         % eps(0,phi_i)
        tempy1=[-grad_y(i)/3, grad_x(i)/2, grad_x(i)/2, 2*grad_y(i)/3, -grad_y(i)/3]'; % I_D*eps(0,phi_i)
        tempy2=sum(n_tr.*eps_0i)*n_tr;                                                % (n_tr x n_tr)*eps(0,phi_i)
        tempy3=sum(tempxy.*eps_0i)*tempxy;                                            % (tempxy x tempxy)*eps(0,phi_i)
        corrector_0i=(coeff*2*sqrt(2)*mu_ie^2/rho_tr)*(tempy1-tempy2)+tempy3/(mu_ie+K*eta^2);  % T_k0*eps(0,phi_i)
```

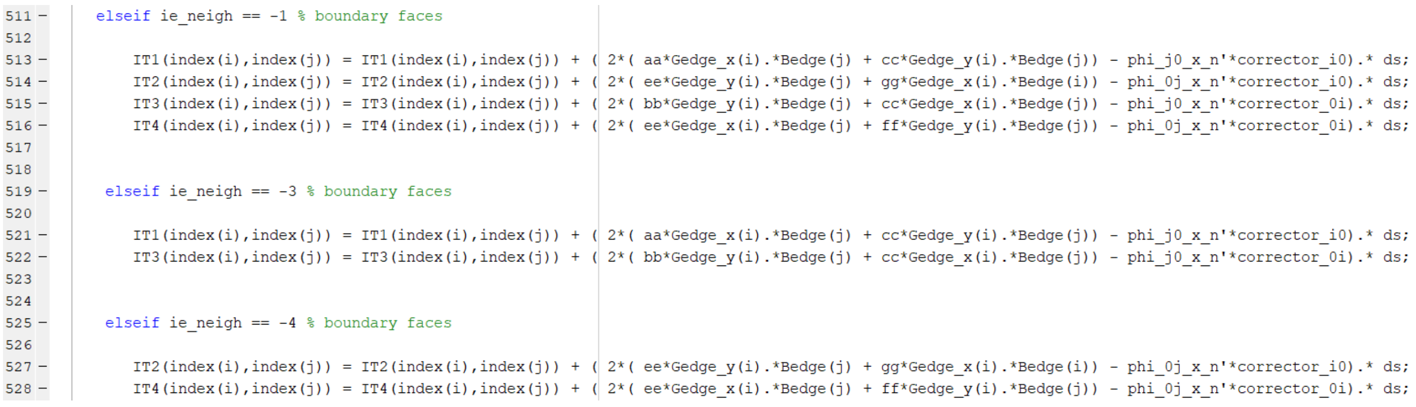
As you can notice the preprocessing is exactly the same as for the implementation of the matrix **V**.



```
489 - for j = 1 : femregion.nln % loop over scalar shape functions
490
491 -     phi_j0_x_n=[Bedge(j)*normals(1,iedg), Bedge(j)*normals(2,iedg), 0, 0, 0]';    % tensor product between (phi_j, 0) and n+
492 -     phi_0j_x_n=[0, 0, Bedge(j)*normals(1,iedg), Bedge(j)*normals(2,iedg), 0]';    % tensor product between (0, phi_j) and n+
493
494 -     if ie_neigh > 0 %ie_neigh ~= -1 && ie_neigh ~= -2 (internal faces)
495
496 -         Bedgeneigh = B_edge_neigh(k,:);
497
498 -         phi_neigh_j0_x_n=[Bedgeneigh(j)*normals(1,iedg), Bedgeneigh(j)*normals(2,iedg), 0, 0, 0]';
499 -         phi_neigh_0j_x_n=[0, 0, Bedgeneigh(j)*normals(1,iedg), Bedgeneigh(j)*normals(2,iedg), 0]';
500
501 -         IT1(index(i),index(j)) = IT1(index(i),index(j)) + ( aa*Gedge_x(i).*Bedge(j) + cc*Gedge_y(i).*Bedge(j) - phi_j0_x_n'*corrector_i0/2).* ds;
502 -         IT2(index(i),index(j)) = IT2(index(i),index(j)) + ( ee*Gedge_y(i).*Bedge(j) + gg*Gedge_x(i).*Bedge(j) - phi_0j_x_n'*corrector_i0/2).* ds;
