RAL-TR-2001-037

A Network Programming approach in Solving Darcy's equations by Mixed Finite-Element Methods

Mario Arioli[1] and Gianmarco Manzini[2]

ABSTRACT

We use the null space algorithm approach to solve the augmented systems produced by the mixed finite-element approximation of Darcy's laws. Taking into account the properties of the graph representing the triangulation, we adapt the null space technique proposed by Arioli and Baldini (2001), where an iterative-direct hybrid method is described. In particular, we use network programming techniques to identify the renumbering of the triangles and the edges, which enables us to compute the null space without floating-point operations. Moreover, we extensively take advantage of the graph properties to build efficient preconditioners for the iterative algorithm. Finally, we present the results of several numerical tests.

Keywords: Augmented systems, sparse matrices, mixed finite elements.

AMS(MOS) subject classifications: 65F05, 65F50.

Current reports available by anonymous ftp to ftp.numerical.rl.ac.uk in directory pub/reports.

[1] M.Arioli@rl.ac.uk, Rutherford Appleton Laboratory,

[2] Gianmarco.Manzini@ian.pv.cnr.it, IAN - CNR, via Ferrata 1, 27100 Pavia, Italy

The work of the first author was supported in part by EPSRC grant GR/R46441/01. The work of second author was supported by EPSRC grant GR/R46427/01.

Computational Science and Engineering Department
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

September 25, 2001

Contents

1 Introduction

The approximation of Darcy's Laws by Mixed Finite-Element techniques produces a finite-dimensional version of the continuous problem which is described by an augmented system. In this paper, we present an analysis of a Null Space method which uses a mixture of direct and iterative solvers applied to the solution of this special augmented system. The properties of this method, in the general case, have been studied by Arioli and Baldini (2001) where its backward stability is proved, when using finite-precision arithmetic, and where a review of the bibliography on the topic is also presented. Here, we will take advantage of network programming techniques for the design of a fast algorithm for the direct solver part and for the building of effective preconditioners. The relationship between the graph properties of the mesh and the augmented system has been pointed out by Alotto and Perugia (1999). Several authors used similar data structures and network techniques in a rather different context or for different purposes. Albanese and Rubinacci (1988), Biró, Preis, Vrisk, Richter and Ticar (1993), and Kettunen, Forsman and Bossavit (1999) have suggested similar techniques in the area of computational electro-magnetics for gauging vector potential formulations. In the field of computational fluid dynamics, analogous methods have been applied to the finite-difference method for the solution of Navier-Stokes equations by Amit, Hall and Porsching (1981) and Hall (1985). Finally, Arioli, Maryška, Rozložník and Tůma (2001*a*) studied a similar approach in the approximation of a 3-D Darcy's Law by Hybrid Finite-Element techniques.

In Section 2, we will briefly summarize the approximation process and describe the basic properties of the linear system and of the augmented matrix.

In Section 3, the Null Space algorithm and its algebraic properties are presented . The direct solver is based on the LU factorization of the submatrix of the augmented system which approximates the divergence operator div . We will see in Section 4, how the basic structures of the matrices involved are described in terms of graph theory and how the LU decomposition can be performed by Network Programming classical algorithms. In particular, we will use "Shortest Path Tree" algorithms to achieve a reliable fast decomposition. Furthermore, the same graph properties allow us to describe the block structure of the projected Hessian matrix on which we will apply the conjugate gradient algorithm. This will be used in Section 5 to develop effective preconditioners.

Finally, in Section 6, we show the results of the numerical tests that we conducted on selected experiments, and in Section 7 we give our conclusions.

In the following, we will denote by $E_1 \in \mathbb{R}^{n\times m}$ and $E_2 \in \mathbb{R}^{n\times(n-m)}$ $(m \le n)$ the matrices

$$E_1 = \begin{bmatrix} I_m \\ 0_{n-m,m} \end{bmatrix}, \text{ and } \quad E_2 = \begin{bmatrix} 0_{m,n-m} \\ I_{n-m} \end{bmatrix}.$$

Given a $n \times m$ matrix B of entries B_{ij} and a n-vector v of entries v_i, we will denote by $|B|$ and $|v|$ the matrix and the vector whose entries are the absolute values of the entries of B and v.

Finally, we assume that the computer arithmetic satisfies the standard model (Higham 1996):

$$fl(\alpha\Box\beta) = (\alpha\Box\beta)(1 + \delta(\Box, \alpha, \beta)) \ ; \quad |\delta(\Box, \alpha, \beta)| \le \varepsilon,$$

where $fl(\cdot)$ denotes the result of a floating-point computation, α and β are floating-point numbers, ε is the rounding unit and $\Box$ is one of $+ - */$.

2 The analytical problem and its approximation

2.1 Darcy's Law

Let us indicate with Ω the spatial domain of computation, where the equations are defined. Mathematically, Ω is a simply connected bounded polygonal domain in $\mathbb{R}^2$ (in 2-D), defined by a closed (1-D) surface Γ. Γ is usually the union of two parts Γ_D and Γ_N, where different boundary conditions are imposed, of Dirichlet and Neumann type respectively: $\Gamma = \Gamma_D \cup \Gamma_N$. The relationship between the pressure p (the total head) and the velocity field $\mathbf{v}$ (the visible effect) in ground-water flow can be expressed as a system of Darcy-type partial differential equations. Under the assumption that the soil matrix is incompressible (soil matrix characteristics - density, texture, specific storage, ...- are not functions of time, space, and pressure itself, as occurs in deformable porous media in which stress, strain, and fluids are strongly coupled), the saturated flow equations are given as:

$$\begin{aligned} \mathbf{v}(\mathbf{x}) &= -\mathcal{K}\,\mathrm{grad}\,p(\mathbf{x}),\ x \in \Omega && (1)\\ \mathrm{div}\,\mathbf{v}(\mathbf{x}) &= f(x),\ x \in \Omega && (2) \end{aligned}$$

In equations (1)-(2) $\mathcal{K}$ is the hydraulic conductivity tensor and $f(x)$ is a source-sink term. Equation (1) relates the vector field $\mathbf{v}$ to the scalar field p via the permeability tensor, which accounts for the soil characteristics. Equation (2) relates the divergence of $\mathbf{v}$ to the source-sink term $f(x)$. These equations are supplemented with a set of boundary conditions for both $\mathbf{v}$ and p:

$$\begin{aligned} p(\mathbf{x})|_{\Gamma_D} &= g_D \text{ on } \Gamma_D && (3)\\ \mathbf{v}\cdot\mathbf{n}|_{\Gamma_N} &= g_N \text{ on } \Gamma_N && (4) \end{aligned}$$

using two regular functions g_D and g_N for Dirichlet and Neumann conditions, and where $\mathbf{n}$ denotes the external normal to Γ. In the following, we assume that $g_N = 0$.

2.2 Mixed finite-element method for Darcy's law

The coupled system of equations (1)-(2) in the hydrostatic pressure p and the velocity field $\mathbf{v}$ has been approximated by a mixed finite-element approach. In this section, we shortly review some basic ideas underlying the numerical method, referring to the literature for a detailed exposition.

A weak formulation is formally obtained in a standard way by multiplying equation (1) by the test functions

$$\mathbf{w} \in \mathcal{V} = \{\mathbf{q}\,|\,\mathbf{q} \in \left(L^2(\Omega)\right)^2, \mathrm{div}\,\mathbf{q} \in L^2(\Omega), \mathbf{q}\cdot\mathbf{n}|_{\Gamma_N} = 0\},$$

and equation (2) by $\phi \in L^2(\Omega)$ and integrating by parts over the domain of computation. We refer to Brezzi and Fortin (1991) for the definition and the properties of the functional space $\mathcal{V}$.

The mixed weak formulation of the problem is:

find $\mathbf{v} \in \mathcal{V}$ and $p \in L^2(\Omega)$ such that

$$\left\{ \begin{array}{lll} \displaystyle\int_\Omega \mathcal{K}^{-1}\mathbf{v}\cdot\mathbf{w}\,d\mathbf{x} \quad - \int_\Omega p \operatorname{div}\mathbf{w}\,d\mathbf{x} & = -\displaystyle\int_{\Gamma_D} g_D \mathbf{w}\cdot\mathbf{n} ds & \forall \mathbf{w} \in \mathcal{V}\,, \\ & & \\ \displaystyle\int_\Omega (\operatorname{div}\mathbf{v})\phi\,d\mathbf{x} & = \displaystyle\int_\Omega f(x)\phi\,d\mathbf{x} & \forall \phi \in L^2(\Omega)\,. \end{array} \right. \tag{5}$$

The discrete counterpart of (5) can be introduced as follows.

Let $\mathfrak{T}_\mathfrak{h}$ be a family of triangulations of Ω, i.e. each $\mathfrak{T}_\mathfrak{h}$ is a set of disjoint triangles $\{T\}$ which covers Ω in such a way that no vertex of any triangle lies in the interior of an edge of another triangle. Let $\mathfrak{h} = \max\limits_{T \in \mathfrak{T}_\mathfrak{h}} \operatorname{diam}(T)$, we assume that $\mathfrak{T}_\mathfrak{h}$ is *regular* in the sense of Ciarlet (1978, page 132), i.e. triangles do not degenerate as $h \to 0$. Moreover, we assume that no triangle has a vertex on Γ_D and any other vertex on Γ_N, and that each triangle cannot have more than one edge lying on Γ.

Consider then the spaces

$$V_\mathfrak{h} = \left\{ \mathbf{w}(\mathbf{x}) : \Omega \to \mathbb{R}^2,\ \mathbf{w}(\mathbf{x})|_T = \alpha\mathbf{x} + \mathbf{z},\ \alpha \in \mathbb{R}\forall T \in \mathfrak{T}_h,\ \mathbf{w}\cdot\mathbf{n}|_{\Gamma_N} = 0 \right\},$$

and

$$Q_\mathfrak{h} = \{\phi(\mathbf{x}) : \Omega \to \mathbb{R},\ \phi(\mathbf{x})|_T = const,\ \forall T \in \mathfrak{T}_h\}\,.$$

The space $V_\mathfrak{h}$ is the usual Raviart-Thomas space (we refer to Brezzi and Fortin, 1991 for a detailed analysis), and $V_\mathfrak{h}$ and $Q_\mathfrak{h}$ are respectively dense in $\mathcal{V}$ and $L^2(\Omega)$. We will now describe the usual mixed finite-element basis for V_h. Let $\{e_j\}_{j=1,\ldots,N_e}$ be the set of the edges of $\mathfrak{T}_h$ where we exclude the edges lying on Γ_N. For each $T \in \mathfrak{T}_\mathfrak{h}$, we have the corresponding three edges $\{e_1, e_2, e_3\}$ and their external normals $\{\mathbf{n}_{e_1}, \mathbf{n}_{e_2}, \mathbf{n}_{e_3}\}$. We define the function $\mathbf{w}_{e_j} \in V_\mathfrak{h}$ on T by the relations (Brezzi and Fortin 1991):

$$\int_{e_i} \mathbf{w}_{e_j}\cdot\mathbf{n}_{e_i}\,ds = \left\{ \begin{array}{l} 1,\ i = j \\ \\ 0,\ i \neq j \end{array} \right. \tag{6}$$

Thus, we have N_e degrees of freedom, which can be interpreted as the 0^{th}-order momentum of the normal component of $\mathbf{v}$.

The set of basis functions $\{\phi_j\}_{j=1,N_T}$ for $Q_\mathfrak{h}$ is composed of functions such that $\phi_j = 1$ on T_j and $\phi_j = 0$ on $\Omega \setminus T_j$ and the number of degrees of freedom N_T is equal to the number of triangles which give the mesh.

The approximation of (5) results from substituting $\mathbf{v}$ and p by the expressions for $\mathbf{v}_\mathfrak{h}$ and $p_\mathfrak{h}$ as linear combination of the basis function $\left\{\mathbf{w}_{e_j}\right\}$ and of $\{\phi_i\}$:

$$p_\mathfrak{h} = \sum_{i=1}^{N_T} p_i\phi_i, \qquad\qquad \mathbf{v}_\mathfrak{h} = \sum_{j=1}^{N_e} u_j\mathbf{w}_{e_j}\,,$$

where $u = \{u_j\}$ and $p = \{p_i\}$ are the discrete unknown vectors associated with the velocity and the pressure respectively.

Therefore, the approximated version of the system (5) is equivalent to the following system of linear equations:

$$\begin{bmatrix} M & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix} \tag{7}$$

where M, A, q, and b are defined as follows

$$(M)_{e_ie_k} = \int_\Omega \mathcal{K}^{-1}\mathbf{w}_{e_i} \cdot \mathbf{w}_{e_k}\, d\mathbf{x}, \tag{8}$$

$$(A)_{e_ij} = -\int_\Omega \operatorname{div} \mathbf{w}_{e_i} \phi_j \, d\mathbf{x}, \tag{9}$$

$$(q)_{e_i} = \int_\Gamma g_D \mathbf{w}_{e_i} \cdot \mathbf{n}\, ds, \tag{10}$$

$$(b)_j = -\int_\Omega f(x)\phi_j \, d\mathbf{x}. \tag{11}$$

Let e_i be an internal edge, i.e. $e_i \cap \Gamma \neq e_i$, and $T'_{e_i} \subset \Omega$ and $T''_{e_i} \subset \Omega$ are the two triangles having e_i in common. First of all, we observe that $supp(\mathbf{w}_{e_i}) = T'_{e_i} \cup T''_{e_i}$. Then, it is straightforward to prove that the maximum number of nonzero entries in row e_i of M is 5 and these entries correspond to the edges which are identified by T'_{e_i} and T''_{e_i} and do not lie on Γ_N. Analogously, the nonzero entries in row e_i of A can only correspond to the triangles T'_{e_i} and T''_{e_i}. If $e_i \subset \Gamma_D$, we have only "one" triangle $T'_{e_i} \subset \Omega$ having e_i as an edge. Therefore, row e_i of A has only one nonzero entry which is the one relative to T'_{e_i}, and the row $M_{e_i\bullet}$ has only three nonzero entries corresponding to the three edges of T'_{e_i}. Furthermore, we have that the pattern of M is equal to the pattern of $|A||A|^T$, because the nonzero entries in row $M_{e_i\bullet}$ only match the nonzero entries of the edges of the triangles T'_{e_i} and T''_{e_i}.

