MASTER'S DEGREE THESIS

Master of Science in Advanced Mathematics and Mathematical Engineering

Title: Numerical approximation of Poisson problems using high-order continuous Galerkin methods with static condensation

Author: Antonio Pérez Álvarez

Advisors: Eloi Ruiz Gironés and Jose Sarrate Ramos

Department: Departament de Matemàtica Aplicada III

Academic year: 2014-2015

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Master Thesis

Numerical approximation of Poisson problems using high-order continuous Galerkin methods with static condensation

Antonio Pérez Álvarez

Advisor: Eloi Ruiz Gironés and Jose Sarrate Ramos

Departament de Matemàtica Aplicada III

Abstract

Keywords: Finite element method, high-order methods, continuous Galerkin, static condensation, Poisson.

Higher-order methods in finite elements can provide better approximations than linear methods, in some problems. This is because they can offer an exponential convergence rate of the solution. Thus, in some applications, high-order methods can be cheaper than low-order methods. Nonetheless, it is of major importance to provide good implementations in order to reduce the computational cost of solving a problem. To this end, we propose to use the classical continuous Galerkin method with static condensation procedure to reduce the memory footprint and the CPU time. The main idea consists on write the unknowns related to the inner nodes of each element in terms of the unknowns related to the boundary nodes of the elements. Thus, this method effectively suppress all the unknowns that correspond to pure interior elemental nodes.

To show these properties, we apply the static condensation technique to the Poisson problem. We will particularize the proposed technique for this problem, and we will compare the obtained solution and the computational cost with a classical implementation of the high-order continuous Galerkin method. In order to formulate the method correctly, all the needed results are introduced. The results show that static condensation is a valid choice since it reduces the computational cost of solving a problem.

Contents

Chapter 1. Introduction 1

Chapter 2. The Numerical method 3
1. Mathematical formulation of the problem 3
2. Weak formulation and Galerkin approximation 4
3. Linear System 10
4. Static condensation Procedure 14

Chapter 3. Examples 17

Chapter 4. Conclusions 23

References 25

Chapter 1
Introduction

When selecting the method of solving an engineering problem formulated in partial differential equations (PDE), we take into consideration several criteria: numerical properties, implementation time, computational cost, robustness and geometric flexibility. Based on the needs of the user, it is selected the one that best meets these criteria. With this presetting, as to the results that we want to obtain, the *Finite Element Method* (FEM) can provide in general good approximations. This method uses piecewise polynomials to approximate the solution of such problems. The latest research in this field [**2, 5, 6, 7, 8, 10, 11, 13, 15, 17, 18, 19**] further indicates that high-order adaptive methods successfully respond to the growing complexity of the problems that are solved. In fact, they provide a better accuracy than lower-order methods for a smooth solution. In recent years, unstructured high-order methods have been postulated as a good technique when performing high-quality approximations. In this project it will be discussed, in the framework of the high-order FEM.

It is important make good implementations of the method. For large problems there is higher resources intake in terms of time and memory, since high-order methods, with regard to the lower-order case, have a high storage cost due to the increase of the number and coupling of the degrees of freedom of the mesh. The aim will be therefore, to provide a tool that allows to reduce the computational cost and the memory requirements.

To solve this issue, we propose to apply the *static condensation* method which involves the suppression of some degrees of freedom of the original problem. In particular, the unknowns associated with nodes that belong to the interior of the elements are written in terms of the unknowns related to the nodes that belong to the boundary of the elements, therefore, the degrees of freedom related to the inner nodes of the elements are affectively suppressed. Note that, as we increase the polynomial degree of the mesh, the number of nodes on the interior of the elements increases more rapidly than the number of nodes on the boundary of the elements. For this reason, the static condensation technique is very interesting to reduce the computational requirements when simulating a physical process.

The rest of the dissertation is structured as follows. In Chapter 2, we present the problem to be solved, and how to discretize it using the continuous Galerkin

method. Then, we introduce the *static condensation* technique in order to reduce the computational requirements of the method. Chapter 3 is focused on providing to the reader two examples embedded in the context of the original problem. On these, will be applied the methodologies described in Chapter 2, to highlight the differences between both methods. The last chapter of this work provides the main conclusions that can be extracted from the results and the lines of the research that could be carried out in the future.

Chapter 2
The Numerical method

This chapter will be focused on the principal objective of the project. That is, to formulate a test problem in a mathematical way through some suitable spaces. After that, it will be discretized using the Continuous Galerkin Method in order to obtain a linear system which allow to produce approximate solutions. Two different techniques will be applied in order to solve the corresponding system and then, in Chapter 3, the computational efficiency of both methods will be compared.

1. Mathematical formulation of the problem

We consider, in general, any bounded domain $\Omega \subset \mathbb{R}^3$, with Lipschitz continuous boundary $\partial\Omega = \Gamma_D \sqcup \Gamma_N$.

We will begin by enunciating some basic results from analysis of partial differential equations, which help us to introduce the problem and its weak (or variational) formulation.

- **Some important spaces**
  We define the space of *square integrable functions*:

  $$L^2(\Omega) = \left\{ f : \Omega \longrightarrow \mathbb{R} \text{ Lebesgue measurable s.t. } \int_\Omega |f(\mathbf{x})|^2 d\mathbf{x} < \infty \right\}.$$

  This is a Hilbert space endowed with the scalar product

  $$(f, g)_{L^2(\Omega)} = \int_\Omega f(\mathbf{x}) g(\mathbf{x}) d\mathbf{x}$$

  and associated norm

  $$||f||_{L^2(\Omega)} = \left(\int_\Omega |f(\mathbf{x})|^2 d\mathbf{x} \right)^{1/2}.$$

  We denote by $H^k(\Omega)$ the *Sobolev space* of order $k \in \mathbb{N}$, $k > 0$

  $$H^k(\Omega) = \{ f \in L^2(\Omega) \text{ s.t. } \partial^{|\alpha|} L^2(\Omega),\ |\alpha| \leq k \}.$$

  In particular: $H^0(\Omega) = L^2(\Omega)$. The space $H^{1/2}(\Gamma) \subset L^2(\Gamma)$ are the set of functions belonging to $L^2(\Gamma)$ that are traces of functions of $H^1(\Omega)$.

Once we have these definitions, we consider as the test problem the Poisson equation with Dirichlet and Neumann boundary conditions denoted by Γ_D and Γ_N, respectively.

$$\begin{cases} -\nabla \cdot (k\nabla u) = f, & x \in \Omega, \\ u = g_D, & x \in \Gamma_D, \\ k\dfrac{\partial u}{\partial \mathbf{n}} = g_N, & x \in \Gamma_N. \end{cases} \tag{1}$$

We have introduced a function $k \in L^\infty(\Omega)$ as coefficient of the Poisson problem. Furthermore, we assume that the function k can be either constant or dependent of x and it is called permeability function. We suppose $u \in C^2(\Omega)$, $f \in L^2(\Omega)$, $g_D \in H^{1/2}(\Gamma_D)$ and $g_N \in L^2(\Gamma_N)$.

Then we proceed to enunciate the weak formulation of Problem (1) with its corresponding discretization which allows giving an approximate solution via the Continuous Galerkin method.

2. Weak formulation and Galerkin approximation

In order to obtain the weak formulation of the problem we consider the set of functions

$$V(\Omega) := \{u \in H^1(\Omega), u_{|\Gamma_D} = g_D\},$$

the space of functions

$$V_0(\Omega) := \{v \in H^1(\Omega), v_{|\Gamma_D} = 0\}$$

and the following theoretical result:

- **Integration by parts formula**
  Let $u : \Omega \longrightarrow \mathbb{R}$ be a $\mathcal{C}^2(\overline{\Omega})$ function and $v : \Omega \longrightarrow \mathbb{R}$ belonging to the space $\mathcal{C}(\Omega)$. The integration by parts formula (or Green formula) allows writing:
  $$-\int_\Omega \Delta u v \; d\Omega = \int_\Omega \nabla u \nabla v \; d\Omega - \int_{\partial\Omega} (\nabla u \cdot \mathbf{n}) v \; dS$$
  where $\mathbf{n}$ denotes the exterior normal vector.