503 -         IT3(index(i),index(j)) = IT3(index(i),index(j)) + ( bb*Gedge_y(i).*Bedge(j) + cc*Gedge_x(i).*Bedge(j) - phi_j0_x_n'*corrector_0i/2).* ds;
504 -         IT4(index(i),index(j)) = IT4(index(i),index(j)) + ( ee*Gedge_x(i).*Bedge(j) + ff*Gedge_y(i).*Bedge(j) - phi_0j_x_n'*corrector_0i/2).* ds;
505
506 -         ITN1(i,j,iedg) = ITN1(i,j,iedg) - ( aa*Gedge_x(i).*Bedgeneigh(j) + cc*Gedge_y(i).*Bedgeneigh(j) - phi_neigh_j0_x_n'*corrector_i0/2).* ds;
507 -         ITN2(i,j,iedg) = ITN2(i,j,iedg) - ( ee*Gedge_y(i).*Bedgeneigh(j) + gg*Gedge_x(i).*Bedgeneigh(j) - phi_neigh_0j_x_n'*corrector_i0/2).* ds;
508 -         ITN3(i,j,iedg) = ITN3(i,j,iedg) - ( bb*Gedge_y(i).*Bedgeneigh(j) + cc*Gedge_x(i).*Bedgeneigh(j) - phi_neigh_j0_x_n'*corrector_0i/2).* ds;
509 -         ITN4(i,j,iedg) = ITN4(i,j,iedg) - ( ee*Gedge_x(i).*Bedgeneigh(j) + ff*Gedge_y(i).*Bedgeneigh(j) - phi_neigh_0j_x_n'*corrector_0i/2).* ds;
```

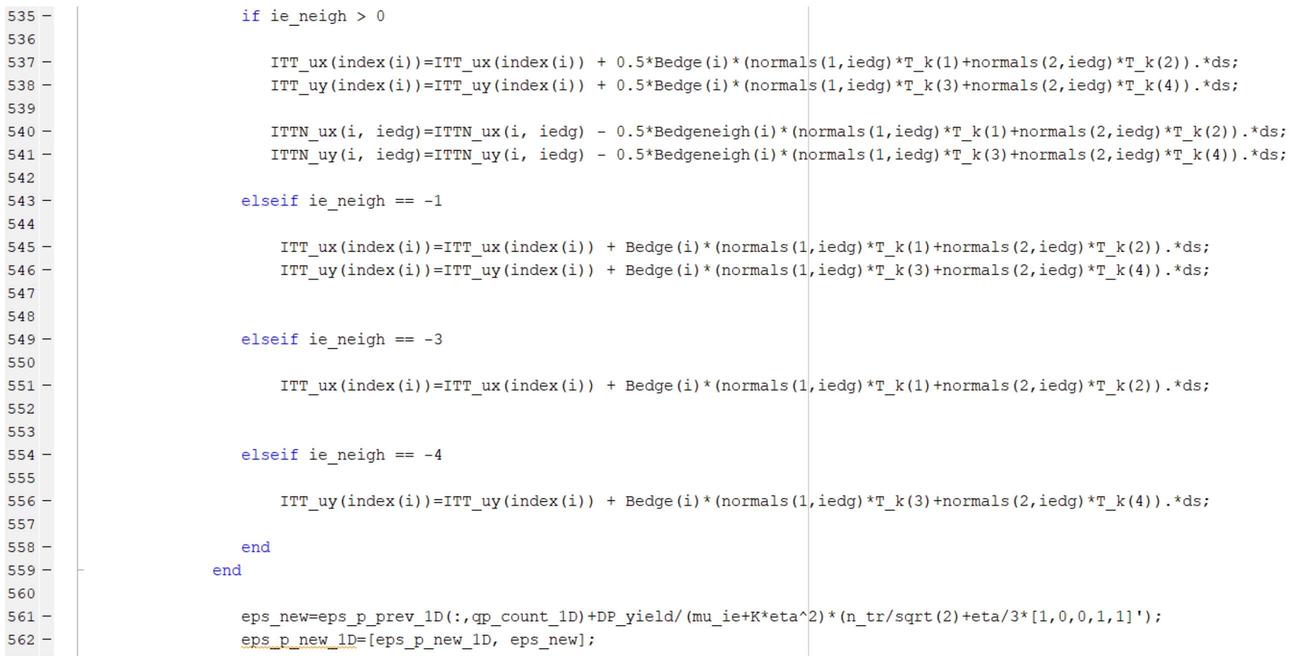
We start with the linearized form, so we consider $\sigma(f) \approx T_k^0 \varepsilon(f)$.
At lines 491-492 we find the tensor product between the outer normal direction and the two types of basis functions ($[\Phi_j, 0]$ at line 491 and $[0, \Phi_j]$ at line 492) of the element under investigation, while at lines 498-499 we have the same computation but for the basis functions of the neighbour element.
Using the notation introduced before, at lines 501-504 the contribution of A^+ is directly inserted in the four sub-matrices of **IT** while, at lines 506-509, the contribution of A^- is stored in the four

sub-tensors of the third-order tensor **ITN**.



