# CEMRACS 2001 : Projet ANDRA
# Simulation du stockage souterrain par des méthodes de décomposition de domaine non conformes

Vincent Martin[*]
vincent.martin@inria.fr

Sébastien Wagner[†‡]
sebastien.wagner@jrc.it

January 10, 2002

[*] Andra, Parc de la croix blanche, 1-7 rue Jean Monnet, 92298 Chatenay-Malabry, France

[†] Laboratoire Modélisation Numérique et Couplages, Avenue George Pompidou, BP 56, 83162 La Valette du Var Cédex, France

[‡] Joint Research Centre - Environment Institute, 1 Via Enrico Fermi, 21020 Ispra (Va), Italy

# Contents

# List of Figures

# List of Tables

**Abstract**

Underground storage is considered as one of the feasible solutions to store the nuclear wastes. ANDRA is one of the French agencies in charge of studying the feasibility of such repositories. And to do so, numerical models dealing with multiscale processes need to be developped. This report presents a non overlapping domain decomposition algorithm with non matching and embedded grids. A finite volume method is used with special Robin interface conditions to solve a non-stationary diffusion-convection problem. The numerical implementation is described and some results are presented on the validation of the method.

# Introduction

Une des missions fondamentales de l'Andra (Agence Nationale pour la gestion des Déchets Radioactifs) est de trouver une solution à long terme au problème des déchets radioactifs produits en France. Trois pistes sont actuellement en voie d'exploration :

- la transmutation des déchets en isotopes moins nocifs ou à durée de vie plus courte.
- leur entreposage réversible et surveillé sous terre, mais à faible profondeur.
- leur stockage définitif en couche géologique profonde.

C'est dans le cadre de cette dernière hypothèse que s'inscrit notre projet au CEMRACS 2001. L'Andra cherche en effet à montrer l'éventuelle faisabilité du stockage profond en garantissant des normes de sécurité préalablement définies. Pour cela, la réalisation de simulations numériques fiables est indispensable, puisque c'est le seul moyen de prédire le comportement des déchets en sous-sol sur des durées de plusieurs milliers d'années.

Il est notamment nécessaire d'effectuer des simulations du transport de radionucléides en milieu souterrain afin de prévoir si des radioéléments dangereux sont susceptibles de ressortir dans la biosphère, et, si cela arrive, dans quelles proportions.

De nombreuses difficultés d'ordre purement numérique apparaissent dans l'étude du transport de contaminants radioactifs : celles-ci sont liées par exemple au très fort contraste de perméabilités entre les différentes couches géologiques, au couplage avec la chimie, au couplage de filiation entre les isotopes, aux échelles de temps, aux échelles d'espace...

C'est ce dernier aspect qui nous occupe principalement ici : les colis de déchets ont des tailles caractéristiques qui vont du décimètre au mètre. Le site de stockage a une hauteur de l'ordre du mètre, mais s'étend sur quelques kilomètres de long (voir figure 2). Et finalement, l'étude du transport doit être effectuée sur une partie importante du bassin géologique (de l'ordre de la dizaine de kilomètres, cf. figure 1) pour visualiser le lieu des sorties de contaminants.

Cette diversité dans les échelles spatiales rend de tels problèmes difficiles à traiter numériquement. En effet, ce type de géométrie nécessite :

- un maillage très raffiné à proximité du stockage, où se passent beaucoup de choses : de forts gradients sont à l'œuvre, différentes physiques (chimie, mécanique, thermique, etc.) interagissent... On qualifie cette échelle de *champ proche*.

- et un maillage beaucoup plus lâche à grande distance, afin de ne pas avoir de calculs trop coûteux. On qualifie cette échelle de *champ lointain*.