Finally, we can fix the direction of the normal vector $\mathbf{n}_{e_i}$ for each e_i in the mesh, without loss of generality: the direction of the flux through the edge will be determined by sign (u_i).

Consequently, we have from (6) and (9), applying the Gauss theorem, that the nonzero entries of A are equal to ±1, and, if $e_i \cap \Gamma_D \neq e_i$, $A_{e_iT'_{e_i}} + A_{e_iT''_{e_i}} = 0$. Therefore, augmenting A with a vector r having nonzero entries only for the indices $e_i \subset \Gamma_D$ and such that $r_{e_i} + A_{e_iT_{e_i}} = 0$, we obtain the incidence matrix A' of an edge-triangle graph. The matrix A' is then "**totally unimodular**[3]" and has rank N_T (Murthy 1992). Moreover, because every submatrix obtained removing a column of A' has full rank (Murthy 1992, page 38), A has rank N_T.

3 Null Space Algorithms

To simplify the notation, let us denote N_e by n and N_T by m. The matrix $M \in \mathbb{R}^{n\times n}$ is a symmetric and positive semidefinite matrix, and $A \in \mathbb{R}^{n\times m}$ is a real full rank matrix. The augmented system (7) has a unique solution because one can show that $\mathrm{Ker}(A^T) \cap \mathrm{Ker}(M) = \{0\}$, (Brezzi and Fortin 1991).

In this section, we take into account the Null Space classical algorithm, which is described by Gill, Murray and Wright (1981), for the minimization of linearly constrained quadratic forms.

[3] A real matrix is totally unimodular if the determinant of every square submatrix of it is either 0, or ±1.

We choose a formulation of this algorithm which is based on the factorization of the augmented matrix.

In Section 2, we already stated that A is a full rank submatrix of a totally unimodular matrix of order $n \times (m+1)$, and its entries are either ± 1 or 0.

As a consequence of these properties, the LU factorization of A is obtainable without floating-point operations, through the permutation matrices P and Q:

$$PAQ = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} = LE_1 = \begin{bmatrix} L_1 & 0 \\ L_2 & I \end{bmatrix} E_1. \tag{12}$$

with L_1 a nonsingular lower triangular matrix. In Section 4, we will see how it is possible to compute P and Q using network programming algorithms. Moreover, without loss of generality, we can assume that all the diagonal entries in L_1 are equal to one and the entries outside the diagonal are zero or -1. This corresponds to a symmetric diagonal scaling of the permuted augmented system.

For the sake of simplicity, we will omit P and Q in the following and assume that M, A, q, and b have been consistently permuted. Let

$$\tilde{M} = L^{-1} M L^{-T}.$$

The augmented matrix $\mathfrak{A}$ can be factorized in the following way:

$$\mathfrak{A} = \begin{bmatrix} L & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \tilde{M}_{11} & \tilde{M}_{12} & I_m \\ \tilde{M}_{12}^T & \tilde{M}_{22} & 0 \\ I_m & 0 & 0 \end{bmatrix} \begin{bmatrix} L^T & 0 \\ 0 & I \end{bmatrix}.$$

Let $Z = L^{-T} E_2$. The matrix Z is a non orthonormal basis of the kernel of A^T.

On the basis of the previous discussion, we obtain the following algorithm:

NULL SPACE ALGORITHM:

$$\begin{aligned} h &= -L^{-1} q, \\ h_1 &= E_1^T h, \\ h_2 &= E_2^T h. \end{aligned}$$

Solve the block lower triangular system:

$$\begin{bmatrix} I_m & 0 & 0 \\ \tilde{M}_{12}^T & \tilde{M}_{22} & 0 \\ \tilde{M}_{11} & \tilde{M}_{12} & I_m \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} b \\ h_2 \\ h_1 \end{bmatrix}, \tag{13}$$

and let

$$\begin{aligned} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} &= L^{-T} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} L_1^{-T}(b - L_2^T u_2) \\ u_2 \end{bmatrix}, \\ p &= z_3. \end{aligned}$$

It follows directly from (12) that $u_2 = z_2$. Frequently, the product $L^{-1}ML^{-T}$ cannot be performed directly because the complexity would be too high ($\mathcal{O}(n^3)$) or because the resulting matrix would be fairly dense despite the sparsity of M. Nevertheless, in some cases the product can be performed successfully obtaining a sparse result. Alternatively, in solving the triangular system (13), we can perform the product of a submatrix of $\tilde{M}$ by a vector in the following way:

$$\tilde{M}_{ij}y = E_i^T L^{-1}(M(L^{-T}E_j y)), \text{ with } \quad (i,j) \; = \; (1,2). \tag{14}$$

This approach has the advantage of performing backward and forward substitution for triangular matrices, and a sparse matrix-vector product.

In the null space algorithm, we also need to invert $\tilde{M}_{22}$, the projected Hessian matrix. We have two alternative ways to proceed. If $n-m$ is small (the number of constraints is very close to the number of unknowns), or $\tilde{M}_{22}$ is still sparse, we can explicitly compute $\tilde{M}_{22}$ using the algorithm (14), and then solve the system $\tilde{M}_{22}u_2 = s$ using a Cholesky factorization. Otherwise, we can solve the linear system $\tilde{M}_{22}u_2 = s$ using a conjugate gradient algorithm (see Greenbaum, 1997) without explicitly computing $\tilde{M}_{22}$ and using (14) to perform the matrix by vector products. We observe that the matrix-vector product (14) is backward stable (Arioli and Baldini 2001). Moreover, we can use an iterative method, which incorporates a stopping criterion based on the *a posteriori* componentwise backward error theory (Higham 1996).

If we use the conjugate gradient method, it is quite natural to have a stopping criterion which takes advantage of the minimization property of this method. At each step j the conjugate gradient method minimizes the energy norm of the error $\delta u_2 = u_2 - \bar{u}_2^{(j)}$ on a Krylov space. The space $\mathbb{R}^{n-m}$ with the norm

$$\|y\|_{\tilde{M}_{22}} = (y^T\tilde{M}_{22}y)^{(1/2)}$$

induces on its dual space the dual norm

$$\|f\|_{\tilde{M}_{22}^{-1}} = (f^T\tilde{M}_{22}^{-1}f)^{(1/2)}.$$

Therefore, a stopping criterion such as the following:

$$\text{IF} \;\; \|\tilde{M}_{22}\overline{u}_2 - \tilde{h}\|_{\tilde{M}_{22}^{-1}} \le \eta\|\tilde{h}\|_{\tilde{M}_{22}^{-1}} \;\; \text{THEN STOP ,} \tag{15}$$

with $\eta < 1$ an *a priori* threshold fixed by the user and where $\tilde{h} = fl(h_2 - \tilde{M}_{12}^T\bar{z}_1)$, will guarantee (Arioli, Noulard and Russo 2001*b*) that the computed solution $\bar{u}_2$ satisfies the perturbed linear system:

$$\begin{aligned} \tilde{M}_{22}\overline{u}_2 &= \tilde{h} + f, \\ \|f\|_{\tilde{M}_{22}^{-1}} &\le \eta\|\tilde{h}\|_{\tilde{M}_{22}^{-1}}. \end{aligned}$$

The choice of η will depend on the properties of the problem that we want to solve, and, in the practical cases, η can frequently be much larger than ε.

Finally, because $Z^TE_2 = I$, we can project the error onto the null space as follows:

$$(E_2^T L^{-1}ML^{-T}E_2)\overline{u}_2 = \tilde{h} + E_2^T L^{-1}\delta q, \tag{16}$$

with

$$\delta q = E_2 f, \quad \|f\|_{\tilde{M}_{22}^{-1}} \le \eta\|\tilde{h}\|_{\tilde{M}_{22}^{-1}} \le \eta\|\overline{u}_2\|_{\tilde{M}_{22}} + \mathcal{O}(\eta^2) \tag{17}$$

Arioli and Baldini (2001) proved that the solution computed by a more general version of the Null Space algorithm of Section 3 is backward stable. In our specific case the theorem can be stated as follows.

Theorem 3.1 *Let $\bar{u}$ and $\bar{p}$ be the values of u and p, solutions of (7), computed with the* **Null Space Algorithm**. *If $2n\varepsilon \ll 1$ then there exist matrices $\delta M \in \mathbb{R}^{n\times n}$ and $\delta A_1,\ \delta A_2 \in \mathbb{R}^{n\times m}$ and the vector $\delta q \in \mathbb{R}^n$ such that*

$$\begin{bmatrix} M+\delta M & A+\delta A_1 \\ (A+\delta A_2)^T & 0 \end{bmatrix} \begin{bmatrix} \bar{u} \\ \bar{p} \end{bmatrix} = \begin{bmatrix} -(q+\delta q) \\ b \end{bmatrix}.$$

Furthermore, we have

$$\begin{aligned} |\delta A_1| &\leq c_1\varepsilon|A|, \\ |\delta A_2| &\leq c_1\varepsilon|A|. \end{aligned}$$

If we use a conjugate gradient solver with (15) and a threshold η:

$$|\delta M| \leq c_2\varepsilon\mathcal{H}(|M| + |\bar{L}||\tilde{M}||\bar{L}|^T)\mathcal{H}^T + \mathcal{O}(\varepsilon^2)),$$

where $\mathcal{H} = I + E_2E_2^T|\bar{L}^{-1}|E_1E_1^T$, and

$$\|\delta q\|_{E_2\tilde{M}_{22}^{-1}E_2^T} \leq \eta\|\overline{u}_2\|_{\tilde{M}_{22}} + \mathcal{O}(\eta^2).$$

If we denote by $A^- = E_1^TL^{-1}$ a pseudo-inverse of A ((Campbell and Meyer Jr 1979)), and by $\mathcal{P} = I - MZ\tilde{M}_{22}^{-1}Z^T$ the oblique projection onto $span(A)$ along $span(MZ)$ (Lancaster and Tismenetsky 1985), it is easy to verify by direct computation that:

$$\mathfrak{A}^{-1} = \begin{bmatrix} Z\tilde{M}_{22}^{-1}Z^T & \mathcal{P}^T(A^-)^T \\ A^-\mathcal{P} & -A^-\mathcal{P}M(A^-)^T \end{bmatrix}. \tag{18}$$

As normal, $\varepsilon << \eta$, and it is appropriate to analyse the influence of the perturbation δM on the error δu neglecting the part depending on ε. More precisely, we will assume that

$$\left\| \mathfrak{A}^{-1} \begin{bmatrix} -\delta A_1\overline{p} - \delta M\overline{u} \\ -\delta A_2^T\overline{u} \end{bmatrix} \right\|_\infty < \left\| \mathfrak{A}^{-1} \begin{bmatrix} -\delta q \\ 0 \end{bmatrix} \right\|_\infty.$$

Thus, we have from (16) and (17) :

$$\begin{bmatrix} \delta u \\ \delta p \end{bmatrix} = \begin{bmatrix} Z\tilde{M}_{22}^{-1}Z^TE_2f \\ A^-\mathcal{P}E_2f \end{bmatrix}.$$

First of all, we need to add, within the conjugate gradient algorithm, some tool for estimating the value $e_{\tilde{M}_{22}} = f^T\tilde{M}_{22}^{-1}f$. This can be achieved using a Gauss quadrature rule as proposed by Golub and Meurant (1997). In particular, this variant of the conjugate gradient algorithm produces a lower bound for $e_{\tilde{M}_{22}}$. As suggested by Golub and Meurant (1997) the Gauss quadrature based lower bound can be made reasonably close to the value of $e_{\tilde{M}_{22}}$ at the price of d additional steps of the conjugate gradient algorithm. Golub and Meurant (1997) indicated $d = 10$ as a successful compromise, and numerical experiments support this conclusion (Golub and Meurant 1997, Arioli and Baldini 2001).

Finally, following Arioli and Baldini (2001), we estimate $\tilde{h}^T\tilde{M}_{22}^{-1}\tilde{h}$, taking into account that

$$\tilde{h}^T\tilde{M}_{22}^{-1}\tilde{h} = \overline{u}_2^T\tilde{M}_{22}\overline{u}_2 + \mathcal{O}(\eta^2).$$

We replace $\overline{u}_2$ with its current evaluation at step j of the conjugate gradient algorithm if $e_{\tilde{M}_{22}} < \eta^2\tilde{h}^T\tilde{h}$.