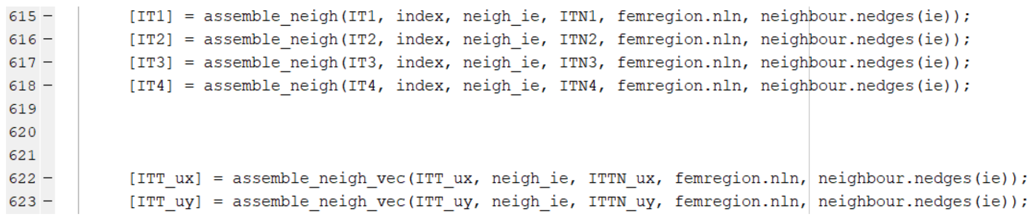
```
    elseif ie_neigh == -1 % boundary faces

        IT1(index(i),index(j)) = IT1(index(i),index(j)) + ( 2*( aa*Gedge_x(i).*Bedge(j) + cc*Gedge_y(i).*Bedge(j)) - phi_j0_x_n'*corrector_i0).* ds;
        IT2(index(i),index(j)) = IT2(index(i),index(j)) + ( 2*( ee*Gedge_y(i).*Bedge(j) + gg*Gedge_x(i).*Bedge(i)) - phi_0j_x_n'*corrector_i0).* ds;
        IT3(index(i),index(j)) = IT3(index(i),index(j)) + ( 2*( bb*Gedge_y(i).*Bedge(j) + cc*Gedge_x(i).*Bedge(j)) - phi_j0_x_n'*corrector_0i).* ds;
        IT4(index(i),index(j)) = IT4(index(i),index(j)) + ( 2*( ee*Gedge_x(i).*Bedge(j) + ff*Gedge_y(i).*Bedge(j)) - phi_0j_x_n'*corrector_0i).* ds;


     elseif ie_neigh == -3 % boundary faces

        IT1(index(i),index(j)) = IT1(index(i),index(j)) + ( 2*( aa*Gedge_x(i).*Bedge(j) + cc*Gedge_y(i).*Bedge(j)) - phi_j0_x_n'*corrector_i0).* ds;
        IT3(index(i),index(j)) = IT3(index(i),index(j)) + ( 2*( bb*Gedge_y(i).*Bedge(j) + cc*Gedge_x(i).*Bedge(j)) - phi_j0_x_n'*corrector_0i).* ds;


     elseif ie_neigh == -4 % boundary faces

        IT2(index(i),index(j)) = IT2(index(i),index(j)) + ( 2*( ee*Gedge_y(i).*Bedge(j) + gg*Gedge_x(i).*Bedge(i)) - phi_0j_x_n'*corrector_i0).* ds;
        IT4(index(i),index(j)) = IT4(index(i),index(j)) + ( 2*( ee*Gedge_x(i).*Bedge(j) + ff*Gedge_y(i).*Bedge(j)) - phi_0j_x_n'*corrector_0i).* ds;
```

For the interfaces on the boundary the computation is simpler since there is no neighbour to consider, however little adaptations are needed. Moreover recall that the term that we are assembling is an integral over all the interfaces except the ones in which a Dirichlet boundary condition is imposed.



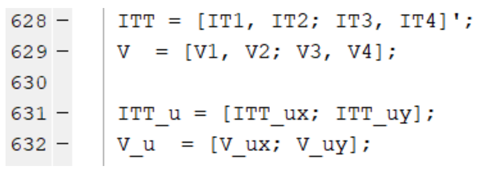
```
535 -                    if ie_neigh > 0
536
537 -                       ITT_ux(index(i))=ITT_ux(index(i)) + 0.5*Bedge(i)*(normals(1,iedg)*T_k(1)+normals(2,iedg)*T_k(2)).*ds;
538 -                       ITT_uy(index(i))=ITT_uy(index(i)) + 0.5*Bedge(i)*(normals(1,iedg)*T_k(3)+normals(2,iedg)*T_k(4)).*ds;
539
540 -                       ITTN_ux(i, iedg)=ITTN_ux(i, iedg) - 0.5*Bedgeneigh(i)*(normals(1,iedg)*T_k(1)+normals(2,iedg)*T_k(2)).*ds;
541 -                       ITTN_uy(i, iedg)=ITTN_uy(i, iedg) - 0.5*Bedgeneigh(i)*(normals(1,iedg)*T_k(3)+normals(2,iedg)*T_k(4)).*ds;
542
543 -                    elseif ie_neigh == -1
544
545 -                        ITT_ux(index(i))=ITT_ux(index(i)) + Bedge(i)*(normals(1,iedg)*T_k(1)+normals(2,iedg)*T_k(2)).*ds;
546 -                        ITT_uy(index(i))=ITT_uy(index(i)) + Bedge(i)*(normals(1,iedg)*T_k(3)+normals(2,iedg)*T_k(4)).*ds;
547
548
549 -                    elseif ie_neigh == -3
550
551 -                        ITT_ux(index(i))=ITT_ux(index(i)) + Bedge(i)*(normals(1,iedg)*T_k(1)+normals(2,iedg)*T_k(2)).*ds;
552
553
554 -                    elseif ie_neigh == -4
555
556 -                        ITT_uy(index(i))=ITT_uy(index(i)) + Bedge(i)*(normals(1,iedg)*T_k(3)+normals(2,iedg)*T_k(4)).*ds;
557
558 -                    end
559 -                end
560
561 -                    eps_new=eps_p_prev_1D(:,qp_count_1D)+DP_yield/(mu_ie+K*eta^2)*(n_tr/sqrt(2)+eta/3*[1,0,0,1,1]');
562 -                    eps_p_new_1D=[eps_p_new_1D, eps_new];
```

We now analyze the exact (nonlinear) form, i.e. $\sigma\left(u\right) \approx T_k$.