Le but de ce projet est de montrer la faisabilité d'un couplage entre le champ proche et le champ lointain grâce à une technique particulière de décomposition de domaines non conforme. Cette technique a été introduite par Achdou, Japhet, Maday et Nataf dans (1) et a déjà prouvé numériquement ses performances sur des cas-test académiques. Dans ces conditions, l'essentiel du projet a consisté dans la réalisation d'un outil informatique, écrit en C++, suffisamment souple pour pouvoir résoudre le problème proposé par l'Andra.

Dans la première partie nous donnerons quelques idées de la physique du transport en milieu poreux et nous définirons plus précisément le cas-test qui servira de validation. Ensuite, dans la partie suivante, nous expliquerons brièvement la décomposition de domaines non conforme utilisée, basée sur les volumes finis centrés sur les mailles, avec un raccord par des conditions de type Robin. Dans la troisième partie, nous décrirons les principales caractéristiques de notre code, ses capacités, ses limitations et les évolutions possibles. Dans la quatrième partie, nous validerons notre code sur des problèmes simples et nous donnerons les résultats obtenus par notre code sur le problème de l'Andra.

# 1 Définition du cas-test

On essaie ici de donner une idée de la physique du problème et d'expliciter les simplifications et hypothèses faites pour définir notre cas-test.

Le problème que nous cherchons à résoudre s'inspire d'une série d'exercices proposés au monde de la recherche : les exercices **couplex** (cf. page web *www.andra.fr/couplex/*). Le projet Andra du Cemracs s'inscrit en parallèle à ces exercices

Nous allons donc reprendre ici une partie du cas couplex1 qui donne une idée des difficultés du modèle.

## 1.1 Exemple de géométrie : Couplex1

Le cas-test couplex1 est un exemple de calcul en champ lointain.

Le domaine de calcul est une section 2D du site de stockage. Ainsi le domaine est un rectangle applati de dimensions $25000 \times 695$ (en mètres).

Quatre couches différentes apparaissent :

- le dogger,
- l'argile (*clay* en anglais), contenant le site,
- le calcaire (*limestone*),
- et la marne (*marl*).

## 1.2 Physique du problème

La résolution d'un problème de transport souterrain se fait généralement en deux étapes successives :

1. calcul de l'écoulement de l'eau dans le sous-sol. On suppose souvent que cet écoulement est stationnaire.
2. calcul instationnaire du transport des contaminants dissous dans l'eau, une fois connu le champ des vitesses de l'eau.

Nous allons présenter maintenant la formulation des lois physiques applicables en milieu poreux.