As we bound the backward error f in the energy norm, it is natural to bound some appropriate energy norms of δu and δp. Arioli and Baldini (2001) suggested two natural choices

$$\begin{aligned} \|\delta u\|_M &= (\delta u^T M\delta u)^{1/2}, && (19)\\ \|\delta p\|_{A^TM^{-1}A} &= (\delta p^T A^T M^{-1}A\delta p)^{1/2}. && (20) \end{aligned}$$

Moreover, Arioli and Baldini (2001) proved that

$$\|\delta u\|_M = \|f\|_{\tilde{M}_{22}^{-1}} + \mathcal{O}(\varepsilon) \le \eta\|\overline{u}_2\|_{\tilde{M}_{22}}, \tag{21}$$

$$\|\delta p\|_{A^TM^{-1}A} \le \eta\|\overline{u}_2\|_{\tilde{M}_{22}}(\rho(\tilde{M}_{22}E_2^TM^{-1}E_2))^{1/2}, \tag{22}$$

where $\rho(B)$ is the spectral radius of the square matrix B.

4 Graph and Network properties

First of all, we recall some useful basic definitions relative to graph theory: further information can be found in Murthy (1992) and Tarjan (1983).

A graph $\mathcal{G} = \{\mathcal{N},\mathcal{A}\}$ is made up of a set of nodes $\mathcal{N} = \{\tau_i\}_{i=1,\dots,n_{\mathcal{N}}}$ and a set of arcs $\mathcal{A} = \{\alpha_j\}_{j=1,\dots,n_{\mathcal{A}}}$. An arc α_j is identified by a pair of nodes τ_i and τ_k: we will denote by $\{\tau_i,\tau_k\}$ an unordered pair and the corresponding undirected arc, and by $[\tau_i,\tau_k]$ an ordered pair and the corresponding directed arc. Either $\mathcal{G}$ is an undirected graph, in which case all the arcs are undirected, or $\mathcal{G}$ is a directed graph, in which case all the arcs are directed. We can convert every directed graph $\mathcal{G}$ to an undirected one called the *undirected version* of $\mathcal{G}$ by replacing each $[\tau_i,\tau_k]$ by $\{\tau_i,\tau_k\}$ and removing the duplicate arcs. Conversely, we can build the *directed version* of an undirected graph $\mathcal{G}$ by replacing every $\{\tau_i,\tau_k\}$ by the pair of arcs $[\tau_i,\tau_k]$ and $[\tau_k,\tau_i]$. In order to avoid repeating definitions, we will denote by (τ_k,τ_i) either an undirected arc $\{\tau_i,\tau_k\}$ or a directed arc $[\tau_i,\tau_k]$, using the context to resolve the ambiguity.

The nodes τ_i and τ_k in an undirected graph $\mathcal{G} = \{\mathcal{N},\mathcal{A}\}$ are *adjacent* if $\{\tau_i,\tau_k\} \in \mathcal{A}$, and we define the *adjacency* of τ_i by

$$Adj_{\tau_i} = \{\tau_j \,:\, \{\tau_i,\tau_j\} \in \mathcal{A}\}.$$

Analogously, in a directed graph $\mathcal{G} = \{\mathcal{N},\mathcal{A}\}$, we define

$$Adj_{\tau_i} = \{\tau_j \,:\, \text{if } [\tau_i,\tau_j] \in \mathcal{A} \text{ or } [\tau_j,\tau_i] \in \mathcal{A}\}.$$

We define the adjacency of an arc as follows:

$$Adj_{\{\tau_i,\tau_j\}} = \{\{\tau_i, \tau_k\} \in \mathcal{A}\} \cup \{\{\tau_j, \tau_k\} \in \mathcal{A}\}$$

for an undirected graph, and

$$Adj_{[\tau_i,\tau_j]} = \{[\tau_i, \tau_k] \in \mathcal{A}\,,\ [\tau_k, \tau_i] \in \mathcal{A}\} \cup \{[\tau_j, \tau_k] \in \mathcal{A}\,,\ [\tau_k, \tau_j] \in \mathcal{A}\}$$

for a directed graph.

A *path* in a graph from node τ_j to node τ_k is a list of nodes $[\tau_j = \tau_{j_1}, \tau_{j_2}, \ldots, \tau_{j_k} = \tau_k]$, such that $(\tau_{j_i}, \tau_{j_{i+1}})$ is an arc in the graph $\mathcal{G}$ for $i = 1, \ldots, k-1$. The path *contains* node τ_i for $i \in [1, \ldots, k]$ and arc (τ_i, τ_{i+1}) for $i \in [1, \ldots, k]$ and *avoids* all other nodes and arcs. Nodes τ_j and τ_k are the *ends* of the path. The path is *simple* if all its nodes are distinct. The path is a *cycle* if $k > 1$ and $\tau_j = \tau_k$ and a *simple cycle* if all its nodes are distinct. A graph without cycles is *acyclic.* If there is a path from node τ_w to node τ_v then τ_v is *reachable* from τ_w.

An undirected graph is *connected* if every node of its undirected version is reachable from every other node and *disconnected* otherwise. The maximal connected subgraphs of $\mathcal{G}$ are its *connected components.*

A *rooted tree* is an undirected graph that is connected and acyclic with a distinguished node τ_R, called *root.* A rooted tree with k nodes contains $k-1$ arcs and has a unique simple path from any node to any other. When appropriate we shall regard the arcs of a rooted tree as directed. A *spanning rooted tree* $\mathcal{T}_R = \{\mathcal{N}, \mathcal{A}_{\mathcal{T}_R}\}$ in $\mathcal{G}$ is a rooted tree which is a subgraph of $\mathcal{G}$ with $n_{\mathcal{N}}$ nodes. If τ_v and τ_w are nodes in $\mathcal{T}_R$ and τ_v is in the path from τ_R to τ_w with $\tau_w \neq \tau_v$ then τ_v is an *ancestor* of τ_w and τ_w is a *descendant* of τ_v. Moreover, if τ_v and τ_w are adjacent then τ_v is the *parent* of τ_w and τ_w is a *child* of τ_v. Every node, except the root, has only one parent, which will be denoted by $parent(\tau_v)$, and, moreover, it has zero or more children. A node with zero children is a *leaf.* We will call the arcs in $\mathcal{A}_{\mathcal{T}_R}$ *in-tree* and the arcs in $\mathcal{A} \setminus \mathcal{A}_{\mathcal{T}_R}$ *out-of-tree.* A *forest* is a node-disjoint collection of trees.

Finally, we define the $depth(\tau_i)$ of a node τ_i in a rooted tree recursively by $depth(\tau_R) = 0$ and $depth(\tau_i) = depth(parent(\tau_i)) + 1$. Similarly, we define the $height(\tau_i)$ as follows:

$$\begin{aligned} height(\tau_i) &= 0, \ \text{ if } \tau_i \text{ is a leaf,} \\ height(\tau_i) &= \max\{height(\tau_w)\ :\ \tau_w \text{ is a child of } \tau_i\} + 1, \ \text{otherwise.} \end{aligned}$$

The *subtree rooted at* node τ_i is the rooted tree consisting of the subgraph induced by the descendants of τ_i and having root in τ_i. The *nearest common ancestor* of two nodes is the deepest node that is an ancestor of both. A node τ_i is a *branching node* if the number of its children is greater than or equal to two. For each out-of-tree arc $[\tau_j, \tau_k]$, the *cycle of minimal length* or the *fundamental cycle* is the cycle composed by $[\tau_j, \tau_k]$ and the paths in $\mathcal{T}_R$ from τ_j and τ_k to their nearest common ancestor.

We can now associate with the triangulation $\mathfrak{T}_{\mathfrak{h}}$ a graph $\mathcal{G} = \{\mathcal{N}, \mathcal{A}\}$ as follows. Let $\Phi : \mathfrak{T}_{\mathfrak{h}} \to \mathcal{N}$ be a bijective function such that $\tau_i = \Phi(T_i)$: the arc $(\tau_i, \tau_j) \in \mathcal{A}$ if and only if the triangles T_i and T_j have a common edge e_ℓ. Furthermore, we add to $\mathcal{N}$ a node τ_R that represents $\mathbb{R}^2 \setminus \Omega$,

$$\mathcal{N} = \mathcal{N} \cup \{\tau_R\}\,,$$

and, for each triangle T_k having the edge e_ℓ lying on Γ_D, we add the arc $(\tau_R, \Phi(T_k))$. The incidence matrix A' of the graph $\mathcal{G}$ is a totally unimodular matrix (Murthy 1992, Tarjan 1983) and its rank is n. If we remove from A' the column corresponding to the root, we obtain the matrix A of problem (7). Moreover, every spanning tree of $\mathcal{G}$ with root in τ_R induces a partition of the rows of A in in-tree rows and out-of-tree rows. If we renumber the in-tree arcs first and the out-of-tree arcs last, we can permute the in-tree arcs and the nodes such that (Murthy 1992) the permuted matrix A has the form

$$PAQ = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} ,$$

where $L_1 \in \mathbb{R}^{n\times n}$ is a lower triangular and non singular matrix.

As the matrix A is totally unimodular then the matrix L_1^{-1} is also a matrix with entries $-1, 0, 1$. Moreover, the matrix $L_2L_1^{-1}$ has entries $-1, 0, 1$ and its rows correspond to the out-of-tree arcs. The number of nonzeros in one of its rows, will be the number of arcs in the cycle of minimal length which the corresponding out-of-tree arc forms with the in-tree arcs. We recall (see Section 3) that, without loss of generality, all the diagonal entries in L_1 can be chosen equal to 1 and, therefore, the entries outside the diagonal are 0 or -1. This signifies selecting the directions of the arcs in $\mathcal{A}_{\mathcal{T}_R}$ as $[\tau_i, parent(\tau_i)]$, $\forall \tau_i$. Given the out-of-tree arc $[\tau_j, \tau_k]$, the values of the nonzero entries in the corresponding row of $L_2L_1^{-1}$ will be 1 if both the nodes of the in-tree arc corresponding to the nonzero entry are ancestors of τ_k, and will be -1 if both the nodes of the in-tree arc corresponding to the nonzero entry are ancestors of τ_j.

We now give some basic results the proof of which is straightforward.

Lemma 4.1 *Let τ_v be a branching node with k children. The descendants of τ_v can be partitioned in k sets $\mathcal{N}_1, \dots, \mathcal{N}_k$ such that $\mathcal{N}_i \cap \mathcal{N}_j = \emptyset$, for $i \neq j$. Each $(\tau_i, \tau_j) \in \mathcal{A}$ with $\tau_i \in \mathcal{N}_i$ and $\tau_j \in \mathcal{N}_j$ is an out-of-tree arc.*

If we define the adjacency of a set of nodes $\mathcal{S}_{\mathcal{N}} \subset \mathcal{N}$ and the adjacency of a set of arcs $\mathcal{S}_{\mathcal{A}} \subset \mathcal{A}$ as follows:

$$Adj(\mathcal{S}_{\mathcal{N}}) = \bigcup_{\tau_i \in \mathcal{S}_{\mathcal{N}}} Adj_{\tau_i}, \;\; Adj(\mathcal{S}_{\mathcal{A}}) = \bigcup_{(\tau_i,\tau_j) \in \mathcal{S}_{\mathcal{A}}} Adj_{(\tau_i,\tau_j)} ,$$

we have the following Corollary.

Corollary 4.1 *Let τ_v be a branching node with k children and $\mathcal{N}_1, \dots, \mathcal{N}_k$ such that $\mathcal{N}_i \cap \mathcal{N}_j = \emptyset$, for $i \neq j$ be the partitioning of the descendants of τ_v. Let $(\tau_q, \tau_j) \in \mathcal{A}$ and $(\tau_p, \tau_i) \in \mathcal{A}$ be out-of-tree arcs, with $\tau_p, \tau_i \in \mathcal{N}_{\ell_1}$ and $\tau_q, \tau_j \in \mathcal{N}_{\ell_2}$. The minimal length circuits $\mathcal{C}_{(\tau_q,\tau_j)} \subset \mathcal{A}_{\mathcal{T}_R} \cup \{(\tau_q, \tau_j)\}$ and $\mathcal{C}_{(\tau_p,\tau_i)} \subset \mathcal{A}_{\mathcal{T}_R} \cup \{(\tau_p, \tau_i)\}$ are disjoint:*

$$\mathcal{C}_{(\tau_p,\tau_i)} \cap \mathcal{C}_{(\tau_q,\tau_j)} = \emptyset ,$$

and

$$\mathcal{C}_{(\tau_p,\tau_i)} \cap Adj(\mathcal{C}_{(\tau_q,\tau_j)}) = \emptyset, \quad Adj(\mathcal{C}_{(\tau_p,\tau_i)}) \cap \mathcal{C}_{(\tau_q,\tau_j)} = \emptyset .$$

Proof. From Lemma 4.1 the descendants of the children of the branching node τ_v form disjoint subtrees. If the two circuits $\mathcal{C}_{(\tau_q,\tau_j)}$ and $\mathcal{C}_{(\tau_p,\tau_i)}$ have an arc in common this would be simultaneously in both the subtrees and it would close a circuit on the ancestor τ_v. This is in contradiction with the definition of tree.