Similarly as before, for an inner interface at lines 537-538 the contribution of A^{+} is directly inserted in the two subvectors of **ITT_u** while, at lines 540-541, the contribution of A^{-} is stored in the two subtensors of the two-order tensor **ITTN_u**. Lines 543-558 take into account the possibility of dealing with a boundary interface with any kind of boundary condition imposed on it. Finally lines 561-562 updates the plastic strain in the current quadrature node following the rule for the case 2.



```
615 -       [IT1] = assemble_neigh(IT1, index, neigh_ie, ITN1, femregion.nln, neighbour.nedges(ie));
616 -       [IT2] = assemble_neigh(IT2, index, neigh_ie, ITN2, femregion.nln, neighbour.nedges(ie));
617 -       [IT3] = assemble_neigh(IT3, index, neigh_ie, ITN3, femregion.nln, neighbour.nedges(ie));
618 -       [IT4] = assemble_neigh(IT4, index, neigh_ie, ITN4, femregion.nln, neighbour.nedges(ie));
619
620
621
622 -       [ITT_ux] = assemble_neigh_vec(ITT_ux, neigh_ie, ITTN_ux, femregion.nln, neighbour.nedges(ie));
623 -       [ITT_uy] = assemble_neigh_vec(ITT_uy, neigh_ie, ITTN_uy, femregion.nln, neighbour.nedges(ie));
```

At lines 615-618 the assemble of **IT** is completed, merging the partial contribution collected in the four submatrices with the contribution stored in the third-order tensor **ITN**. Without entering in details, the function **assemble_neigh** receives two matrices to merge and some other parameters that allow, for every contribution stored in the neighbour matrix (whose position is determined by the local indexes of the neighbour's basis functions), the global index associated to it, so that the merging can occur in the right way. We implemented in complete analogy the function **assemble_neigh_vec** which does the same thing but with vectors instead of matrices and it is used at lines 622-623 to complete the assembly of **ITT_u**.



```
628 -   ITT = [IT1, IT2; IT3, IT4]';
629 -   V   = [V1, V2; V3, V4];
630
631 -   ITT_u = [ITT_ux; ITT_uy];
632 -   V_u   = [V_ux; V_uy];
```

The assembly of the $2N_h \times 2N_h$ matrices and the $2N_h$-dimensional vectors is done at lines 628-632. We recall that the matrix **IT** is the transpose of the quantity we are interested in, while the vector **ITT_u** doesn't need any transformation.

4.5. Numerical results

In this section we present some numerical results obtained with the code described in the last section. Due to the complexity of the problem it's very hard to impose an analytical exact solution, so we will show convergence results only for the Newton algorithm.

4.5.1. A comparison between elastic and elasto-plastic responses

We start analyzing the elasto-plastic response corresponding to the same input data that of the problem shown in section 3.5.1. but instead of a homogeneous Dirichlet condition we impose the displacement to be equal to 10 for both components in all the boundary. The tolerance for the criterion to is set equal to 10^{-8} for simple convergence and 10^{-10} for super-convergence.

Step	Nº iter	Load incr	Total load	Criterion	Super conv
1	5	0.001	0.001	$9.72 * 10^{-11}$	yes
2	3	0.002	0.003	$2.016 * 10^{-12}$	yes
3	3	0.004	0.007	$7.40 * 10^{-11}$	yes
4	2	0.008	0.015	$8.5904 * 10^{-13}$	yes
5	2	0.016	0.031	$3.0309 * 10^{-13}$	yes
6	2	0.032	0.063	$1.421 * 10^{-12}$	yes
7	2	0.064	0.127	$1.176 * 10^{-12}$	yes
8	1	0.128	0.255	$2.2091 * 10^{-13}$	yes
9	1	0.256	0.511	$3.5034 * 10^{-13}$	yes
10	1	0.512	1.023	$1.402 * 10^{-12}$	yes

Table 1. Newton algorithm information for each loading step

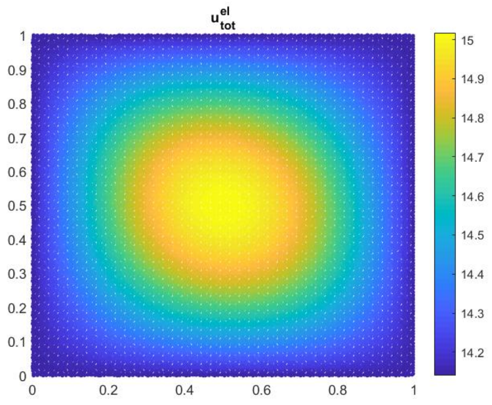


Fig23. Solution obtained imposing pure elastic response

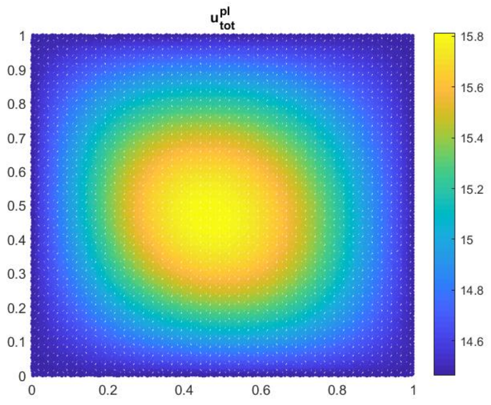


Fig24. Elasto-plastic response with Drucker-Prager yield criterion

Before discussing the results we recall that both the total load and the load increment represent a fraction of the total force so their value ranges from 0 to 1). For this simple case the algorithm converges rapidly as we can see from table 1: the Newton algorithm always reach the super convergence, so the load increment is doubled every time and so 11 load steps are enough to reach the full load. Since the initial guess is the pure elastic response, the first load step is the one that needs more iterations but then, after capturing the trend of the plastic response, the number of needed iterations quickly decreases.