Then, we multiply at the right side of u by any test function $v \in V_0(\Omega)$ and apply the integration by parts formula. Thus, the left hand side (lhs.) of the Poisson equation becomes:

$$-\int_\Omega \nabla\cdot(k\nabla u)v \; d\Omega = \int_\Omega k\nabla u \nabla v \; d\Omega - \int_{\partial\Omega} v\, k\, \frac{\partial u}{\partial \mathbf{n}} \; d\Gamma = \int_\Omega k\nabla u\nabla v \; d\Omega - \int_{\Gamma_N} v g_N \; d\Gamma_N$$

since $v = 0$ on the Dirichlet boundary.

Taking also the source term sited at the right hand side (rhs.) of the Poisson equation in the Problem (1), we have that the solution of our original problem is equivalent to the solution of the following variational problem:

Find $u \in V(\Omega)$ such that :

$$\int_\Omega k\nabla u \nabla v \; d\Omega - \int_{\Gamma_N} g_N v \; d\Gamma = \int_\Omega f v \; d\Omega, \quad \forall v \in V_0(\Omega). \tag{2}$$

Now, we isolate the terms depending only on v from those which depend also on u:

$$\int_\Omega k\nabla u\nabla v \; d\Omega = \int_\Omega fv \; d\Omega + \int_{\Gamma_N} g_N v \; d\Gamma, \quad \forall v \in V_0(\Omega).$$

Then we have a simpler formulation in terms of functionals:

(3) Find $u \in V(\Omega)$ such that :
$$a(u,v) = L(v), \quad \forall v \in V_0(\Omega).$$

We will use the weak formulation in order to discretize the continuous equation and transform the PDE into a linear system of equations:

$$\mathbf{Au} = \mathbf{b}.$$

We will explain better this step in passing to the system at the end of the present Chapter in Section 2.2. Then, in Section 3, we will compute the coefficients of matrix $\mathbf{A}$.

The variational problem in Equation (3) is, in fact, symmetric and its linear system could be solved via some very common and well known methods, like the Cholesky factorization, because $\mathbf{A}$ is non-singular. Our proposal in this project is to provide a better methodology to reduce the amount of memory used, since for large problems the matrix of the linear system can use a lot of memory resources.

2.1. Decomposition of the domain Ω. In this part of the chapter, the domain will be decomposed in order to discretize Equation (3). Although it is shown for a two-dimensional problem for viewing purposes, the method works for a three-dimensional problem without any modifications.

Let be $\mathcal{T}_h(\Omega)$ a polyhedral decomposition of Ω in the following manner:

$$\overline{\Omega} = \bigcup_{T \in \mathcal{T}_h(\Omega)} T$$

being T the elements of the tessellation. For instance, a simple case of a square divided into 16 elements in FIG. 1 is shown.

We will consider in our study quadrilateral elements, for 2D case, and hexahedral elements, for $3D$ case. Onwards, h denotes the diameter of the element considered, $h = diam(T)$.

OBSERVATION 1. The extension to the three-dimensional case is assumed as natural making appropriate adjustments notation, that is, adding new dimensional elements (volumes, inner faces, boundary faces, inner nodes, inner boundary nodes, boundary nodes and edges).

The typical tessellation of a decomposition has the usual topological properties of a triangulation and for our study case is composed by conformal elements. The main properties to take into account are:

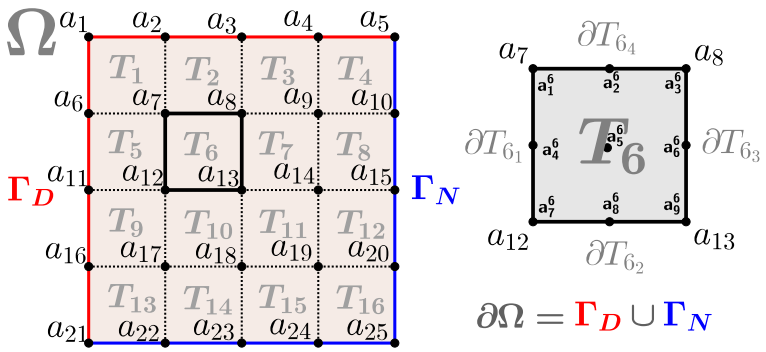

FIG. 1. Simple decomposition of a $2D-$square domain Ω into a tessellation $\mathcal{T}_h(\Omega)$ formed by 16 elements denoted by T_e, $e = 1, \ldots, 16$. Each one of them has four vertices, and four edges ∂T_{e_i}, $i = 1, \ldots, 4$. There is a global numeration for the nodes which are also vertices of the element but we have omitted this numeration for the remaining nodes. It is shown in detail the sixth element, which is composed by 9 nodes (four of them are also vertices), this should be for a shape function of second degree on each variable. It is shown, also, the global numeration for its vertices and the local one for the nine nodes of the element.

(1) $\forall T_1, T_2 \in \mathcal{T}_h,\ T_1 \neq T_2 : \partial T_1 \cap \partial T_2 = \begin{cases} \emptyset \\ 1 \text{ common vertex.} \\ 1 \text{ common edge.} \\ 1 \text{ common face.} \end{cases}$

OBSERVATION 2. As a consequence: It is not allowed have overlapping elements or hanging nodes/edges. That is, there cannot be any vertex in the middle of edges and by extension vertices or edges in middle of faces.

(2) The decomposition has to be compatible with the boundary conditions.

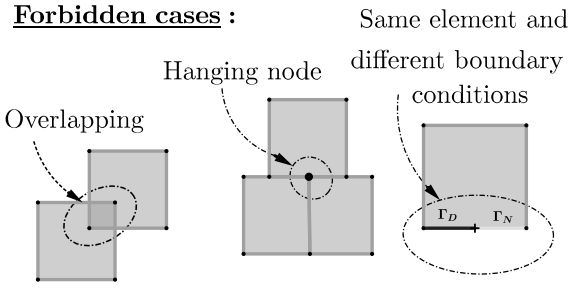

FIG. 2. Main cases of incorrect construction of a triangulation ($2D$ case).

We must make mention of the fact that there will be two types of nodal numerations. The first one is local, with respect to each element and the second one is global, with respect to the mesh. In FIG. 1 is shown the global and local numeration for the vertices.

To end up with this summary, we denote by $\partial T_i^e \in \Gamma$ where Γ is the set of all the boundaries of the elements, inners and external. Notice here that an edge (face) is interior if it is the intersection of two elements, and exterior or boundary if it is the intersection of one element and the boundary of the domain. Finally, it is considered the set of all the nodes of the decomposition as follows:

$$\Sigma_h = \bigcup_{T\in\mathcal{T}_h} \Sigma_T = \bigcup_{T\in\mathcal{T}_h} \{a_i^T : 1 \leq i \leq N_T\} = \{a_i : 1 \leq i \leq N\},$$

where N_T is the number of nodes of the element T and N is the number of nodes in the whole mesh. Thus, we have denoted by a^T the local numeration of the nodes for a given element T, and by a the global numeration. The notation introduced here is a gentle adaptation of which is shown in [**8**]. This allows to reference the nodes of the domain and the nodes of each element individually.

OBSERVATION 3. It is important to notice that the numeration is not very detailed. It is supposed pertinently done, knowing that it is given data.

2.2. Matricial and variational formulations. Once the domain is discretized, we have a sequence of nodes. Therefore, we can consider the corresponding finite-dimensional subspaces where we can select a suitable finite basis $\{\phi_1, \phi_2, \ldots, \phi_N\}$. It is important to say at this point that the shape functions that we are going to use are high-order polynomials.

Before beginning let us define some approximation spaces that we will use onwards.

$$X_h(\Omega) := \{u_h \in \mathcal{C}^0(\overline{\Omega}) \text{ s.t. } u_{h_{|T}} \in \mathbb{P}_T^p, \forall T \in \mathcal{T}_h\}.$$

$$V_h := \{u_h \in X_h \text{ s.t. } u_{h|\Gamma_D} = g_{D_h}\}.$$

$$V_h^0 := \{v_h \in X_h \text{ s.t. } v_{h|\Gamma_D} = 0\}.$$

The h subscript will mean discretized. Note that $X_h(\Omega)$ is used to approximate $H^1(\Omega)$, V_h to approximate $V(\Omega)$ and V_h^0 to approximate $V_0(\Omega)$. Above, g_{D_h} denotes the approximation of g_D.