Similarly, if $\mathcal{C}_{(\tau_p,\tau_i)}$ and $Adj(\mathcal{C}_{(\tau_q,\tau_j)})$ have a common arc this must be an in-tree arc because it lies on $\mathcal{C}_{(\tau_p,\tau_i)} \cap \mathcal{A}_{\mathcal{T}_R}$. This is in contradiction with the results of Lemma 4.1 because this arc connects two disjoint sets. □

Finally, if the root τ_R of the spanning tree is a branching node with k children τ_i^D , $i = 1, \dots, k$, the subtrees having τ_i^D as roots form a forest. Therefore, the matrix L_1 can be permuted in block diagonal form with triangular diagonal blocks.

We refer to the work of Murthy (1992), Tarjan (1983), and Gallo and Pallottino (1986), for surveys of different algorithms for computing spanning trees. An optimal choice for the rooted spanning tree is the one minimizing the number of nonzero entries in the matrix $Z = L^{-T}E_2$ the columns of which span the null space of A^T. Deo, Prabhu and Krishnamoorthy (1982) proved that the equivalent problem of finding the tree for which the sum of all the lengths of the fundamental circuits is minimal, is an $\mathcal{NP}$-complete problem. Berry, Heath, Kaneko, Lawo, Plemmons and Waed (1985), Coleman and Pothen (1986), Coleman and Pothen (1987), Deo et al. (1982), Gilbert and Heath (1987), Itai and Rodeh (1978), and Pothen (1989) have proposed several algorithms which select a rooted spanning tree reducing or minimizing the number of nonzero entries in Z.

In this paper, we propose two different approaches based on the introduction of a function *cost* defined on each arc of the graph and describing some physical properties of the original problem.

From Corollary 4.1 it follows that a rooted spanning tree, having the largest possible number of branching nodes, normally has many disjoint circuits. The columns of Z corresponding to these disjoint circuits are structurally orthogonal, i.e. the scalar product of each pair is zero, independent of the values of the entries.

Moreover, we choose the function $cost : \mathcal{A} \to \mathbb{R}_+$ in the following way:

$$\forall \alpha \in \mathcal{A} \begin{cases} cost(\alpha) = 0 & \text{if } \exists i,\ \alpha = (\tau_R, \tau_i) \\ cost(\alpha) = M_{\alpha\alpha} & \text{otherwise} \end{cases} \tag{23}$$

Using the *cost* function, we can compute the spanning tree rooted in τ_R solving the Shortest Path Tree (SPT) problem (Gallo and Pallottino 1986) on the graph. In particular, we have chosen the to implement the heap version of the shortest path tree algorithm (see the SHEAP algorithm in Gallo and Pallottino, 1986).

The resulting spanning tree has an interesting interpretation: in the presence of islands of very low permeability in Ω, (i.e. the values of $\mathcal{K}^{-1}$ are very large there), the paths from the root to a node corresponding to a triangle lying outside the islands of low permeability, will be made of nodes corresponding to triangles also lying outside the islands. The path "circumnavigates" the islands. Therefore, the set of these paths identifies the lines where the flux will be bigger, with a reasonable approximation.

Owing to the fact that we chose the cost as zero for the arcs containing τ_R, both strategies will give a forest if the number of zero cost arcs is greater than one.

We observe that we do not need to build the matrix A explicitly: the tree (or forest) can be computed using the graph only. Moreover, the solution of the lower and of the upper triangular systems can be performed taking advantage of the *parent* function alone. This results in a very fast and potentially highly parallel algorithm.

For a problem with only Dirichlet conditions and an isotropic mesh (i.e. the number of triangles along each direction is independent from the direction), we can have a forest with a number of trees proportional to $\mathfrak{h}^{-1}$, and, therefore, the matrix L_1 has $\mathcal{O}(\mathfrak{h}^{-1})$ diagonal blocks of average size $\mathcal{O}(\mathfrak{h}^{-1})$.

Finally, we point out that most non-leaf nodes in the SPT have two children.

5 Preconditioning and quotient tree

In the presence of islands of very low permeability in Ω, the values of $\mathcal{K}$ can differ by many orders of magnitude in the different regions of the domain. In this case, the projected Hessian matrix $\tilde{M}_{22}$ has a condition number which is still very high. It is then necessary to speed up the convergence of the conjugate gradient by use of a preconditioner. Obviously, it is not usually practical to explicitly form $\tilde{M}_{22}$ to compute or identify any preconditioner.

In this section, we will show how we can compute the block structure of $\tilde{M}_{22}$ using only the tree and the graph information. We will then use the block structure to compute a preconditioner.

Denoting by H the matrix $L_2L_1^{-1}$, the projected Hessian matrix $\tilde{M}_{22}$ can be written as follows:

$$\tilde{M}_{22} = Z^T M Z = M_{22} + HM_{11}H^T - (M_{21}H^T + HM_{12}). \tag{24}$$

The row and column indices of the matrix $\tilde{M}_{22}$ correspond to the out-of-tree arcs. Moreover, we recall that the matrix H has entries $-1, 0, 1$, its row indices correspond to the out-of-tree arcs, and the number of nonzeros in one of its rows will be the number of arcs in the cycle of minimal length which the corresponding out-of-tree arc forms with the in-tree arcs.

Lemma 5.1 *Let α and β be two out-of-tree arcs, and $\mathcal{C}_\alpha \subset \mathcal{A}_{\mathcal{T}_R} \cup \alpha$ and $\mathcal{C}_\beta \subset \mathcal{A}_{\mathcal{T}_R} \cup \beta$ be their corresponding circuits of minimal length, then*

$$M_{\alpha\beta} \neq 0 \iff \alpha, \beta \in Adj_\alpha \cap Adj_\beta \tag{25}$$

$$\mathcal{C}_\alpha \cap \mathcal{C}_\beta \neq \emptyset \iff \left\{ \begin{array}{c} \mathcal{C}_\alpha \cap Adj(\mathcal{C}_\beta) \neq \emptyset \\ \mathcal{C}_\beta \cap Adj(\mathcal{C}_\alpha) \neq \emptyset \end{array} \right\} \tag{26}$$

Proof. Because the orientation of the arcs in the graph is arbitrary and will be determined by the sign of the solution, we will use the undirected version of the graph. Let $\alpha = (\tau_v, \tau_w)$ and $\beta = (\tau_x, \tau_y)$, where we have $\tau_v = \Phi(T_1)$, $\tau_w = \Phi(T_2)$, $\tau_x = \Phi(T_3)$, and $\tau_y = \Phi(T_4)$. Moreover, α corresponds uniquely to the common edge e_{12} between the triangles T_1 and T_2, and β corresponds uniquely to the common edge e_{34} between the triangles T_3 and T_4, From (8), we have that $M_{\alpha\beta} \neq 0$ if and only if $supp(\mathbf{w}_{e_{12}}) \cap supp(\mathbf{w}_{e_{34}}) \neq \emptyset$. Since $supp(\mathbf{w}_{e_{12}}) = T_1 \cup T_2$ and $supp(\mathbf{w}_{e_{34}}) = T_3 \cup T_4$, then $M_{\alpha\beta} \neq 0$ if and only if the two supports have a triangle in common. Assuming, without loss of generality, that $T_1 = T_3$ is this common triangle, (25) follows from the definition of *Adj*.

In (26) the $\Rightarrow$ is trivial.

We give a proof *ab absurdo* for the $\Leftarrow$.

First of all, we note that the cardinal number of Adj_{τ_j} is less or equal than 3, $\forall \tau_j \in \mathcal{A}$ (a triangle has three edges).

If we assume that $\mathcal{C}_\alpha \cap Adj(\mathcal{C}_\beta) \neq \emptyset$, $\mathcal{C}_\beta \cap Adj(\mathcal{C}_\alpha) \neq \emptyset$ and $\mathcal{C}_\alpha \cap \mathcal{C}_\beta = \emptyset$, there must exist a node τ_j such that τ_j belongs to both the paths in the tree linking τ_v to τ_w and τ_x to τ_y respectively. Because each node in the path has only one *parent* and one *child* , the Adj_{τ_j} contains the *parent* and the *child* of the first path and those of the second path. Because we assumed that $\mathcal{C}_\alpha \cap \mathcal{C}_\beta = \emptyset$ the cardinal number of Adj_{τ_j} would be 4 which is in contradiction with the upper bound on the cardinal number. □

Thus, from (24) and the previous Lemma 5.1, we have the following Corollary.

Corollary 5.1 *Let α and β be two out-of-tree arcs, and $\mathcal{C}_\alpha \subset \mathcal{A}_{\mathcal{T}_R} \cup \alpha$ and $\mathcal{C}_\beta \subset \mathcal{A}_{\mathcal{T}_R} \cup \beta$ be their corresponding circuits of minimal length, then*

$$\begin{aligned}\tilde{M}_{\alpha\beta} = M_{\alpha\beta} + &\sum_{\gamma \in \mathcal{C}_\alpha \cap Adj(\mathcal{C}_\beta)} \sum_{\rho \in \mathcal{C}_\beta \cap Adj(\mathcal{C}_\alpha)} H_{\gamma\alpha} H_{\rho\beta} M_{\gamma\rho} \\ &- \sum_{\gamma \in \mathcal{C}_\alpha \cap Adj_\beta} H_{\gamma\alpha} M_{\gamma\beta} - \sum_{\gamma \in \mathcal{C}_\beta \cap Adj_\alpha} H_{\gamma\beta} M_{\beta\gamma}\end{aligned}$$

and

$$\mathcal{C}_\alpha \cap \mathcal{C}_\beta = \emptyset \;\Rightarrow\; \tilde{M}_{\alpha\beta} = 0 \tag{27}$$

Proof. The first part of the Corollary follows directly from the expansion of (24). The implication (27) follows from Lemma 5.1 taking into account that $\mathcal{C}_\alpha \cap Adj(\mathcal{C}_\beta) \supseteq \mathcal{C}_\alpha \cap Adj_\beta$, $\mathcal{C}_\beta \cap Adj(\mathcal{C}_\alpha) \supseteq \mathcal{C}_\beta \cap Adj_\alpha$, and $Adj_\alpha \cap Adj_\beta \neq \emptyset \Rightarrow \mathcal{C}_\alpha \cap \mathcal{C}_\beta \neq \emptyset$. □

Corollary 5.1 gives the complete description of the pattern of $\tilde{M}_{22}$ in terms of the rooted spanning tree. However, we observe that the results (26) and (27) rely on the 2-D structure of the mesh and they cannot be generalized to a mesh in 3-D.

In the second part of this section, we build the block structure of $\tilde{M}_{22}$ using the Quotient Tree concept, without explicitly forming the matrix $\tilde{M}_{22}$.

In the following, we process the root node separately from the other nodes. The root node will always be numbered by 0, and if it is a branching node with k children the tree will contain k subtrees directly linked to the root. Given a rooted spanning tree $\mathcal{T}_R = \{\mathcal{N}, \mathcal{A}_{\mathcal{T}_R}\}$, let $\mathcal{B} \subseteq \mathcal{N} \setminus \{0\}$ be the set of the branching nodes in $\mathcal{T}_R$, and let $\mathcal{L} \subseteq \mathcal{N} \setminus \{0\}$ be the set of the leaves in $\mathcal{T}_R$. We define the set

$$\mathcal{Y} = \mathcal{B} \cup \mathcal{L} = \{\tau_1, \ldots, \tau_k\}\,.$$

If $\mathcal{Y} = \mathcal{N} \setminus \{0\}$, then $\mathcal{T}_R$ is a binary tree. Otherwise, we can compute the following paths:

$$\begin{aligned}&\forall \tau_i \in \mathcal{Y} \\ &\qquad \wp_{\tau_i} = \{\tau_i, parent(\tau_i), parent(parent(\tau_i)), \ldots, parent^k(\tau_i)\} \text{ and } \wp_{\tau_i} \cap \mathcal{Y} = \{\tau_i\}.\end{aligned}$$

The path $\wp_{\tau_i}$ connects τ_i to all its ancestors which are not branching nodes, and it can contain τ_i alone.

The set $\mathfrak{P} = \{\wp_{\tau_1}, \ldots, \wp_{\tau_\ell}\} \cup \{0\}$ is a partition of $\mathcal{N}$:

$$\wp_{\tau_j} \cap \wp_{\tau_i} = \emptyset, \; \bigcup_{i=1}^{\ell} \wp_{\tau_i} = \mathcal{N}\,.$$

Therefore, we can build the quotient tree

$$\mathcal{T}/\mathfrak{P} = \{\mathfrak{P}, \mathfrak{E}_{\mathcal{T}}\}, \quad (\wp_{\tau_i}, \wp_{\tau_j}) \in \mathfrak{E}_{\mathcal{T}} \Longleftrightarrow Adj\,|_{\mathcal{T}}\,(\wp_{\tau_i}) \cap \wp_{\tau_j} \neq \emptyset,$$

and the quotient graph

$$\mathcal{G}/\mathfrak{P} = \{\mathfrak{P}, \mathfrak{E}_{\mathcal{G}}\}, \quad (\wp_{\tau_i}, \wp_{\tau_j}) \in \mathfrak{E}_{\mathcal{G}} \Longleftrightarrow Adj(\wp_{\tau_i}) \cap \wp_{\tau_j} \neq \emptyset,$$

where $Adj\,|_{\mathcal{T}}$ is the restriction of the Adj operator to the graph $\mathcal{T}$.

For the sake of clarity in the following, we will call the quotient tree nodes Q-nodes. The root of the quotient tree is still the node 0, and each subtree rooted at its children has a binary quotient tree.