From figures 23 and 24 we can make a comparison between the pure elastic and the elasto-plastic responses and we can see that in the latter we have an extra displacement, given by the plastic component of the strain.

4.5.2. Non-smooth data on a square domain

As last result we analyze a square domain with the same type of boundary conditions presented in section 3.5.1 but imposing, on the top left part of the domain, a Dirichlet boundary condition on the y direction equal to -1. The mesh is composed by $N = 100$ elements and the polynomial degree is set equal to 1.

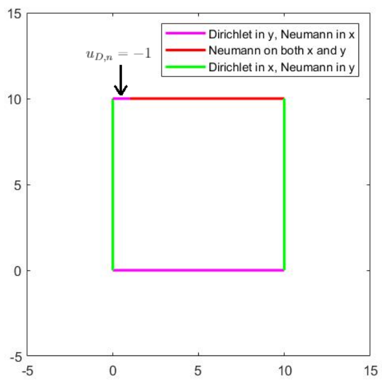


Fig25. Domain with boundary conditions

This problem may seems to be very simple but it is not and we will see that it brings some complications during its resolution.
For this particular problem we can compare the solution with the result that we obtained using another already implemented code, that solves the same problem with a Finite Element Continuous Galerkin method. For more details about the code we remind to [4].
Since the we non smooth data, the algorithms converges more slowly compared to the previous case, so we will show the results only for the first three iterations:

Step	Nº iter	Load incr	Total load	Criterion	Super conv
1	9	0.001	0.001	$2.2292 * 10^{-13}$	yes
2	9	0.001	0.002	$7.7999 * 10^{-13}$	yes
3	12	0.001	0.003	$5.7398 * 10^{-13}$	yes

Table 2. Newton algorithm information for the first three loading step