The space

$$\mathbb{P}_T^p = \{\phi : T \subset \mathbb{R}^d \to \mathbb{R} \text{ s.t. } \phi \circ F_T = \hat{\phi} \in Q^p(\hat{T})\}, \tag{4}$$

where

$$Q^p(\hat{T}) = \left\{ \phi = \phi(x_1, ...x_d) = \sum_{0\leq\alpha_i\leq p} \gamma_{\alpha_1} \ldots \gamma_{\alpha_d} x_1^{\alpha_1} \ldots x_d^{\alpha_d},\ \gamma_\alpha \in \mathbb{R} \right\}$$

denote the space of polynomials with degree $\leq p$ in each variable, is a space of polynomials with total degree $\leq p$.

The functions F_T and $\hat{\phi}$ will be explained in detail on the section 3 of this chapter.

OBSERVATION 4. Notice that in FIG. 1 it was shown a very particular case of nodal decomposition for the selected element to decompose, T_6. In the case of high-order shape functions for edges, quadrilaterals and hexahedrals, the next formula is satisfied:

$$dim(Q^p(\hat{T})) = (p+1)^n, \quad \text{where } n = dim(\hat{T})$$

which connect the polynomial degree with the number of nodes required for the polynomial to be well defined.

Now we will detail how to discretize the problem described in Equation 3

Firstly, we define the shape functions as continuous piecewise Lagrangian polynomials that satisfy the following properties:

- $\phi_i \in X_h, \qquad 1 \leq i \leq N.$
- $\phi_i(a_j) = \delta_{ij}, \quad 1 \leq i, j \leq N.$

Note that the following relations are satisfied: $V_h(\Omega) \subset X_h(\Omega)$ and $V_0(\Omega) \subset H^1(\Omega)$ as FIG. 3 shows.

$$\begin{array}{ccc} V_h^0(\Omega) & \approx & V_0(\Omega) \\ \cap & & \cap \\ X_h(\Omega) & \approx & H^1(\Omega) \\ \cup & & \cup \\ V_h(\Omega) & \approx & V(\Omega) \end{array}$$

FIG. 3. Principal relations between the main spaces.

Concretely, $dim(V_h^0) = n_h < \infty$ and $dim(X_h) = N < \infty$. Thus, the functions of X_h and V_h^0 can be expressed using a finite linear combination of shape functions.

Now, we put this idea into Equation (3).

$$\begin{aligned} &\text{Find } u_h \in V_h(\Omega) \text{ such that:} \\ &a(u_h, v_h) = L(v_h), \quad \forall v_h \in V_h^0(\Omega). \end{aligned} \tag{5}$$

In fact, $u_h \in X_h$ and , for this reason,

$$u_h = \sum_{i=1}^{N} u_i \phi_i$$

where $u_i := u_h(a_i) \in \mathbb{R}$. In particular $u_h \in V_h$ and therefore we can write it as a linear combination of the basis of V_h. From (5) and take into account Equation (2) we can write:

$$\sum_{i=1}^{N} u_i \int_\Omega \nabla\phi_i k \nabla v_h \, d\Omega = \int_\Omega f v_h \, d\Omega - \int_{\Gamma_N} g_N v_h \, d\Gamma, \quad \forall v_h \in V_h^0. \tag{6}$$

It is only necessary to satisfy Equation (6) by all the functions that form a basis of V_h^0. Note that the shape functions that vanish at the Dirichlet condition form a basis of the space of functions V_h^0. Thus, Equation (6) has to be satisfied by all the shape functions that vanish at the Dirichlet boundary. Then, writing in terms of functionals:

Find $u_i, i = 1, \dots, N$ such that:

$$\sum_{i=1}^{N} u_i a(\phi_i, \phi_j) = L(\phi_j), \quad \forall \phi_j \in V_h^0. \tag{7}$$

In Equation (7) there are more unknowns than equations and therefore, this problem is not well posed. To solve this issue we have to introduce the Dirichlet data condition into the problem. We can assume, without loss of generality, that the shape functions are ordered in such a way that the first ones vanishes at Γ_D. Thus, $\{\phi_1, \dots, \phi_{n_h}\}$ is a basis of V_h^0. We can write u_h as:

$$u_h = \sum_{i=1}^{n_h} u_i \phi_i + \sum_{i=n_h+1}^{N} u_i \phi_i.$$

Finally, our problem becomes:

Find $u_i \in \mathbb{R}$, for $i = 1, \dots, n_h$ such that:

$$\sum_{i=1}^{n_h} u_i a(\phi_i, \phi_j) = L(\phi_j) - \sum_{i=n_h+1}^{N} u_i a(\phi_i, \phi_j), \quad \forall \phi_j \in V_h^0. \tag{8}$$

Here the second summation is known, since, $u_i = g_D(a_i)$ with $i = n_h + 1, \dots, N$.

Notation 5.

$$\mathbf{A} \in \mathcal{M}_{n_h \times n_h}, \ A_{ij} = a(\phi_i, \phi_j).$$

$$\mathbf{b} \in \mathbb{R}^{n_h}, \ b_j = L(\phi_j) - \sum_{i=n_h+1}^{N} g_{Di} a(\phi_i, \phi_j).$$

$$\mathbf{u} \in \mathbb{R}^{n_h}, \ u_{h_i} = u_h(a_i).$$

Notice that here the only unknown is the vector $\mathbf{u}$.

Thus, using the previous notation we have:

$$\sum_{i=1}^{n_h} u_{h_i} A_{ij} = b_j \Leftrightarrow \mathbf{A}\mathbf{u} = \mathbf{b}.$$

As the reader can observe, this last computations end with a linear system. Finally, with this procedure we reach a simpler problem:

Find $\mathbf{u} \in \mathbb{R}^{n_h}$ such that:

$$\mathbf{A}\mathbf{u} = \mathbf{b}. \tag{9}$$

After computing the solution of the system we simply add the Dirichlet condition in the correspondent omitted nodes at the rhs.

Lemma 6 (Existence and uniqueness of solution. Lax-Milgram). Let be V a Hilbert space endowed with the norm $||\cdot||_V$, $a(u, v)$ a bilinear form and $L(v)$ a lineal form. The weak problem:

Find $u \in V$ such that:

$$a(u, v) = L(v), \quad \forall v \in V$$

has unique solution if and only if $a(\cdot,\cdot)$ is continuous and coercive, and $L(\cdot)$ is continuous.

Finally ending with this section, we only need to add that thanks to the LEMMA 6 the Formulation (3) has unique solution. Furthermore, this previous result also allows to ensure that the Galerkin problem associated to it, Equation (5), admits an unique solution too. Moreover, this result allows to enunciate that the linear system wrote above in Equation (9) has its **A** matrix symmetric and positive definite. Therefore it can be solved via a direct or iterative method and has an unique solution. See [**3, 16**] for more details about this subsection.

After this introduction of the main concepts and notation we will proceed with the main formulation of the method described in this master thesis.

3. Linear System

Although the previous formulation is well posed from a mathematical point of view, it does not describe an efficient implementation. Thus, in this section we describe how to assemble the *stiffness matrix* **A** and the rhs. vector **b** or, which is the same, construct it as efficiently as we can.

3.1. Reference and physical elements: Transformations. This section will define central concepts before to obtain the linear system.

For each physical element, T, we use a given reference element, $\hat{T}$, in order to define the shape functions. For quadrilaterals and hexahedrals we use $[-1,1]^2$ and $[-1,1]^3$, respectively. Note that the physical element can be defined in terms of the reference element using a diffeomorphism F_T in the following manner:

$$F_T : \hat{\mathbf{x}} \in \hat{T} \subset \mathbb{R}^d \longrightarrow \mathbf{x} \in T \subset \mathbb{R}^n$$

$$\hat{\mathbf{x}}_i \longrightarrow F_T(\hat{\mathbf{x}}_i) = \sum_i^{N_T} \hat{\phi}_i \mathbf{x}_i,$$

where d is the dimension of reference space and n of the physical one. It is important to clarify that this application in general is not linear. See caption of FIG. 5.