5.1 Data structures and separators

We use a data structure for the representation of the graph $\mathcal{G}$ which is based on the matrix A. We assume that we have access to both row and column entries of A by the compressed sparse collection of row and column data structures as they are described by Duff, Erisman and Reid (1989). Every time we renumber the nodes in the graph, we assume that the data structure describing the rows of A will be consistently updated. The use of the double representation of A facilitates and speeds up the implementation of all the algorithms we will use in the following. Moreover, in our specific case where the matrix A is the incidence matrix of a graph representing a bidimensional mesh, the row and column compressed data structure can be easily implemented using vector arrays as follows: Each column of A has at most 3 nonzero entries, then, we store for each node (i.e. triangle) i the 3 arcs (i.e. edges) indices consecutively in position $i,\ i+1,\ i+2$. If the node corresponds to a triangle having an edge on the Neumann boundary, we explicitly store a 0 which means that that triangle has the root as a neighbour. Analogously, because each row j of A has only 2 nonzero entries, we store the 2 nodes (i.e. triangles) describing the arc j in position j and $j+1$ of a vector. Again, we need to explicitly store some 0 values for the Dirichlet edges. The overhead of the explicit storage of the 0 is less than the one we will have for storing the pointers within the classical row and column compressed forms because the number of the boundary edges is of the order of the square root of the number of triangles in the mesh.

A tree can be easily described by using a vector where in the entry τ_i we store $parent(\tau_i)$. In our implementation, we use a slightly more complex data structure. For each node τ_i, we know the $parent(\tau_i)$ and the list of its children. Descending the tree with a *depth first search*, we renumber the nodes and we build the chains. Therefore, we associate with each node the following objects:

- parent,
- children list,
- chain index,
- depth.

In Figure1, we give a simple example of a tree and, in Table 1, we list the labels of each node. It is relevant to observe that the reordering obtained by the depth first search of $\mathcal{T}$ renumbers the

Node index	Parent	Children list	Chain Index	depth
0	0	1,21,31	0	0
1	0	2	1	1
2	1	3	1	2
3	2	4	1	3
4	3	5,10	1	4
5	4	6	2	5
6	5	7	2	6
7	6	8	2	7
8	7	9	2	8
9	8		2	9
10	4	11	3	5
11	10	12	3	6
12	11	13	3	7
13	12	14,17	3	8
14	13	15	4	9
15	14	16	4	10
16	15		4	11
17	13	18	5	9
18	17	19	5	10
19	18	20	5	11
20	19		5	12
21	0	22	6	1
22	21	23	6	2
23	22	24	6	3
24	23	25,28	6	4
25	24	26	7	5
26	25	27	7	6
27	26		7	7
28	24	29	8	5
29	28	30	8	6
30	29		8	7
31	0	32	9	1
32	31	33	9	2
33	32		9	3

Table 1: Labels of nodes in the tree of Figure 1.

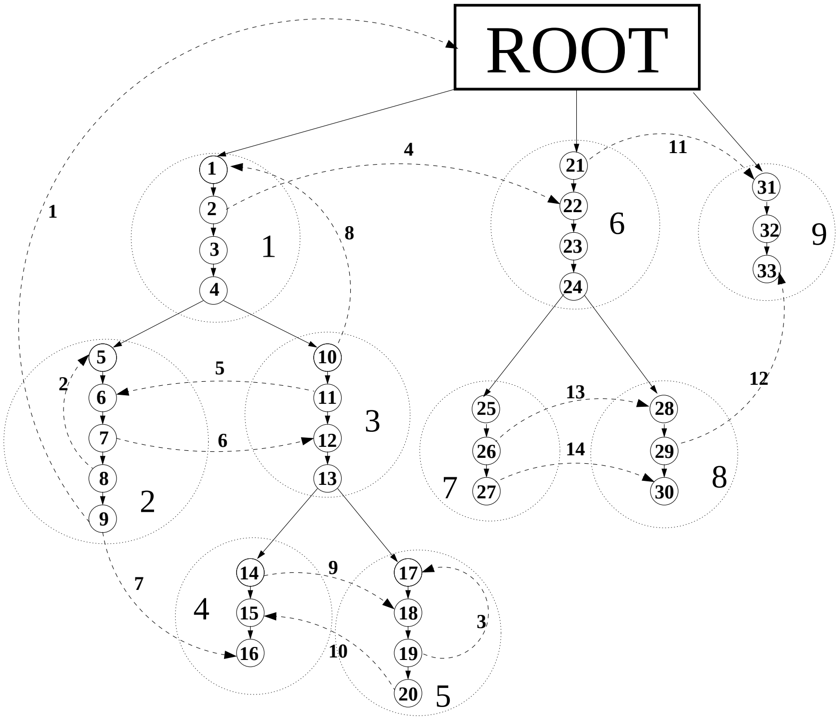

Figure 1: Example

nodes such that the nodes forming a chain are consecutive. Therefore, we can directly access the list using vector arrays.

After the identification of the chains, we build the quotient tree, and, descending the quotient tree $\mathcal{T}/\mathfrak{P}$ with a *depth first search*, we renumber the nodes and build its data structure. In the new data structure, we associate with each Q-node the following objects:

- Q-parent in $\mathcal{T}/\mathfrak{P}$,
- first and last node in $\mathcal{T}$ of the chain corresponding to the Q-node,
- depth in $\mathcal{T}/\mathfrak{P}$,
- Q-last, the last Q-node of the subtree rooted in the current Q-node,
- Q-star, the list of the out-of-tree arcs in $\mathcal{G}$ which have one extreme belonging to the Q-node,
- Q-children, the list of the Q-nodes children of the Q-node.

In Figure 2, we give the quotient tree relative to the example of Figure 1, and in Table 2, we list the labels of each Q-node.

Taking advantage of the data structures described above, we can order the out-of-tree arcs in the following way. Firstly, we identify the Q-nodes which are children of the external root (node 0), and the subtrees rooted in each of these Q-nodes. Then, we separate each of the subtrees

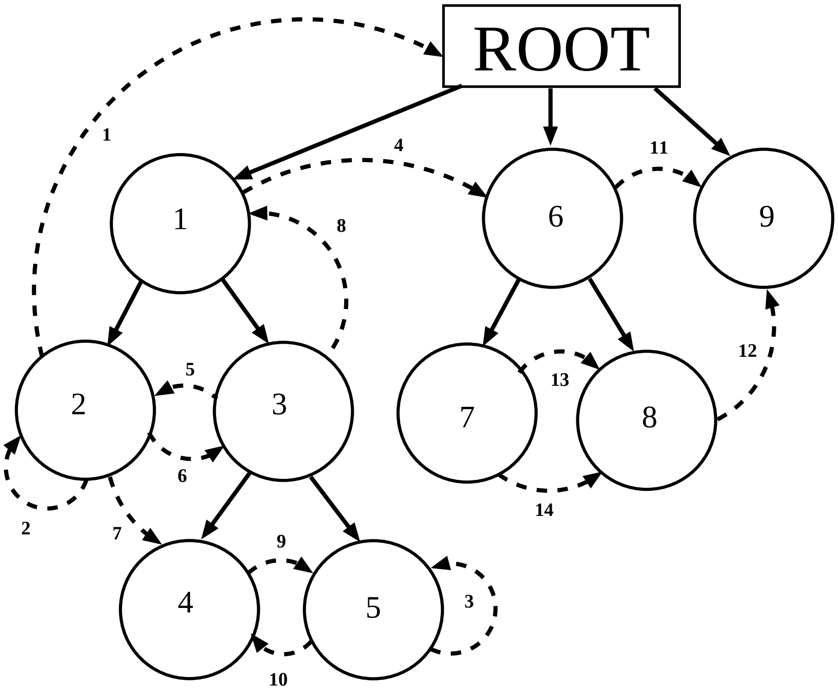

Figure 2: Quotient tree relative to the tree of Figure 1

Q-node	Q-parent	First, last node of chain	Depth	Q-last	Q-star	Q-children
1	0	1,4	1	5	4,8	2,3
2	1	5,9	2	2	1,2,5,6,7	
3	1	10,13	2	5	5,6,8	4,5
4	3	14,16	3	4	7,9,10	
5	3	17,20	3	5	3,9,10	
6	0	21,24	1	8	4,11	7,8
7	6	25,27	2	7	13,14	
8	6	28,30	2	8	12,13,14	
9	0	31,33	1	9	11,12	

Table 2: Labels of the Q-nodes in Figure 2.

from the others marking the out-of-tree edges that connect it to the others. The out-of-tree arcs lying within one of these subtrees cannot have fundamental cycles with the out-of-tree arcs lying within one of the others, because of Corollary 4.1. This corresponds to a one-way dissection applied to $\tilde{M}_{22}$. Then, within each of the subtrees, we seek the out-of-tree arcs that separate the two subtrees rooted in the Q-nodes children of the Q-node root of the subtree containing both of them. This is equivalent to a nested dissection strategy applied to one of the diagonal blocks resulting from the previous one-way dissection phase.

Before starting the one-way phase, we visit the tree and identify in each Q-node the out-of-tree arcs that are within the Q-node and the arcs directly linking the external root (node 0) to a Q-node. These arcs are stored in *sep.list* which is a queue data structure. The phase relative to the one-way dissection can be described by the following algorithm:

Algorithm 5.1

```
procedure root.sep.count(sep.size)
        for each $i \in children(0)$ do
                $Q_i$ = chain(i); sep.size($Q_i$) = 0 ;
                for $Q_k$ = $Q_i$:Q-last($Q_i$), do
                    for each $\ell = (p,q) \in$ Q-star($Q_k$) with chain(p)= $Q_k$ do
                            if $q >$ last(Q-last($Q_i$)) or $q <$ first($Q_i$) then
                              sep.size($Q_i$) = sep.size($Q_i$) + 1;
                            end if;
                    end for each;
                end for;
        end for each;
end procedure.

procedure sep.tree (sep.list,sep.size,$Q_{root}$,mask)
        for $Q_k$ = $Q_{root}$:Q-last($Q_{root}$), do
            for each $\ell = (p,q) \in$ Q-star($Q_k$) with chain(p)= $Q_k$ do
                    if mask($\ell$) = 0 and {$q >$ last(Q-last($Q_i$)) or $q <$ first($Q_i$)} then
                      insert $\ell$ in sep.list ;
                      mask($\ell$) = 1 ;
                      sep.size = sep.size + 1;
                    end if;
            end for each;
        end for;
end procedure.

procedure one.way.dissection(sep.list,sep.size,mask)
        root.sep.count(sep.size)
        sort Q-children($Q_{root}$) in decreasing order of sep.size ;
        for each child $\in$ Q-children($Q_{root}$) do
                sep.tree (sep.list, sep.size(child), child, mask);
        end for each;
end procedure.
```

Because each out-of-tree arc can be counted only twice, the complexity of Algorithm 5.1 is $\mathcal{O}(2\xi)$, where ξ is the number of nonzero entries in the submatrix L_2 and is equal to twice the number of the out-of-tree arcs.

From Corollary 4.1, Corollary 5.1, and Lemma 5.1 it follows that each block i, of size $sep.size(i)$, contains out-of-tree arcs that have fundamental cycles intersecting each other. Therefore, each of these blocks corresponds to a full diagonal block in $\tilde{M}_{22}$. The order of the diagonal block is equal to the corresponding value of *sep.size.* The out-of-tree arcs in *sep.list* form the external *separator.* The external separator identifies a curve on the mesh. Thus, because the graph is planar, the size χ of the external separator will be $\mathcal{O}(\sqrt{m})$ (m is the number of triangles in the mesh).

Finally, we renumber the out-of-tree arcs, such that the arcs in the external separator are the last ones and the arcs lying within a descendant subtree of root are consecutive. This is equivalent to permute the rows of the matrix L_2 so that we have a block diagonal submatrix followed by a block of rows connecting the previous diagonal blocks.

In Figure 3 (a) and (b), we show the results of the one-way dissection on the matrix A when root has only 3 descendant subtrees. We observe that each subtree is now disconnected from the others. We can now perform the nested dissection phase on each of the subtrees which are rooted at the children of the root. Each of these subtrees is a binary tree and it can be identified by the Q-node on which is rooted. Firstly, we visit each subtree and we reorder the Q-children list of each Q-node such that the first child is the root of the subtree with the least number of nodes, and the other children are sorted in increasing order with respect to the number of nodes in their respective subtrees. In the following Algorithm 5.2, we denote by Q_{root} the root of one of the Q-subtrees obtained from the one-way dissection phase. Moreover, we label each out-of-tree arc ℓ with the quotient tree ancestor *QT-ancestor(ℓ)*, the Q-node closing the fundamental cycle of ℓ in the spanning tree.

By means of *QT-ancestor* and by the data structure of the spanning tree, we can implicitly describe the structure of the null space matrix Z.

Algorithm 5.2

```
procedure nested.sep(sep.list, sep.size, $Q_{root}$, mask, QT-ancestor)
          if Q-children($Q_{root}$) $\neq \emptyset$ then
             $Q_1$ head of Q-children($Q_{root}$) list;
             $Q_2$ tail of Q-children($Q_{root}$) list;
             sep.tree(sep.list, sep.size($Q_1$), $Q_1$, mask);
             sep.size($Q_2$) = 0;
             for each $\ell \in$ Q-star($Q_{root}$) do
                       if mask($\ell$) = 0 then
                          insert $\ell$ in sep.list ;
                          mask($\ell$) = 1 ;
                          sep.size($Q_2$) = sep.size($Q_2$) +1;
                          QT-ancestor($\ell$) = $Q_{root}$;
                       end if
             end for each;
             nested.sep(sep.list, sep.size, $Q_1$, mask);
```

```
            nested.sep(sep.list, sep.size, $Q_2$, mask);
        end if;
end procedure
```

Algorithm 5.2 takes advantage of the binary structure of the quotient subtree and of the reordering of the Q-children list whose first entry has the least number of descendants.