Also in this case, at least for the first load steps, we reach super convergence even if this time the Newton algorithm needs more iterations. Notice that, even if we reach super convergence, we are not doubling the load increment and this was only to compare our solution with the reference one in which the doubling of the load was not performed.

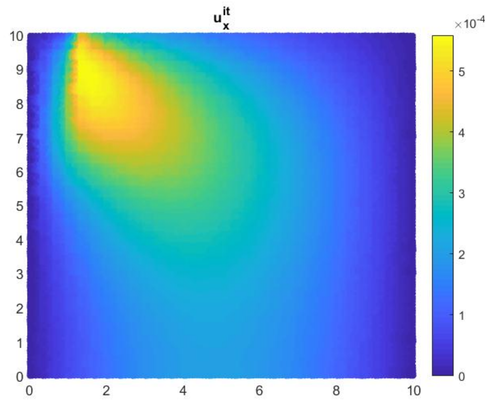


Fig26. Displacement in the x direction after 1 iteration (DGFEM)

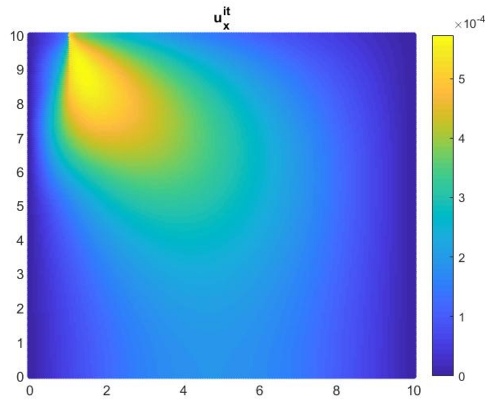


Fig27. Displacement in the x direction after 1 iteration (reference code)

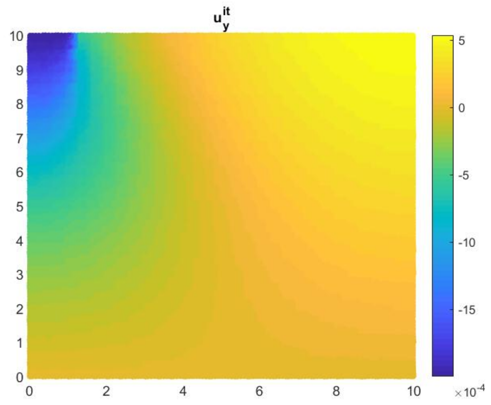


Fig28. Displacement in the y direction after 1 iteration (DGFEM)

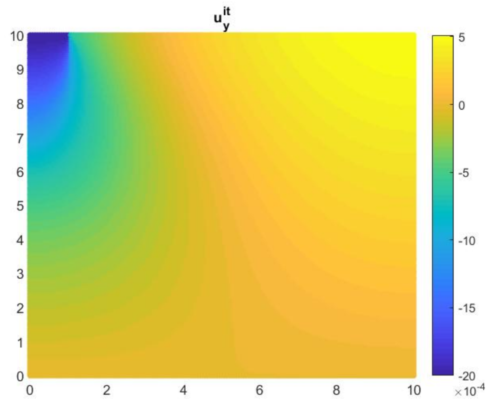


Fig29. Displacement in the y direction after 1 iteration (reference code)

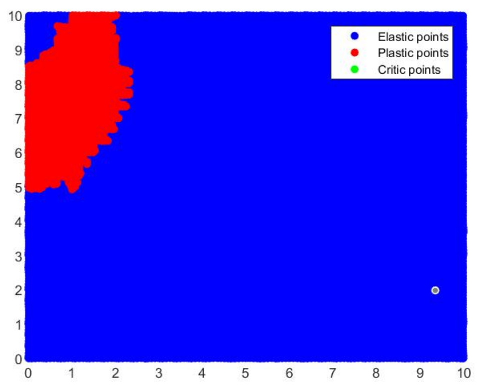


Fig30. Distribution of different types of nodes after 1 iteration (DGFEM)

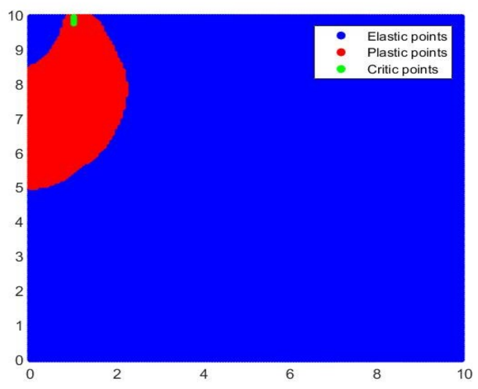


Fig31. Distribution of different types of nodes after 1 iteration (reference code)

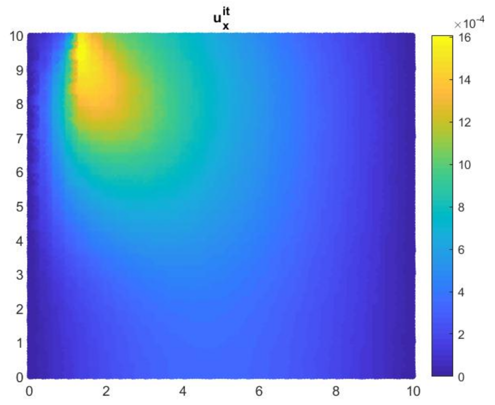


Fig32. Displacement in the x direction after 2 iterations (DGFEM)

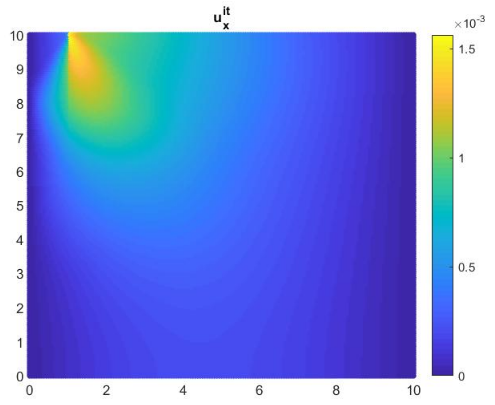


Fig33. Displacement in the x direction after 2 iteration (reference code)

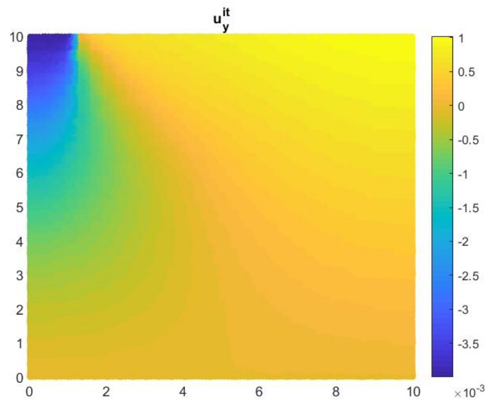


Fig34. Displacement in the y direction after 2 iterations (DGFEM)

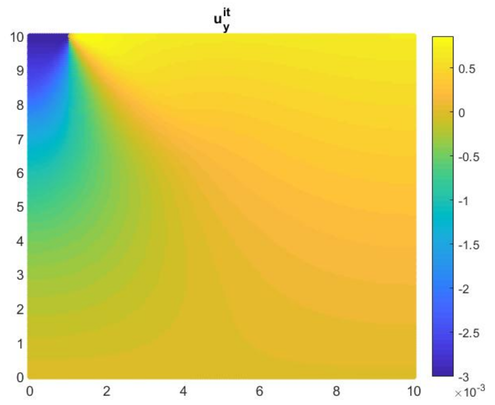


Fig35. Displacement in the y direction after 2 iterations (reference code)