EXAMPLE 7 (Tri-linear polynomial basis for a hexahedral element).

$$\hat{\phi}_i \in \hat{T} \subset \mathbb{R}^3 \longrightarrow \mathbb{R}; \quad \forall i = 1, \ldots, 8.$$

$$\begin{aligned}
\hat{\phi}_1(\xi,\eta,\zeta) &= (1-\xi)(1-\eta)(1-\zeta), & \hat{\phi}_5(\xi,\eta,\zeta) &= \xi\eta(1-\zeta),\\
\hat{\phi}_2(\xi,\eta,\zeta) &= \xi(1-\eta)(1-\zeta), & \hat{\phi}_6(\xi,\eta,\zeta) &= \xi(1-\eta)\zeta,\\
\hat{\phi}_3(\xi,\eta,\zeta) &= (1-\xi)\eta(1-\zeta), & \hat{\phi}_7(\xi,\eta,\zeta) &= (1-\xi)\eta\zeta,\\
\hat{\phi}_4(\xi,\eta,\zeta) &= (1-\xi)(1-\eta)\zeta, & \hat{\phi}_8(\xi,\eta,\zeta) &= \xi\eta\zeta.
\end{aligned}$$

EXAMPLE 8 (Tri-linear affine transformation). We consider opportune to introduce how this type of applications can bring the reference elements into the physical ones for a simple case. We have considered here an hexahedral parallelepiped, which

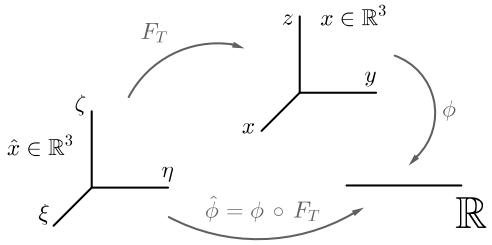

FIG. 4. Relational diagram of central applications. F_T is a diffeomorphism between $\hat{T}$and T.

means parallel faces two to two, as physical element. So in this particular case, the hexahedron considered is linear and so is F_T.

$$F_T : \hat{\mathbf{x}} = (\xi, \eta, \zeta) \in \mathbb{R}^3 \longrightarrow \mathbf{x} = (x, y, z) \in \mathbb{R}^3$$
$$\hat{\mathbf{a}}_0 = (\xi_0, \eta_0, \zeta_0) \longrightarrow F_T(\hat{\mathbf{a}}_0) = \mathbf{a}_0 = (x_0, y_0, z_0).$$

In particular:

$$F_T(\hat{\mathbf{a}}_i) = \mathbf{B}_T \hat{\mathbf{x}} + \mathbf{b}_T = \mathbf{a}_i.$$

Since we are dealing with an hexahedron and consider here the tri-linear case, $i = 1, \ldots, 8$. The matrix $\mathbf{B}_T \in \mathcal{M}_{3\times 3}$ and it is invertible; and $\mathbf{b}_T = \mathbf{a}_1 \in \mathbb{R}^3$. (See FIG. 5). More explicitly the formula for F_T reads as follows:

$$\begin{aligned} F_T &= \left(\begin{array}{c|c|c} \mathbf{a}_2 - \mathbf{a}_1 & \mathbf{a}_4 - \mathbf{a}_1 & \mathbf{a}_5 - \mathbf{a}_1 \end{array} \right) \hat{\mathbf{x}} + \mathbf{b}_T = \\ &= \begin{pmatrix} \mathbf{x}_2 - \mathbf{x}_1 & \mathbf{x}_4 - \mathbf{x}_1 & \mathbf{x}_5 - \mathbf{x}_1 \\ \mathbf{y}_2 - \mathbf{y}_1 & \mathbf{y}_4 - \mathbf{y}_1 & \mathbf{y}_5 - \mathbf{y}_1 \\ \mathbf{z}_2 - \mathbf{z}_1 & \mathbf{z}_4 - \mathbf{z}_1 & \mathbf{z}_5 - \mathbf{z}_1 \end{pmatrix} \begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} = \\ &= \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} (1 - \xi - \eta - \zeta) + \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} \xi + \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \eta + \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \zeta. \end{aligned}$$

Remembering the space $\mathbb{P}_T^p$ introduced in (4) and the diagram shown in FIG. 4, we can write:

$$\hat{\phi}_i = \phi_i \circ F_T \Leftrightarrow \hat{\phi}_i \circ F_T^{-1} = \phi_i, \quad \forall i = 1, \ldots, N_T.$$

This will allow to evaluate the shape functions, in the physical space, although they are defined in the reference space. Recall that, we are doing all the computations element by element and therefore, we have to join them in order to build the linear system.

3.2. Computation of local contributions of the stiffness matrix and the rhs. term. Finally, in this subsection, we will compute the elemental contributions and assemble them to obtain the linear system. Recall that we have defined the shape functions and their derivatives in the reference element. However, we have to

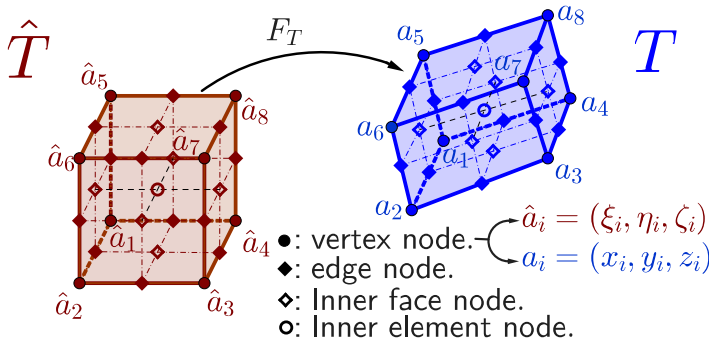

FIG. 5. General invertible application, F_T, from the reference element, $\hat{T}$, to the physical one, T. Particular representation for a second degree polynomial. It is important the fact that T could be curved and so is recommended to take higher order polynomials if we want to approach it better. Therein lies the importance of to take high-order polynomials basis and because of that we will have more accurate in the approximations. On the one hand this is due to the approach of the elements and, on the other hand, on the approximations of shape functions.

compute the bilinear form $a(\phi_i, \phi_j)$ and $L(\phi_j)$ in the physical space. To this end, we define the elemental contribution of the form $a(\phi_i, \phi_j)$ as:

$$a_{_T}(\phi_i, \phi_j) = \int_T \nabla\phi_i \nabla\phi_j k \, dT. \tag{10}$$

Due to the chain rule we can write:

$$\hat{\nabla}\hat{\phi}_i(\hat{\mathbf{x}}) = \hat{\nabla}(\phi_i(F_T(\hat{\mathbf{a}}_i))) = \mathbf{B}_T^t((\nabla\phi_i \circ F_T)(\hat{\mathbf{x}}))) \Leftrightarrow$$
$$\Leftrightarrow \nabla\phi_i(\mathbf{x}) = \mathbf{B}_T^{-t}((\hat{\nabla}\hat{\phi}_i \circ F_T^{-1})(\mathbf{x})), \qquad \forall i = 1, \ldots, N_T,$$

where

$$\mathbf{B}_T = \mathbf{D}\mathbf{F}_T \quad \text{and} \quad |\mathbf{B}_T| = det(\mathbf{B}_T).$$