These two properties allow the possibility of separating the two subtrees rooted in the children of the current Q_{root} visiting only one of them and the Q-star of Q_{root}.

Moreover, at each recursion we visit and separate a subtree which, in the worst case, has half of the nodes of the tree in it. Let N_Q be the number of nodes in one of the subtrees obtained after the one-way dissection phase. The Nested Dissection phase applied to this subtree will visit all the levels of the tree. For each level, algorithm 5.2 will separate the subtrees with the least number of descendants. In the worst case, when the number of nodes in each subtree is half of the nodes of the previous subtree which contains it, the number of levels is $\mathcal{O}(\log N_Q)$. Thus the worst complexity of algorithm 5.2 is $\mathcal{O}(N_Q \log N_Q)$.

If the tree is unbalanced, i.e. only one of the two children of a Q-node is a branching Q-node, the complexity of algorithm 5.2 is $\mathcal{O}(N_Q)$.

Using the *sep.size* and *sep.list*, we can build the data structure *ND*, which is composed of *ND.list* and *ND.pointer*. *ND.list* contains the entries of *sep.list* in reverse order. *ND.pointer(Q_i)* contains the pointer to the first entry in *ND.list* of the separator of the tree rooted in Q_i. In the following algorithm, we denote by N_{out} the number of out-of-tree arcs. We point out that the number of entries in *sep.list* is equal to N_{out}.

Algorithm 5.3

```
procedure nested.dissection(sep.list, sep.size, ND, QT-ancestor)
        for $\ell$ = 1:$N_{out}$ do
            ND.list($N_{out}$ - $\ell$ +1) = sep.list($\ell$);
            temp = QT-ancestor($N_{out}$ - $\ell$ +1);
            QT-ancestor($N_{out}$ - $\ell$ +1) = QT-ancestor($\ell$);
            QT-ancestor($\ell$) = temp;
        end for;
        size = 0;
        for $Q_i$ = $Q_{root}$ : Q-last($Q_{root}$) do
            size = size +sep.size($Q_i$);
            ND.pointer($Q_i$) = $N_{out}$ - size + 1;
        end for;
end procedure
```

The *ND.list* gives the permutation that will reorder rows and columns of $\tilde{M}_{22}$ in a nested dissection order.

Finally, we can build the pattern of the upper triangular part of $\tilde{M}_{22}$ by using the following algorithm 5.4, which takes advantage of the consecutive order of the Q-nodes forming a tree, obtained by applying a *depth first search* on $\mathcal{T}/\mathfrak{P}$, and of the nested ordering of the out-of-tree arcs in *ND*. We point out that, for each Q_i, the rows and the columns with indices between

ND.pointer(Q_i*)* and *ND.pointer(*Q_i*+1)* form a full diagonal block in $\tilde{M}_{22}$. This pattern of the $\tilde{M}_{22}$ is described by the usual compressed row collection data structure *(irow,jcol)* where for the row i of $\tilde{M}_{22}$ is stored in *jcol(irow(i):irow(i+1)-1)*.

Algorithm 5.4

```
procedure insert.columns(ND, QT, irow, jcol, jcount, count)
          for $Q_k$ = QT+1:Q-last(QT) do;
              for each $\ell = (p, q) \in$ Q-star($Q_k$) with chain(p)= $Q_k$ do
                       if $q >$ last(Q-last($Q_i$)) or $q <$ first($Q_i$) then
                         jcount = jcount + 1;
                         jcol(jcount) = $\ell$;
                         count = count + 1;
                       end if;
              end for each;
          end for;
end procedure;
procedure $\tilde{M}_{22}$-pattern(ND, QT-ancestor, irow, jcol)
          irow(1) = 1; jcount = 0;
          for i = 1:$N_{out}$ do
              count = 0;
              $Q_1$ head of Q-children(QT-ancestor(i)) list;
              $Q_2$ tail of Q-children(QT-ancestor(i)) list;
              for k = i:ND.pointer($Q_1$+1) - 1 do
                  jcount = jcount + 1;
                  jcol(jcount) = ND.list(k);
                  count = count + 1;
              end for;
              let $(p_i, q_i)$ the ends of arc i;
              $Qp_i$ = chain($p_i$); $Qq_i$ = chain($q_i$);
              if QT-ancestor(i) $\neq Qp_i$ and QT-ancestor(i) $\neq Qq_i$ then
                insert.columns(ND, QT-ancestor(i), irow, jcol, jcount, count)
                irow(i+1) = irow(i) + count;
              else
                if $q_i >$ last(Q-last($Q_1$)) or $q_i <$ first($Q_1$) then
                  insert.columns(ND, $Q_1$, irow, jcol, jcount, count);
                  irow(i+1) = irow(i) + count;
                else
                  insert.columns(ND, $Q_2$, irow, jcol, jcount, count);
                  irow(i+1) = irow(i) + count;
                end if;
              end if;
          end for;
end procedure
```

Finally, we observe that the final value of *jcount* gives the total number of nonzero entries of the upper triangular part of $\tilde{M}_{22}$. Therefore, without the explicit computation of the real values of the nonzero entries in $\tilde{M}_{22}$ we can predict the total amount of memory we need for storing the matrix.

5.2 Preconditioners

The *a priori* knowledge of the structure of $\tilde{M}_{22}$, allows us to decide what kind of preconditioner we can afford. If we are subjected to a strong limitation of the size of the memory in our computer, we can choose among several alternative preconditioners.

We have a choice ranging from the diagonal matrix obtained by using the diagonal part of M_{22} and the block Jacobi preconditioner using the diagonal blocks corresponding to the separators.

The possibility of using the simplest choice of the diagonal of M_{22} is sensible because the SPT algorithm places on this diagonal the biggest entries of the diagonal of M.

In Section 6, we will give numerical evidence that this choice is very efficient for several test problems.

Nevertheless, in the presence of strong discontinuities and of anisotropies in the permeability function $\mathcal{K}$, we are obliged to use either the diagonal Jacobi preconditioner or a block diagonal Jacobi.

Finally, by algorithm 5.4, we are able to decide about the feasibility of building and factorizing $\tilde{M}_{22}$ by a sparse direct Cholesky algorithm.

6 Numerical experiments

6.1 Test problems

We generated the test problems using four different domains. The first two are square unit boxes and in the second one we have four rectangular regions where the tensor $\mathcal{K}$ assumes different values. In Figure 4, we plot the geometry of Domain 1 and the boundary conditions. In Figure 5, we plot the geometry of Domain 2 and the boundary conditions. The values of the tensor $\mathcal{K}$ are chosen as follow

$$\mathcal{K}(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in \Omega \setminus \{\Omega_1 \cup \Omega_2 \cup \Omega_3 \cup \Omega_4\}, \\ 0.5 & \mathbf{x} \in \Omega_1, \\ 10^{-4} & \mathbf{x} \in \Omega_2, \\ 10^{-6} & \mathbf{x} \in \Omega_3, \\ 10^{-8} & \mathbf{x} \in \Omega_4. \end{cases}$$

The two remaining domains have an L-shape geometry. In Figure 6, we plot the geometry and the boundary conditions of the Domain 3. In Figure 7, we plot the geometry of the fourth and last Domain 4 and the relative boundary conditions: within the domain, we have four rectangular regions where the tensor $\mathcal{K}$ takes the same values defined for the second domain in (28).

In (2), we take the right-hand side $f(\mathbf{x}) = 0$ in all our test problems. For the domains one and three, the tensor $\mathcal{K}$ in (1) is isotropic. For a given triangulation, its values are constant within each triangle and this value is computed following the law:

$$\mathcal{K}_{T_i} = 10^{-12r_i^3}\ ,\ i = 1, \ldots, N_T$$

where $r_i \in [0, 1]$ are numbers computed using a random uniform distribution.

For each domain, we generated 4 meshes using TRIANGLE (Shewchuck 1996). In Tables 3 and 4, we report, for our domains, the number N_T of triangles, the number N_E of edges, the number N_V of vertices of each mesh, the number $M.nnz$ of nonzero entries in the matrix M, the number $A.nnz$ of nonzero entries in matrix A, and the corresponding value of $\mathfrak{h}$.

	Mesh 1	Mesh 2	Mesh 3	Mesh 4
N_T	153	1567	15304	155746
N_E	246	2406	23130	234128
N_V	94	840	7827	78383
$M.nnz$	1164	11808	114954	1168604
$A.nnz$	429	4599	45601	466319
$\mathfrak{h}$	0.2090	0.0649	0.0225	0.0069

Table 3: Data relative to the meshes for domains 1 and 2.

	Mesh 1	Mesh 2	Mesh 3	Mesh 4
N_T	156	1494	15206	150033
N_E	251	2305	23102	225599
N_V	96	812	7843	75567
$M.nnz$	1187	12269	114662	1125797
$A.nnz$	442	4386	45462	449281
$\mathfrak{h}$	0.1625	0.0590	0.0186	0.0063

Table 4: Data relative to the meshes for domains 3 and 4.

6.2 Practicalities

We analysed the reliability of the stopping criterion when we change the parameter d. In Figures 8 and 9, we display the behaviour of the estimates of the true relative energy norm of the error for Mesh 3, Domain 3 and Domain 4: for the other cases the behaviour is similar. The results show that the choice $d = 10$ is the best compromise between reliability and cost. When convergence is slow and there are regions where the slope changes rapidly, the choice $d = 5$ can be inaccurate. We reserve for future work the study of a self-adaptive technique which will change the value of d with the slope of the convergence curve. In Section 3, we discussed the opportunity of starting the estimate of the relative error only when $e_{\tilde{M}_{22}} < \eta^2 \tilde{h}^T \tilde{h}$. This allows a reduction of the number of additional matrix-vector products. Both figures show an initial phase where the estimates have not been computed because of the introduction of this check on the absolute value of the error.

Moreover, to avoid an excessive number of additional matrix-vector products in the stopping criterion, we choose to update the value of the denominator $\overline{u}_2^{(k)T} \tilde{M}_{22} \overline{u}_2^{(k)}$ every 10 steps of the conjugate gradient method. The energy norm of $\overline{u}_2^{(k)}$ converges quite quickly to the energy norm of the solution and this justifies our choice. In Figures 10 and 11, we see that, after 25% of the iterations, the ratio $\frac{||u||_{\tilde{M}_{22}}}{||u^{(k)}||_{\tilde{M}_{22}}}$ is greater than 0.9.

6.3 Numerical results

We generated and ran all our test problems on a SPARC processor of a SUN ENTERPRISE 4500 (4CPU 400 MHertz, 2GByte RAM). In our test runs, we compare the performance of our approach with the performance of **MA47** of the **HSL2000** library (HSL 2000). The package **MA47** implements a version of the LDL^T decomposition for symmetric indefinite matrices that takes advantage of the structure of the augmented system (Duff, Gould, Reid, Scott and Turner 1991). The package is divided into three parts corresponding to the symbolic analysis where the reordering of the matrix is computed, the factorization phase, and the final solution using the triangular matrices.

Similarly, the null space algorithm which we implemented, can be subdivided into three phases: a first symbolic phase where the shortest path tree and the quotient tree are computed, a second phase where the projected Hessian system is solved by the conjugate gradient algorithm, and a final third phase where we compute the pressure. This enables us to compare the direct solver **MA47** with the null space approach in each single phase.

Generally, in the test runs that we will present, we fix the parameter d in the stopping criterion to the value of 10. Nevertheless, we will show the influence of different choices on the parameter d on the stopping criterion using Mesh 3.

In Table 5, we give the CPU times (in seconds) and the storage (in MByte) required by **MA47** and the CPU times (in seconds) of the null space algorithm where we use the diagonal of M_{22} to precondition the projected Hessian matrix within the conjugate gradient algorithm.

From Table 5, we see that the null space algorithm performs better in the case of random permeability which can be a realistic simulation of an underground situation. Nevertheless, the global CPU time of the null space algorithm can be 10 times more than the CPU time of the direct solver. We point out that the **MA47** storage requirement for the L and D factors grows with the size of the problem whereas the null space algorithm needs only the storage of the matrices M and A. We forecast that this will become even more favourable to the null space algorithm when we want to solve 3D simulations: for these problems the **MA47** storage could become so large that we could be obliged to use an out-of-core implementation.