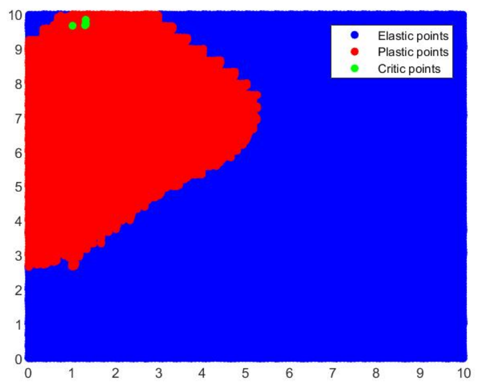


Fig36. Distribution of different types of nodes after 2 iteration (DGFEM)

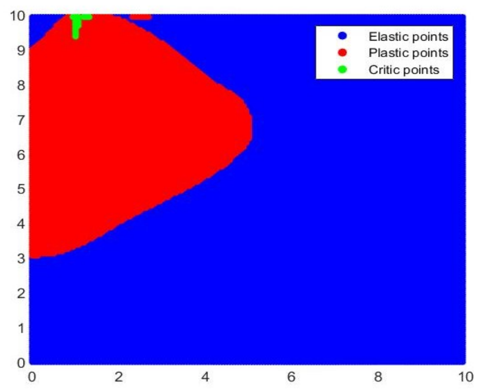


Fig37. Distribution of different types of nodes after 2 iteration (reference code)

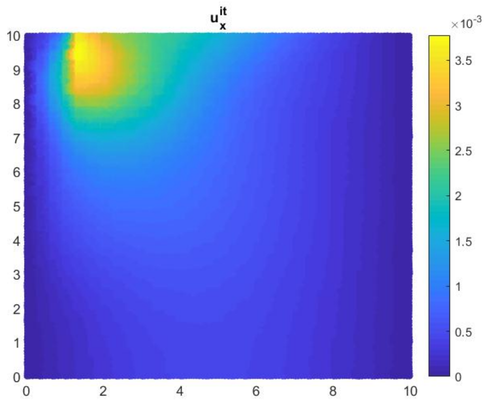


Fig38. Displacement in the x direction after 3 iterations (DGFEM)

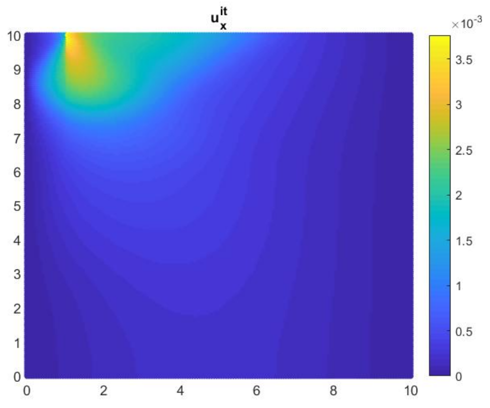


Fig39. Displacement in the x direction after 3 iterations (reference code)

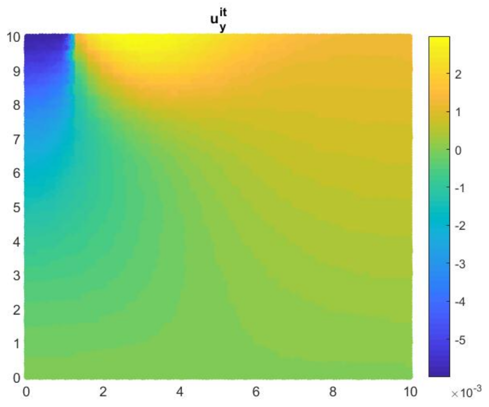


Fig40. Displacement in the y direction after 3 iterations (DGFEM)

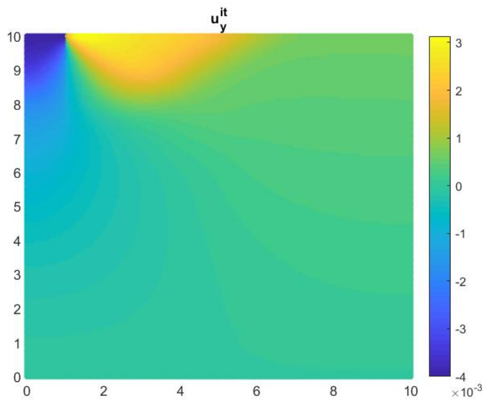


Fig41. Displacement in the y direction after 3 iterations (reference code)

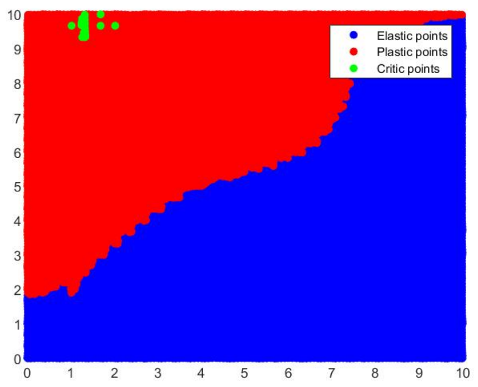


Fig42. Distribution of different types of nodes after 3 iterations (DGFEM)

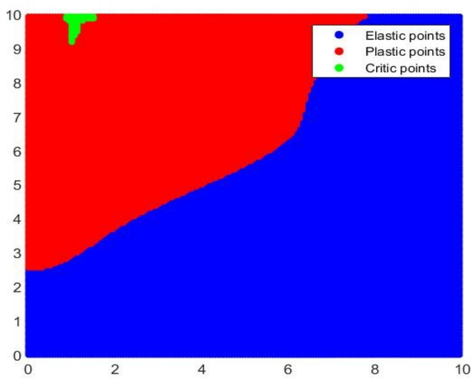


Fig43. Distribution of different types of nodes after 3 iterations (reference code)

From figures 26-43 we can compare the results obtained with our code with the one obtained with the reference code. For the first three iterations the displacements in both directions and the distribution of the three possible types of nodes over the domain are reported. As we can see the plots of the displacement are very similar to the reference ones and, comparing the distributions of the types of nodes, we can be sure that the plastic response is captured by our code.