Here, $\nabla = \begin{pmatrix}\partial x & \partial y & \partial z\end{pmatrix}^t$ and $\hat{\nabla} = \begin{pmatrix}\partial \xi & \partial \eta & \partial \zeta\end{pmatrix}^t$. Applying this to Equation (10):

$$a_{_T}(\phi_i, \phi_j) = \int_{\hat{T}} (\nabla\phi_i \circ F_T)(\hat{\mathbf{x}}) \cdot k(F_T(\hat{\mathbf{x}})) \cdot (\nabla\phi_j \circ F_T)(\hat{\mathbf{x}}) \cdot |\mathbf{B}_T| \, d\hat{\mathbf{x}} =$$
$$= \int_{\hat{T}} \mathbf{B}_T^{-t}\hat{\nabla}\hat{\phi}_i(\hat{\mathbf{x}}) \cdot k(F_T(\hat{\mathbf{x}})) \cdot \mathbf{B}_T^{-t}\hat{\nabla}\hat{\phi}_j(\hat{\mathbf{x}}) \cdot |\mathbf{B}_T| \, d\hat{\mathbf{x}}, \quad \forall i, j = 1, \ldots, N_T.$$

The same is true for the right hand side. In this case we divide it in two terms:

$$L_{_T}(\phi_j) = \int_T f\phi_j \, dT - \int_{\Gamma_N} g_N\phi_j \, d\Gamma, \quad \forall j = 1, \ldots, N_T.$$

- **The source term:**

$$\int_T f(\mathbf{x}) \cdot \phi_j(\mathbf{x}) \, d\mathbf{x} = \int_{\hat{T}} f(F_T(\hat{\mathbf{x}})) \cdot \hat{\phi}_j(\hat{\mathbf{x}}) \cdot |\mathbf{B}_T| \, d\hat{\mathbf{x}},$$

for all $j = 1, \ldots, N_T$.

- **The Neumann condition:**
  In this second integral of the linear form it is important realize that it have to be done separately to the ones of computing domain integrals for the matrices and the source terms, because it belongs to a boundary case. Note that we only need to compute this integral on the Neumann boundary, otherwise, this integral vanishes. Take into account also that if the boundary condition is an edge we will assume that F_T maps a curve γ in the physical space, while if it is a face F_T maps a surface.

$$\int_{\Gamma_N} g_N(\mathbf{x}) \cdot \phi_j(\mathbf{x})\, d\mathbf{x} = \sum_{\partial\hat{T} \in \hat{\Gamma}_N} \int_{\partial\hat{T}} g_N(F_T(\hat{\mathbf{x}})) \cdot \hat{\phi}_j(\hat{\mathbf{x}}) \cdot ||\mathbf{B}_T||\, d\hat{\mathbf{x}},$$

  for all ϕ_j that do not vanishes in Γ_N. Note that, in fact, we are parameterizing a boundary entity into the physical domain. Thus, the matrix $\mathbf{B}_T$ is rectangular and $||\mathbf{B}_T|| = |\gamma'|$ for curves and $||\mathbf{B}_T|| = |\mathbf{n}|$ (outwards normal vector) for surfaces.

In order to compute these integrals we approximate them using a quadrature formula.

- **Integration rule**
  In the method have been used high-order polynomials, for instance those called Legendre polynomials of degree n. They are defined by:

$$P_n(x) = \frac{1}{2^n!} \frac{d^n}{dx^n}(x^2 - 1)^n, \quad n = 0, 1, \ldots.$$

  Or taking the recurrence formula:

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x).$$

  To compute integrals of such polynomials we have to overtake that rarely may be obtained analytically, so we could use the Gauss-Legendre quadrature formula, which is able to integrate exactly polynomials of degree up to $2n-1$. In general this formula has the shape:

$$\int_{\hat{T}} \hat{\phi}(\hat{\mathbf{x}}) d\hat{\mathbf{x}} \approx \sum_{i=1}^{n_g} \hat{\omega}_i \hat{\phi}(\hat{\mathbf{b}}_i). \tag{11}$$

  Here $\hat{\mathbf{b}}_i$ are the Gauss points and $\hat{\omega}_i$ the Gauss weights.

Finally, we can explain how the matrix $\mathbf{A}$ is. What we do, in general, is to apply the quadrature formula introduced in (11).

$$a_{_T}(\phi_i, \phi_j) \approx \sum_{g_i}^{n_g} |\mathbf{B}_T| \cdot \mathbf{B}_T^{-t} \hat{\nabla} \hat{\phi}_i(\hat{\mathbf{x}}_{g_i}) \cdot k(F_T(\hat{\mathbf{x}}_{g_i})) \cdot \mathbf{B}_T^{-t} \hat{\nabla} \hat{\phi}_j(\hat{\mathbf{x}}_{g_i}) \cdot \hat{\omega}_{g_i},$$

$$\begin{aligned} L_{_T}(\phi_j) = f_{_T} - g_{_T} \approx & \sum_{g_i}^{n_g} |\mathbf{B}_T| \cdot f(F_T(\hat{\mathbf{x}}_{g_i})) \cdot (\phi_j \circ F_T)(\hat{\mathbf{x}}_{g_i}) \cdot \hat{\omega}_{g_i} - \\ & - \sum_{g_i}^{n_g} F_{T_{|\Gamma_N}}(\hat{\mathbf{x}}_{g_i}) \cdot (\phi_j \circ F_{T_{|\Gamma_N}})(\hat{\mathbf{x}}_{g_i}) \cdot ||\mathbf{B}_{T_{|\Gamma_N}}|| \cdot \hat{\omega}_{g_i}, \end{aligned}$$

where $\hat{\omega}_{g_i}$ denotes the Gauss weights and $\hat{\mathbf{x}}_{g_i}$ the Gauss points.

Through this methodology were computed the elemental contributions of the $\mathbf{A}$ matrix, which means that $\mathbf{A}$ will be constructed by adding all the elemental contributions, a_T. To this end we define an index mapping operator, denoted by $\mathbb{O}_T(\cdot)$, which maps the indexes of the corresponding local contributions of the element to the global indexes of the matrix $\mathbf{A}$. The resulting matrix and rhs. can be expressed as:

$$a_{_T}(\phi_i, \phi_j) = \mathbf{A}_T \Rightarrow \mathbf{A} = \sum_{T \in \mathcal{T}_h} \mathbb{O}_T(\mathbf{A}_T),$$

$$L_{_T}(\phi_j) = \mathbf{b}_T \Rightarrow \mathbf{b} = \sum_{T \in \mathcal{T}_h} \mathbb{O}_T(f_T) - \sum_{T \in \mathcal{T}_h} \mathbb{O}_T(g_{N_T}) - \sum_{i=n_h+1}^{N} g_{Di} a(\phi_i, \phi_j).$$

Note that the matrix is symmetric and positive definite. In addition, the matrix $\mathbf{A}$ is sparse, in the sense that there is a large number of null entries. Thus, the CSR sparse matrix storage to efficiently use the memory resources is used.

Only remains to say that once the assembly is done, there are good reenumeration algorithms, like the reverse Cuthill-McKee, which can minimize the band-width of the matrix, see [**9**].

For the interested reader, there can be found more detailed explanations about the theme exposed in this section in [**8, 12, 16**].

4. Static condensation Procedure

To reduce the size of the linear system to be solved we are going to use the static condensation procedure [**8, 12, 14**]. Notice that the solution obtained is the same and therefore, the methodology introduced here only reduces the computational costs.