In Table 6, we display the behaviour of three different preconditioners on the conjugate gradient iteration number. We fixed the mesh (Mesh 3) and we use as a preconditioner one of the following matrices: $diag(M_{22})$, $diag(\tilde{M}_{22})$ (the classical Jacobi), and $blockdiag(\tilde{M}_{22})$ (block Jacobi) which has been computed using the quotient tree of the shortest path tree (see Section 5). For each preconditioner and each domain, we display the number of conjugate

Mesh	Domain	**MA47**				null space algorithm		
		Symbolic	Factorization	Solver	Storage	Symbolic	CG(#Iterations)	Solve
1	1	0.008	0.013	0.001	0.048	0.002	0.013 (12)	0.002
1	2	0.008	0.008	0.001	0.035	0.002	0.013 (14)	0.002
1	3	0.008	0.010	0.001	0.032	0.001	0.013 (13)	0.001
1	4	0.007	0.007	0.001	0.040	0.002	0.016 (17)	0.001
2	1	0.088	0.173	0.006	0.634	0.017	0.145 (19)	0.018
2	2	0.088	0.101	0.005	0.458	0.007	0.275 (35)	0.019
2	3	0.079	0.152	0.006	0.556	0.011	0.145 (20)	0.018
2	4	0.084	0.095	0.005	0.418	0.009	0.265 (37)	0.017
3	1	1.058	3.028	0.114	9.15	0.911	4.681 (41)	0.290
3	2	1.070	1.577	0.084	6.06	0.087	11.63 (101)	0.265
3	3	1.035	2.393	0.111	8.14	0.290	4.945 (44)	0.280
3	4	1.041	1.310	0.081	5.72	0.088	12.06 (106)	0.268
4	1	14.63	264.6	1.543	132.32	7.8	210.6 (176)	2.941
4	2	14.55	49.03	1.091	81.89	1.179	463.0 (390)	2.855
4	3	13.58	84.33	1.481	118.9	3.887	298.7 (272)	2.748
4	4	13.54	34.49	1.067	77.62	1.056	445.6 (395)	2.733

Table 5: **MA47** vs null space algorithm: CPU times (in seconds) and storage (in MBytes).

gradient iterations, the CPU time (in seconds) for the building of the matrix, and the CPU time (in seconds) spent by the conjugate gradient algorithm to solve the projected Hessian linear system. From the results of Table 6, we conclude that the simplest preconditioner $diag(M_{22})$ is faster even if the conjugate gradient algorithm does more iterations than the conjugate gradient algorithm using the other preconditioners. The Jacobi and the block Jacobi preconditioner building cost is very high and overwhelms the good performance of the conjugate gradient algorithm.

Domain	Preconditioner	CG #Iterations	CPU Time (in seconds)	
			Building	CG solve
1	$diag(M_{22})$	41	0.026	4.681
1	$diag(\tilde{M}_{22})$	28	13.60	3.212
1	$blockdiag(\tilde{M}_{22})$	19	15.06	2.854
2	$diag(M_{22})$	101	0.025	11.63
2	$diag(\tilde{M}_{22})$	79	13.16	8.859
2	$blockdiag(\tilde{M}_{22})$	69	12.13	10.18
3	$diag(M_{22})$	44	0.025	5.049
3	$diag(\tilde{M}_{22})$	26	13.94	2.954
3	$blockdiag(\tilde{M}_{22})$	19	15.63	3.122
4	$diag(M_{22})$	106	0.025	12.06
4	$diag(\tilde{M}_{22})$	92	13.10	10.04
4	$blockdiag(\tilde{M}_{22})$	79	13.23	12.37

Table 6: Comparison between the preconditioners: CPU times and # Iterations of conjugate gradient algorithm.

In Table 7, we report the CPU time (in seconds) spent by our implementation of the Algorithms 5.1, 5.2, and 5.3. These algorithms build the nested dissection structure of $\tilde{M}_{22}$ and the null space matrix Z. In particular, the nested dissection structure can be useful in building a parallel version of the matrix-vector products involving the implicit form of the matrix $\tilde{M}_{22}$. Moreover, it is possible to use the structure for the evaluation of the complexity of the explicit computation of the null space matrix Z. We observe that, comparing Table 7 and Table 5, the computational cost of Algorithms 5.1, 5.2, and 5.3 is comparable with the cost of the **MA47** symbolic phase. Algorithms 5.1, 5.2, and 5.3 are the basis on which it is possible to build a parallel version of the matrix-vector computation (14). In particular, it is possible to use Algorithm 5.3 to generate a suitable data structure for the solution of the triangular systems involving the matrices L and L^T.

7 Conclusions

We have analysed a variant of the Null Space algorithm that takes full advantage of the relation between the structure of the augmented system and the network programming structure of the mixed finite-element approximation of the Darcy's equations. We remark that this method can be applied to any diffusion equation.

Mesh	Domain	CPU Time (in Seconds)
		Algorithms 5.1,5.2,5.3
1	1	0.001
1	2	0.001
1	3	0.002
1	4	0.001
2	1	0.025
2	2	0.009
2	3	0.022
2	4	0.006
3	1	1.225
3	2	0.159
3	3	0.982
3	4	0.178
4	1	29.89
4	2	4.775
4	3	28.53
4	4	5.118

Table 7: CPU Time (in seconds) for the computation of nested dissection structure for null space matrix Z .

We compared the performance of our prototype version of the algorithm with a well established direct solver and we concluded that even if our implementation can be 10 times slower than the direct solver, the absence of fill-in makes our code competitive for large problems.

In particular, we want to highlight that the absence of fill-in is promising when we need to solve Darcy's equations in 3D domains. It is reasonable to expect that the fill-in and complexity of a direct solver, applied to the augmented systems related to the approximation of Darcy's equations in 3D domains, will grow as $\mathcal{O}(m^{2/3})$ and $\mathcal{O}(m^2)$ respectively. Our algorithm will instead have no fill-in and its complexity will only depend on the condition number of the scaled projected Hessian matrix Z^TMZ. We can reasonably assume that this condition number will not change with the dimension of the domain Ω, analogous to the behaviour of the classical finite-element method. Therefore, the stopping criterion will stop the conjugate gradient algorithm after a number of steps which will not depend on the dimension of the domain Ω. We reserve for future work the analysis of the performance of our algorithm for three dimensional domains.

We did not implement a parallel version of our algorithm. Nevertheless, we are confident that a parallel version will speed up the performance. In particular, the matrix-vector product involving Z^TMZ can largely benefit from the parallelization, where the block structure of L_1 can be exploited.

Finally, the method we propose could also be generalized to the solution of nonlinear problems with linear equality constraints, where it would be possible to build a specialized version of the nonlinear conjugate gradient algorithm taking advantage of the total unimodularity of A.

References

Albanese, R. and Rubinacci, G. (1988), 'Integral formulation for 3D eddy-current computation using edge elements', *IEEE Proc. A* **135**, 457–462.

Alotto, P. and Perugia, I. (1999), 'Mixed finite element methods and tree-cotree implicit condensation', *CALCOLO* **36**, 233–248.

Amit, R., Hall, C. A. and Porsching, T. A. (1981), 'An application of network theory to the solution of implicit Navier-Stokes difference equations', *J. Comp. Phys.* **40**, 183–201.

Arioli, M. and Baldini, L. (2001), 'A backward error analysis of a null space algorithm in sparse quadratic programming', *SIAM J. Matrix Anal. and Applics.* **23**, 425–442.

Arioli, M., Maryška, J., Rozložník, M. and Tůma, M. (2001*a*), Dual variable methods for mixed-hybrid finite element approximation of the potential fluid flow problem in porous media, Technical Report RAL-TR-2001-023, RAL.

Arioli, M., Noulard, E. and Russo, A. (2001*b*), 'Stopping criteria for iterative methods: Applications to PDE's', *CALCOLO* **38**, 97–112.

Berry, M. W., Heath, M. T., Kaneko, I., Lawo, M., Plemmons, R. J. and Waed, R. C. (1985), 'An algorithm to compute a sparse basis of the null space', *Numer. Math.* **47**, 483–504.

Biró, O., Preis, K., Vrisk, G., Richter, K. R. and Ticar, I. (1993), 'Computation of 3D magnetostatic fields using a reduced scalar potential', *IEEE Trans. Magnetics* **29**, 1329–1332.

Brezzi, F. and Fortin, M. (1991), *Mixed and Hybrid Finite Element Methods*, Vol. 15, Springer-Verlag, Berlin.

Campbell, S. L. and Meyer Jr, C. D. (1979), *Generalized Inverses of Linear Transformations*, Pitman, San Francisco.

Ciarlet, P. G. (1978), *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam.

Coleman, T. F. and Pothen, A. (1986), 'The null space problem I. Complexity', *SIAM, Algebraic Discrete Math.* **7**(4), 527–537.

Coleman, T. F. and Pothen, A. (1987), 'The null space problem II. Algorithms.', *SIAM, Algebraic Discrete Math.* **8**, 544–563.

Deo, N., Prabhu, G. M. and Krishnamoorthy, M. S. (1982), 'Algorithms for generating fundamental cycles in a graph', *ACM, Trans. Math. Softw.,* **8**, 27–42.

Duff, I. S., Erisman, A. M. and Reid, J. (1989), *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, UK.

Duff, I. S., Gould, N. I. M., Reid, J. K., Scott, J. A. and Turner, K. (1991), 'The factorization of sparse symmetric indefinite equations', *IMA, J. Numer. Anal.* **11**, 181–204.

Gallo, G. and Pallottino, S. (1986), ‘Shortest path methods: a unifying approach’, *Mathematical Programming Study* **26**, 38–64.

Gilbert, J. R. and Heath, M. T. (1987), ‘Computing a sparse basis for the null space’, *SIAM, Algebraic Discrete Math.* **8**, 446–459.

Gill, P. H., Murray, W. and Wright, M. H. (1981), *Practical Optimization*, Academic Press, London, UK.

Golub, G. H. and Meurant, G. (1997), ‘Matrices, moments and quadrature II; how to compute the norm of the error in iterative methods’, *BIT* **37**, 687–705.

Greenbaum, A. (1997), *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, PA.

Hall, C. A. (1985), ‘Numerical solution of Navier-Stokes problems by the dual variable method’, *SIAM, J. Alg. Disc. Meth.* **6**, 220–236.

Higham, N. J. (1996), *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA.

HSL (2000), ‘Harwell Software Library. A collection of Fortran codes for large scale scientific computation’, http://www.cse.clrc.ac.uk/Activity/HSL.

Itai, A. and Rodeh, M. (1978), ‘Finding a minimum circuit in a graph’, *SIAM, J. Comput* **7**, 413–423i.

Kettunen, L., Forsman, K. and Bossavit, A. (1999), ‘Gauging in Whitney spaces’, *IEEE Trans. Magnetics* **35**, 1466–1469.

Lancaster, P. and Tismenetsky, M. (1985), *The Theory of Matrices, 2nd ed.*, Academic Press, New York.

Murthy, K. G. (1992), *Network Programming*, Prentice Hall, Englewood Cliffs, NJ.

Pothen, A. (1989), ‘Sparse null basis computations in structural optimization’, *Numer. Math.* **55**, 501–519.

Shewchuck, J. R. (1996), ‘A two-dimensional quality mesh generator and Delaunay triangulator’, http://www.cs.cmu.edu/afs/cs.cmu.edu/project/quake/public/www/triangle.html.

Tarjan, R. E. (1983), *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA.

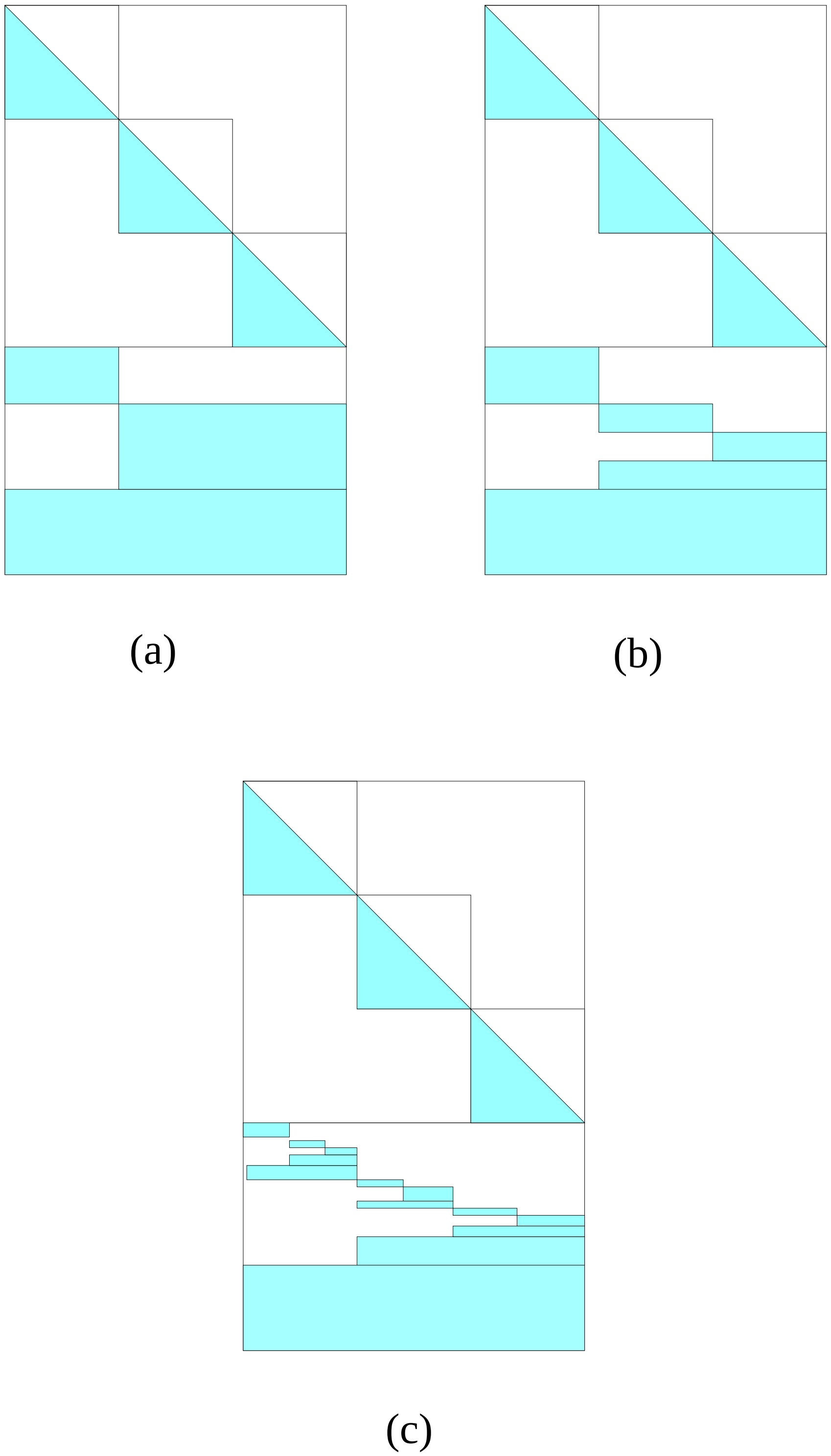

Figure 3: Example of a one-way dissection (a, b), and of a nested dissection (c) on the matrix A