5. Conclusions and further developments

The main focus of this project was to implement in Matlab a code in order to solve the elasto-plastic problem. We started deriving the DGFEM formulation for the Poisson problem with all the theoretical associated results.
We then moved to the elastic problem, where we introduced the possibility to assign different kind of boundary conditions for each component of the displacement and to construct unstructured polyhedric meshes. For this problem we showed numerical results about 3 simple examples in order to test the new features and the convergence results perfectly match the theoretical estimates.
Finally we generalized the code with the aim to capture also the plastic response. For this purpose we adopted the Drucker-Prager yield criterion. In order to overcome the non-linearity of the problem we divided the total load in a fixed number of load steps and each one is solved by means of a Newton algorithm. We then showed the results of two problem, in particular for the last we obtained very similar results to the one produced by a reference code for the first iterations.
However for the last problem a full simulation was run and, after some load steps, instabilities occur and the Newton algorithm is no more able to reach convergence.
For this reason in the following we propose some ideas that could fix some issues and some others for further developments:

- This numerical methods was already studied and implemented in [4] but with a different method (Continuous Galerkin), so the well posedness and the convergence results should be proved even for our method.
- Since we have non smooth data and we know that one problem in the algorithm could be the presence of bad scaled matrices one can try to replace the Newton algorithm with one of its inexact versions. For this purpose in the code we implemented also the Quasi-Newton algorithm, more precisely the BFGS method, but the parameters need some tuning.
- One can try to tune the optimal penalty coefficient, the load increment and the hyper-parameters of the Newton algorithm in order to have convergence until the final step.
- For each load step we solve a non linear problem through the Newton algorithm, moreover we need a big number of elements in order to capture the plastic behaviour of the solution and we could have many load steps. For these reasons the code is very expensive in computational terms and it may takes hours to see the final result. A possible solution to this problem could be the vectorization the code (as it was done in [4] for FEMCG). This would dramatically decrease the overall computational cost.
- Different yield criteria could be implemented in order to study the behaviour of different materials.

- Extend the model to time dependent problems (this is a non trivial task). For example one interesting application of an extension of the elastoplastic problem is seismic wave propagation, in which the plastic behaviour could be used to represent the correct response for soil materials. A good starting point for this purpose can be found in [5].

References

[1] Antonietti P. F. Quarteroni A. *Numerical performance of discontinuous and stabilized continuous Galerkin methods for convection–diffusion problems.* Applied Mathematics and Computation. 2019, pp. 595–614.

[2] Owen D. R. J. Hinton E. *Finite Element in Plasticity: Theory and Practice.* University College of Swansea, UK, 1980.

[3] Stein E. *Singular Integrals and Differentiability Properties of Functions.* Princeton University Press, Princeton, 1970.

[4] Cermak M. Sysala S. Valdman J. *Efficient and flexible MATLAB implementation of 2D and 3D elastoplastic problems.* Applied Mathematics and Computation. 2019, pp. 595–614.

[5] Antonietti P. F. Facciolà C. Houston P. Mazzieri I. Pennesi G. Verani M. *High order discontinuous Galerkin methods on polyhedral grids for geophysical applications: seismic wave propagation and fractured reservoir simulations.* 2020.

[6] Blaheta R. Zeman J. Kruis J. Koudelka T. Cermak M. Sysala S. *Subdifferential-based implicit return-mapping operators in computational plasticity.* Applied Mathematics and Mechanics. 2016, pp. 1318–1338.

[7] Owen D. R. J. Peric D. de Souza Neto E. A. *Computational Methods for Plasticity.* 2008.