First, we reorder the elemental nodes, in such a way that the inner ones come first than the boundary ones. Thus, we can write also the elemental unknown solution in terms of polynomial basis as before:

$$u_T = \sum_{k=1}^{N_T} u_k \phi_k = \sum_{k=1}^{n_T} u_k^i \phi_k^i + \sum_{k=n_T+1}^{N_T} u_k^b \phi_k^b.$$

The local solution is $u_T = u_k^b + u_k^i$, denoting by the superscripts b and i boundary and interior nodes of the element respectively. Taking from previous subsection 3.2 the local contributions and reordering them with this new numeration we can write a local system such that:

$$\mathbf{A}_T \mathbf{u}_T = \mathbf{b}_T \Leftrightarrow \left(\begin{array}{c|c} \mathbf{A}_T^{ii} & \mathbf{A}_T^{ib} \\ \hline \mathbf{A}_T^{bi} & \mathbf{A}_T^{bb} \end{array} \right) \left(\begin{array}{c} \mathbf{u}_T^i \\ \hline \mathbf{u}_T^b \end{array} \right) = \left(\begin{array}{c} \mathbf{b}_T^i \\ \hline \mathbf{b}_T^b \end{array} \right).$$

From which is obtained:

$$\mathbf{A}_T^{ii} \mathbf{u}_T^i + \mathbf{A}_T^{ib} \mathbf{u}_T^b = \mathbf{b}_T^i \Leftrightarrow \mathbf{u}_T^i = (\mathbf{A}_T^{ii})^{-1} (\mathbf{b}_T^i - \mathbf{A}_T^{ib} \mathbf{u}_T^b), \tag{12}$$

which essentially means that we are putting the unknowns belonging to the interior of an element in terms of the unknowns on the boundary of the same element. For the second row:

$$\mathbf{A}_T^{bi}\mathbf{u}_T^i + \mathbf{A}_T^{bb}\mathbf{u}_T^b = \mathbf{b}_T^b.$$

Now we substitute the expression of $\mathbf{u}_T^i$ to get:

$$\begin{aligned}
&\mathbf{A}_T^{bi}[(\mathbf{A}_T^{ii})^{-1}(\mathbf{b}_T^i - \mathbf{A}_T^{ib}\mathbf{u}_T^b)] + \mathbf{A}_T^{bb}\mathbf{u}_T^b = \mathbf{b}_T^b \Leftrightarrow \\
\Leftrightarrow &\mathbf{A}_T^{bi}(\mathbf{A}_T^{ii})^{-1}\mathbf{b}_T^i - \mathbf{A}_T^{bi}(\mathbf{A}_T^{ii})^{-1}\mathbf{A}_T^{ib}\mathbf{u}_T^b + \mathbf{A}_T^{bb}\mathbf{u}_T^b = \mathbf{b}_T^b \Leftrightarrow \\
\Leftrightarrow &[-\mathbf{A}_T^{bi}(\mathbf{A}_T^{ii})^{-1}\mathbf{A}_T^{ib} + \mathbf{A}_T^{bb}]\mathbf{u}_T^b = \mathbf{b}_T^b - \mathbf{A}_T^{bi}(\mathbf{A}_T^{ii})^{-1}\mathbf{b}_T^i.
\end{aligned}$$

Let us define now $\mathbf{A}_T^B := -\mathbf{A}_T^{bi}(\mathbf{A}_T^{ii})^{-1}\mathbf{A}_T^{ib} + \mathbf{A}_T^{bb}$, which is usually called block Schur complement. Besides it has good properties in the sense that if the original matrix is symmetric and positive definite this one is still symmetric and positive definite. At this point, the interior nodes of the element are not unknowns, the only ones remaining are those corresponding to boundary of the elements. Let us define also $\mathbf{b}_T^B := \mathbf{b}_T^b - \mathbf{A}_T^{bi}(\mathbf{A}_T^{ii})^{-1}\mathbf{b}_T^i$. With this procedure we can rewrite again the local system as:

$$\left(\begin{array}{c|c} \mathbf{A}_T^{ii} & \mathbf{A}_T^{ib} \\ \hline 0 & \mathbf{A}_T^B \end{array}\right)\left(\begin{array}{c} \mathbf{u}_T^i \\ \hline \mathbf{u}_T^b \end{array}\right) = \left(\begin{array}{c} \mathbf{b}_T^i \\ \hline \mathbf{b}_T^B \end{array}\right).$$

Finally, to complete the process of resolution, we conclude by bringing these new local contributions to the $\mathbf{A}^B$ matrix such as previously was done:

$$\mathbf{A}^B = \sum_{T\in\mathcal{T}_h} \mathbb{O}_T^B(\mathbf{A}_T^B),$$

$$\mathbf{b}^B = \sum_{T\in\mathcal{T}_h} \mathbb{O}_T^B(\mathbf{b}_T^B).$$

Notice that the system associated involves only boundary data of the elements, and therefore we can write the system as:

$$\mathbf{A}^B\mathbf{u}^b = \mathbf{b}^B.$$

The effort to compute the solution of this system is reduced due to the reduction in the number of unknowns. Although we have to implement a more complicated method, if we are dealing with high-order polynomials this work will be rewarded. When the degree of the polynomial basis is increased the number of nodes that we are using is directly related with the polynomial degree, and therefore with the pure interior ones. The important thing is that when the degree is increased there will be more inner-nodes per element and here is where we win with this new system.

Anyway, the problem is not solved yet. To finish with the writing of the whole solution we have to go back from this system taking with us the solution obtained. We put it in Equation (12) and thus we obtain the missing components of the solution and which correspond to $\mathbf{u}_T^i$. Notice that this last process is done element by element. However, this is not an inconvenient since these systems are very little and they are independent of each other. Using the previous methodology and

relying on Equation (8) it could be defined the new local problem:

$$u_h \in X_h(T) : \begin{cases} u_h = u_T^b, & \text{on } \partial T, \\ \int_T \nabla u_h \nabla v_h k dT = \int_T v_h f dT, & \forall v_h \in V_h^{\partial T}. \end{cases}$$

Thereby the unknown, u_T^i, can be written in terms of u_T^b.

The principal idea can be achieved from FIG. 6. The purple points are the ones correspond to elemental interior unknowns and they will not be unknowns. Therefore, we only have as unknowns the red ones because the greens are known boundary data.

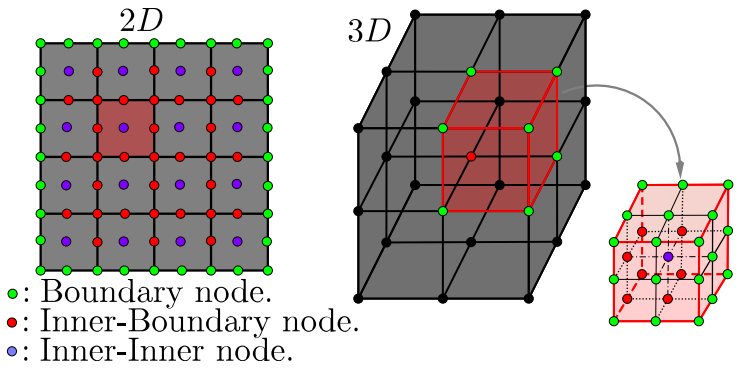

FIG. 6. Classification of nodes considering a particular case of shape functions of second degree (3 *nodes* × *dimension*). Right: $2D$ case. Left: $3D$ case with a detailed element clarifying the nodes.

To conclude, remark that we get rid of several unknowns in the same process of assembly, since this is the same as in the previous subsection. As soon as the degree increases, the number of pure interior nodes do it too and therefore the number of variables/equations to be solved that can be suppressed increases.

In comparison with the classical procedure shown in section 3, we are doing the same process but once we have the local contributions we reorder them with the purpose of supplying them in a simpler way to the global matrix $\mathbf{A}$. This allows to save computational time and memory storage, while the solution obtained is the same.

Chapter 3
Examples

Once introduced the methods above and implemented the program which do the whole computations of this CG FEM of Higher-Order, we test it with some examples in order to obtain some numerical results. The tests were performed in a HP ENVY Intel®Core (TM), with a i5-3317U CPU (1.7 GHz) and 6 GB RAM. For both we will consider the problem:

$$\begin{cases} -\nabla \cdot (k\nabla u) = f, & x \in \Omega, \\ u = g_D, & x \in \Gamma_D. \end{cases}$$

The domain will be in both cases a unity cube in $\mathbb{R}^3$ and there will not be Neumann conditions.

Example 9. For this first example we take:

$$u(x, y, z) = x^5 + y^5 + z^5,$$

and constant permeability $k = 1$. We first apply the classical method, without static condensation, and we execute it for degrees 1, 3, 5 and 7. Then we proceed doing the same with static condensation method. The details of the executions results are shown in TABLE 1. A fixed number of 64 elements is taken. For first order degree polynomials, since the number of inner nodes is zero, the number of unknowns remains equal for both cases. As the polynomial degree increases, begins to be more effective to use the static condensation methodology. For instance, for a polynomial degree 7, about 2/3 of unknowns are removed and the memory usage is reduced to 1/5 with respect to the Classical methodology. Finally, the execution time is also lower. Fig. 1 represent the approximate solution provided by the method implemented.