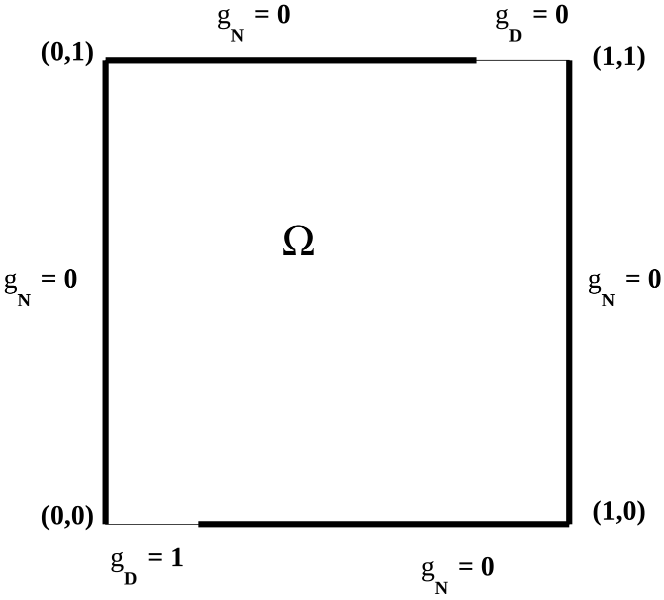

Figure 4: Geometry of the first domain Ω.

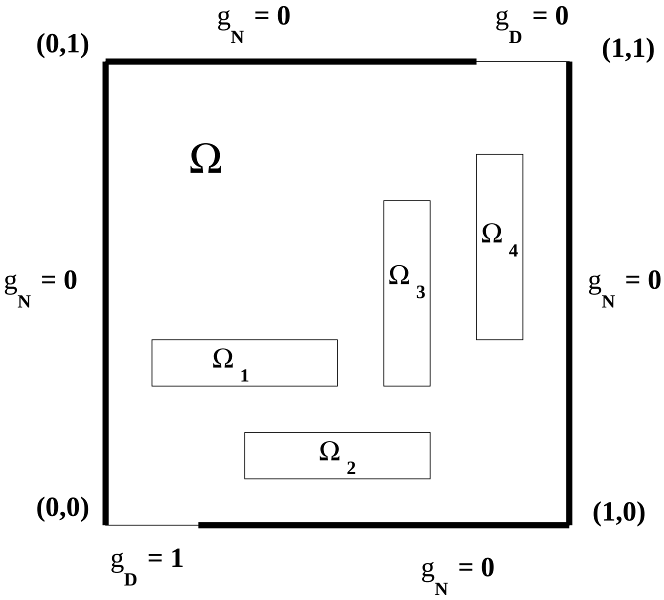

Figure 5: Geometry of the second domain Ω.

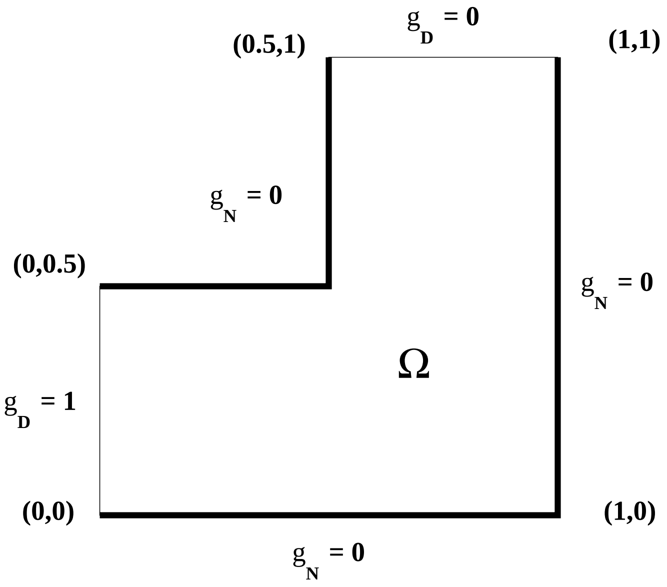

Figure 6: Geometry of the third domain Ω.

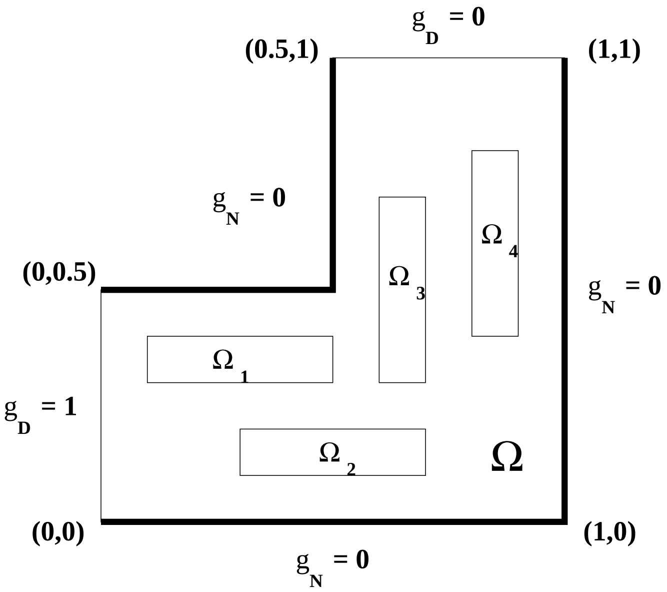

Figure 7: Geometry of the fourth domain Ω.

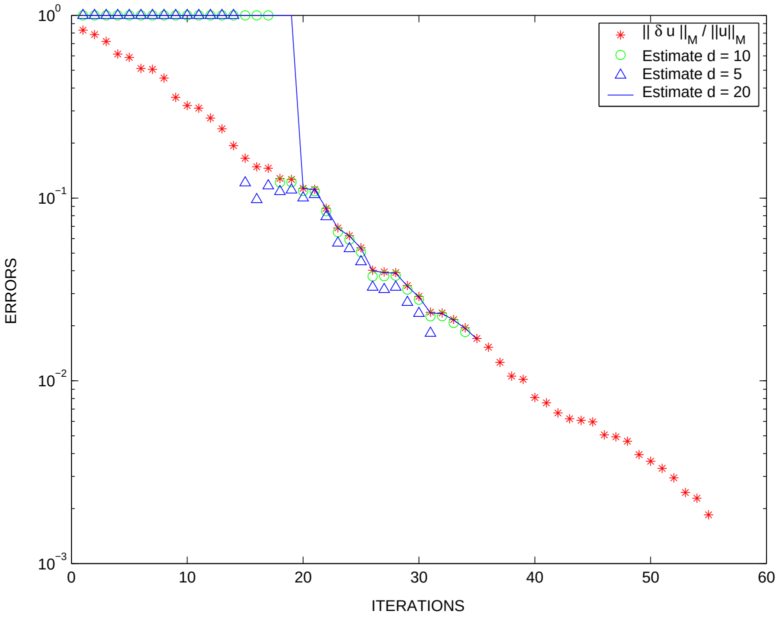

Figure 8: Error energy norm and its estimates for $d = 5$, $d = 10$, and $d = 20$ for Mesh 3 and domain 3.

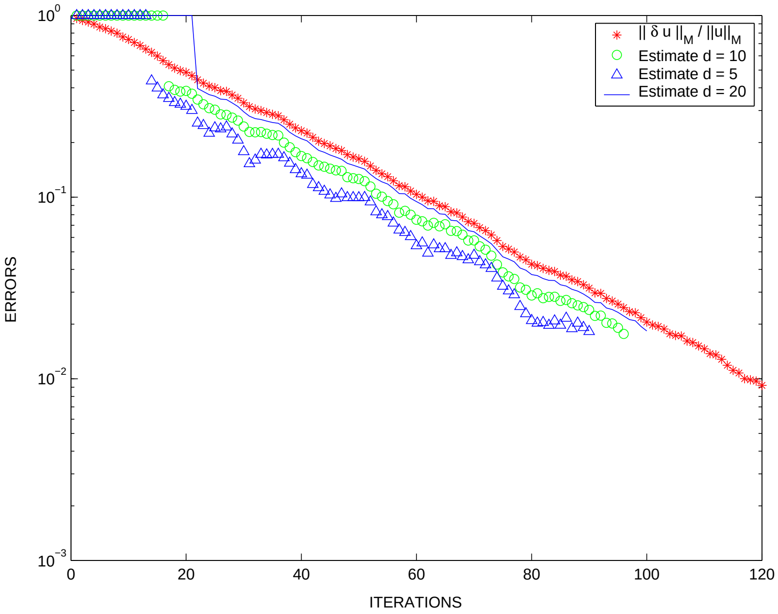

Figure 9: Error energy norm and its estimates for $d = 5$, $d = 10$, and $d = 20$ for Mesh 3 and domain 4.

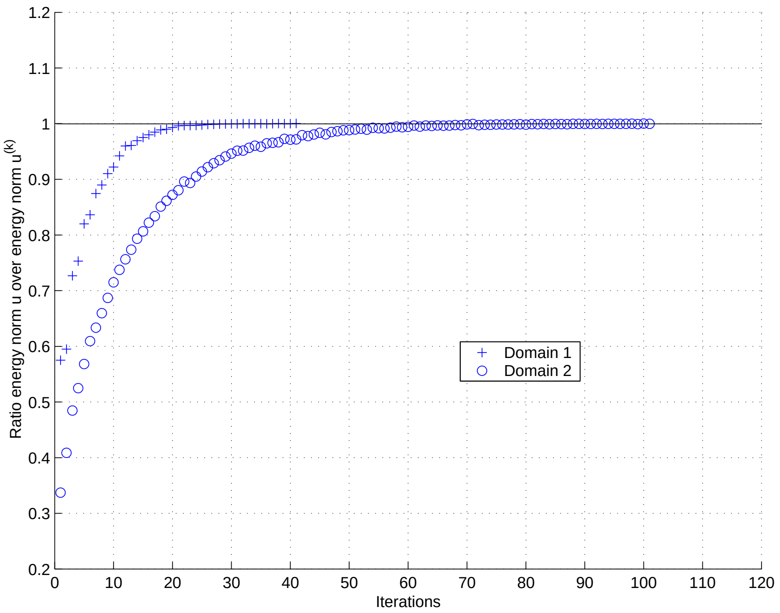

Figure 10: Convergence of $\frac{||u||_{\tilde{M}_{22}}}{||u^{(k)}||_{\tilde{M}_{22}}}$ for Mesh 3 and domains 1 and 2.

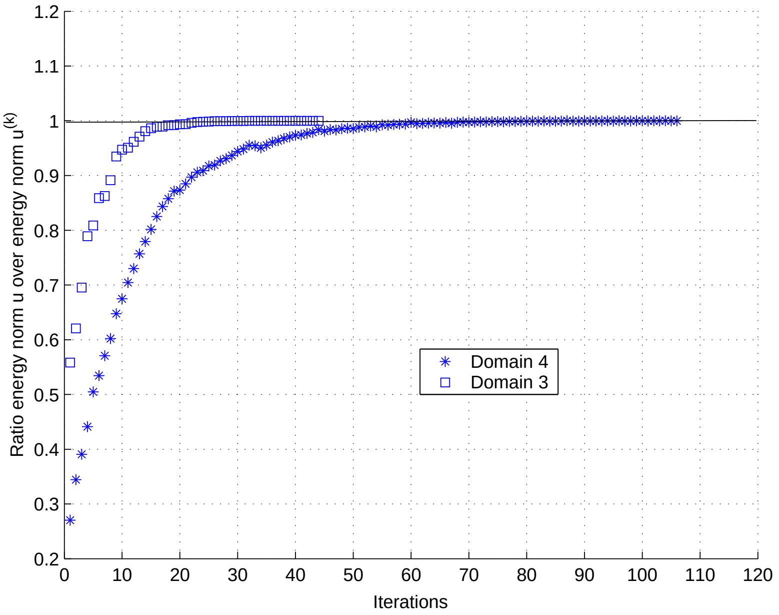

Figure 11: Convergence of $\frac{||u||_{\tilde{M}_{22}}}{||u^{(k)}||_{\tilde{M}_{22}}}$ for Mesh 3 and domains 3 and 4.