In the implementation, it is important to have a good measurer of the accuracy of the approximations. To this end, it is introduced here how the error is obtained for each case of our test problems, that is, the computation of the L^2-error.

$$E = ||u - u_h||_{L^2(\Omega)} = \sqrt{\int_\Omega (u - u_h)^2 d\Omega}.$$

Notice that due to u is known in these examples it can be computed exactly and since they are very small we present it here in a logarithmic scale. For both cases there is an important decrease from degree 1 to degree 3.

EXAMPLE 1		p=1	p=3	p=5	p=7
Classical approach	N. of Nodes	125	2197	9261	24389
	N. of Unkn.	27	1331	6859	19683
	Exec. T.	0.947999 s.	6.427 s.	121.559 s.	951.16 s.
	Matrix size	0.0016 MB.	1.1932 MB.	19.3112 MB.	126.985 MB.
Static condensation	N. of Nodes	125	2197	9261	24389
	N. of Unkn.	27	819	2763	5859
	Exec. T.	0.805999 s.	5.875999 s.	96.97699 s.	887.322 s.
	Matrix size	0.0016 MB.	0.7357 MB.	6.6980 MB.	27.6203 MB.

TABLE 1. Execution results for Example 1. A fixed number of 64 elements is taken.

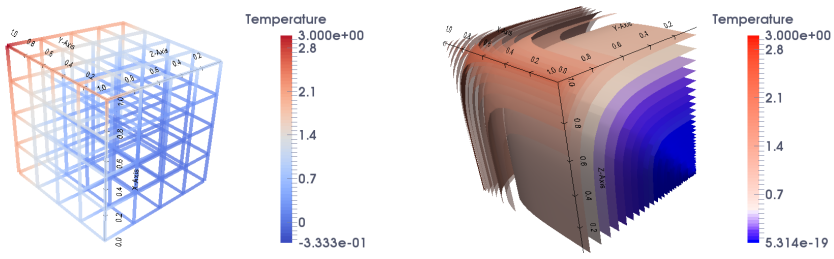

FIG. 1. Plots obtained with ParaView from the results computed with Python for the first example. From left to right: Representation of the solution in the edges of the whole decomposed domain and level surfaces using polynomials of degree 7. Note that the color scale is not the same.

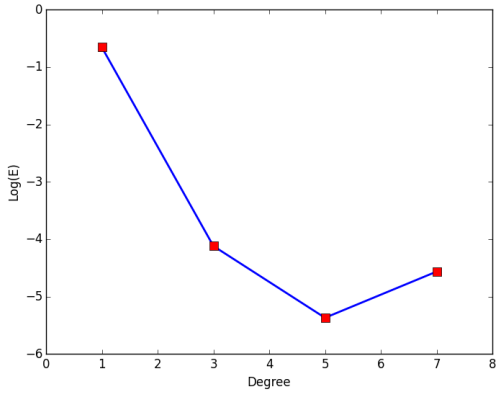

FIG. 2. Example 1. Numerical logarithmic error. There is a high decay of the error. Notice that already appears numerical error for degree greater than 5.

Finally, in order to give a last sample of the measure of computational costs reduction between these methodologies, the program is executed for this first example shown above with degrees 3 and 7. The representation below, in FIG. 3, shows in black points the non-empty entries of each matrix. The improvements of the results are obvious.

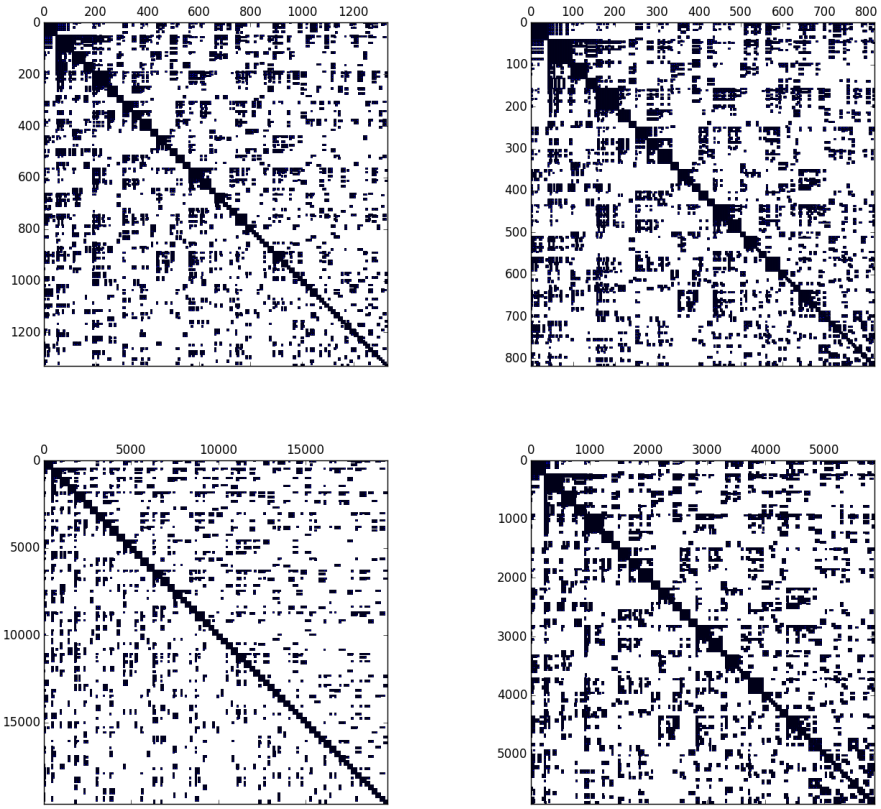

FIG. 3. Non-empty entries of 4 matrices. Above, for degree 3. Below, for degree 7. At the left with the classical implementation process described in Section 3 of Chapter 2. At the right with static condensation procedure described in Section 4 of Chapter 2.

EXAMPLE 10. For the second example we take:

$$u(x, y, z) = \sin(2\pi x)\sin(2\pi y)\sin(2\pi z)$$

and constant permeability $k = 1$. We apply the resolution methods just as in the previous example but now we take orders 1, 2 and 4. As previously it was done, TABLE 2 highlights the differences between the methods observed in the execution process. In this example instead of fixing the number of elements, we fix the number of nodes to 4913 which means that h/p is constant. As the polynomial degree is incremented, the number of elements is forced to decrease in order to have adjusted the number of nodes required to define the shape functions. Thus, although the differences are lower than in TABLE 1, it leads to analogous results with the advantage of taking the same number of nodes for $p = 1$ and $p = 4$. For $p = 4$ a half of memory is used and one half of the unknowns are left using static condensation procedure. Some plots of the solutions are shown in FIG. 4.

The error, following the previous rule, is shown in FIG. 5.

In a last study it will be taken a more complex test function, in order to appreciate the results better. For instance, a 3-wave sinus:

$$u(x, y, z) = \sin(6\pi x)\sin(6\pi y)\sin(6\pi z).$$

A very visual plot will represent this solution in FIG. 6 (approximated by the high-order CG FEM described). That is, a sinus with three waves in the unity cube.

EXAMPLE 2		p=1	p=2	p=4
Classical approach	N. of Elements	4096	512	64
	N. of unknowns	3375	3375	3375
	Execution Time	16.9539 s.	16.8589 s.	38.9670 s.
	Matrix size	0.267795 MB.	1.916728 MB.	5.655261 MB.
Static condensation	N. of Elements	4096	512	64
	N. of unknowns	3375	2863	1647
	Execution Time	16.3020 s.	15.1900 s.	30.8699 s.
	Matrix size	0.2677955 MB.	1.67692 MB.	2.5785 MB.

TABLE 2. Execution results for Example 2. Instead of fixing the number of elements, we fix the number of nodes to 4913 which means that h/p is constant.

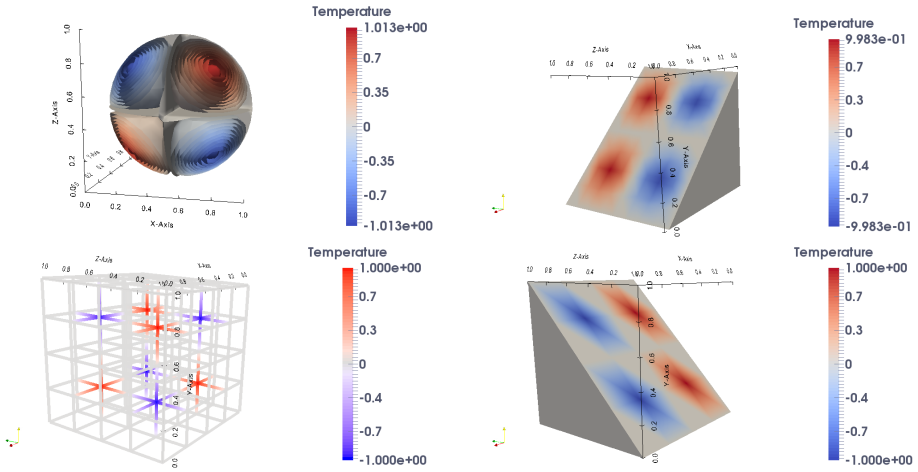

FIG. 4. Plots obtained with ParaView from the results computed with Python for the second example. We have take for these plots polynomial degree 4. From top to bottom and from left to right: Intersection of a sphere with 25 level surfaces for the sinus in the cube domain. Section of the solution domain. Representation of the solution over the edges showing the non-zero kernels. Another section, now from the other side.

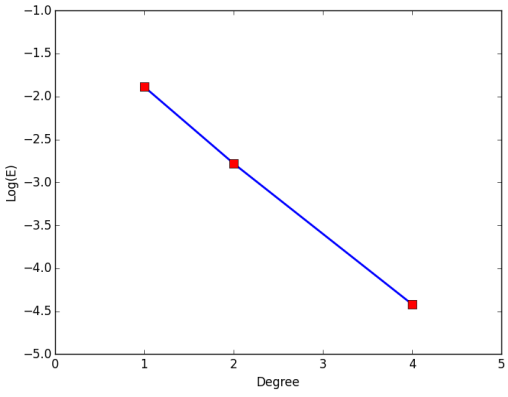

FIG. 5. Example 2. Numerical logarithmic error. Almost linear decay of the error.

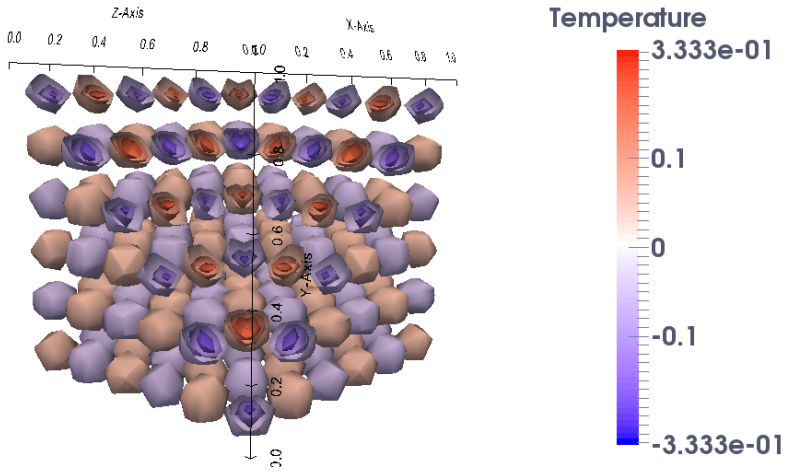

FIG. 6. This plot shows the second function of the second example. About 15 level surfaces are represented intersecting a wedge.

Chapter 4
Conclusions

In this document it was shown a classic formulation of the FEM for the Poisson problem. In addition, we have applied the static condensation technique to optimize the implementation which improves both computational time and memory storage. As we have seen, the higher the polynomial degree, the more benefits we obtain when applying the static condensation technique. Specifically, the number of nodes in the interior of the elements are written in terms of the nodes of the boundary of the elements. Thus, we only need to solve a linear system related to the unknowns of the nodes that belong to the boundary of the elements. Then, a local elemental process is performed in order to compute the solution in the whole domain. Notice that as the polynomial degree p increases, the number of nodes do it too. In particular, the number of purely inner nodes grow as p^n and the nodes belonging to borders do it as p^{n-1} where n denotes the space dimension. Thus, static condensation can potentially reduce the computational resources used to solve a problem. The results show that in general the implementation of the static condensation process reward the effort to implement when working with a moderate degree polynomials.

This work is open to future extensions or improvements which would be aimed at further optimizing the process. This would allows to solve larger and more complex problems with lower costs. We consider two main lines of work for the improvement of the project. The first concerns the language used for implementation. The current implementation is performed in Python [**1, 4**], it is an interpreted programming language and therefore uses computer resources during the execution of the program to interpret the code. This could be overcome by writing the code in a compiled language such as *C*, *C++* or *Fortran*. A second line of work, which could improve time costs, refers to the parallelization of code. The static condensation technique allows to parallelize the computation of the solution inside each element. Since the solution inside each element is computed as a post-process and each element is independent, it is straightforward to parallelize this part of the method [**14**]. Another way of parallelization, though more complex, would be to divide the problem into several sub-problems before calculating the solution.

References

[1] Bahit E *Python para principiantes*. Autoedición, 2012.

[2] Deville M O, Fischer P F and Mund E H. *High-order methods for incompressible fluid flow*, volume 9. Cambridge University Press, 2002.

[3] Discacciati M. Notes of the course: "*Numerical methods for Partial Differential equations*", *chapters* 1 *to* 3, 2014.

[4] Downey A, Elkner J and Meyers C. *How to Think Like a Computer Scientist. Learning with python*. Green Tea Press, $1^{\frac{st}{}}$ edition, 2002.

[5] Hesthaven J S and Warburton T. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Texts in Applied Mathematics. Springer, 2007.

[6] Huerta A, Angeloski A, Roca X, and Peraire J. *Efficiency of high-order elements for continuous and discontinuous Galerkin methods*. Int. J. Numer. Methods Eng., 96:529-560, 2013.

[7] Karniadakis G and Sherwin S. *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press, 2013.

[8] Kirby R M, Sherwin S J and Cockburn B. *To CG or to HDG: A comparative study*. Springer, 2011.

[9] Liu W-H and Sherman A H. *Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices.* SIAM, 1976.

[10] Löhner R. *Error and work estimates for high-order elements. Int. J. Numer. Methods Fluids*, 67(12):2184 2188, 2011.

[11] Löhner R. *Improved error and work estimates for high-order elements*. Int. J. Numer. Methods Fluids, 72:1207-1218, 2013.

[12] Sayas F-J. *A gentle introduction to the finite Element method*, 2008

[13] Schwab C. *p-and hp-finite element methods: Theory and applications in solid and fluid mechanics*. Clarendon Press Oxford, 1998.

[14] Šolín P, Segeth K and Ivo D. *Higher-order finite element methods*. Chapman & Hall/CRC, 2004.

[15] Szabo B A and Babuška I. *Finite Element Analysis*. John Wiley & Sons New York, 1991.

[16] Viaño Rey J M and Figueiredo J. *Implementaçao do método de elementos finitos. Notas*, 2000.

[17] Vos P E, Sherwin S J and Kirby R M. *From h to p efficiently: implementing finite and spectral/hp element methods to achieve optimal performance for low- and high-order discretizations. J. Comput. Phys.*, 229(13):5161-5181, 2010.

[18] Wang ZJ, Fidkowski K, Abgrall R, Bassi F, Caraeni D, Cary A, Deconinck H, Hartmann R, Hillewaert K, Huynh H T, Kroll N, May G, Persson P-O, van Leer B and Visbal M. *High-order CFD methods: current status and perspective*. International Journal for Numerical Methods in Fluids, 72(8):811-845, 2013.

[19] Yano M. *An optimization framework for adaptive higher-order discretizations of partial differential equations on anisotropic simplex meshes*. PhD thesis, Massachusetts Institute of Technology, 2012.