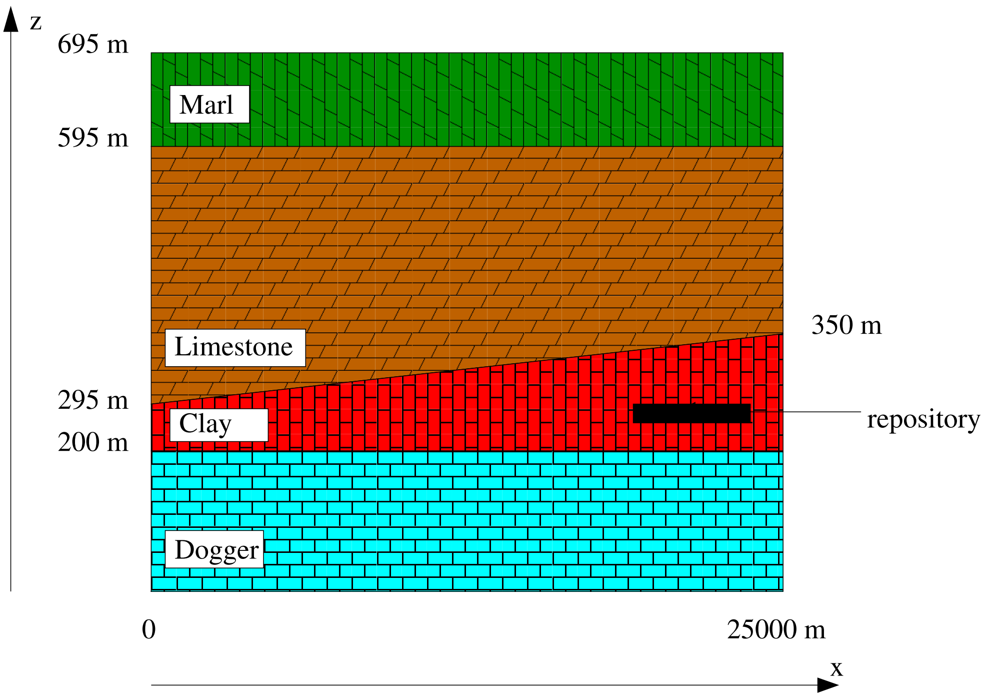


Figure 1: *Géométrie de couplex1. (tiré de* www.andra.fr/couplex/*)*

## 1.3 Ecoulement : la loi de Darcy

On suppose que toutes les couches géologiques sont saturées en eau [1] et que l'écoulement est stationnaire. La loi de Darcy donne l'expression de la vitesse $\mathbf{u}$ en fonction de la charge hydrodynamique $H = P/\rho g + z$ :

$$\mathbf{u} = -K\nabla H \tag{1}$$

où $P$ est la pression, $\rho$ la masse volumique, $z$ l'altitude et $g$ est l'accélération de pesanteur. Le tenseur de perméabilité $K$ est généralement supposé constant par couche. La conservation de la masse s'écrit en régime stationnaire :

$$\nabla \cdot (K\nabla H) = 0 \tag{2}$$

On présente des exemples de valeur de la perméabilité dans les différentes roches de couplex1 dans la table 1. Il faut noter la très forte hétérogéité des milieux : ainsi l'écoulement est-il très lent dans l'argile –milieu très imperméable– et beaucoup plus rapide dans le calcaire.

[1]Cette simplification permet de traiter un problème linéaire.

| | Marne | Calcaire | Argile | Dogger |
|---|---|---|---|---|
| $K$ (m/year) | 3.1536e-5 | 6.3072 | 3.1536e-6 | 25.2288 |

Table 1: *Tenseur de permeabilité dans les quatre couches géologiques.*

## 1.4 Transport

Nous considérons une espèce $i$ qui sort du site de stockage et se dissout dans l'eau. Sa concentration $C_i$ suit l'équation de convection-diffusion dans les roches :

$$R_i\omega(\frac{\partial C_i}{\partial t} + \lambda_i C_i) - \nabla \cdot (\mathbf{D}_i \nabla C_i) + \mathbf{u} \cdot \nabla C_i = f_i \quad \text{dans le milieu poreux.} \tag{3}$$

avec les notations :

- $R_i$ est le facteur de retard;
- $\omega$ : la porosité effective;
- $\lambda_i = \log 2/T_i$ avec $T_i$ : le temps de demi-vie de l'élément;
- $\mathbf{D}_i$ : le tenseur de diffusion/dispersion effective symétrique défini positif qui dépend de la vitesse de Darcy selon l'expression :

  $$\mathbf{D}_i = d_{ei}I + |\mathbf{u}|\left[\alpha_{li}E(\mathbf{u}) + \alpha_{ti}(I - E(\mathbf{u}))\right]$$

  avec

  $$E_{kj}(\mathbf{u}) = \frac{u_k u_j}{|\mathbf{u}|^2}.$$

On donne la valeur des différents coefficients pour des éléments significatifs, dans le tableau 2 ci-dessous.

| | $^{129}$I | | | $^{242}$Pu | | |
|---|---|---|---|---|---|---|
| | $de_1$(m$^2$/year) | $\alpha_L$(m) | $\alpha_T$(m) | $de_2$(m$^2$/year) | $\alpha_L$(m) | $\alpha_T$(m) |
| Dogger | 5.0e-4 | 50 | 1 | 5.0e-4 | 50 | 1 |
| Argile | 9.48e-7 | 0 | 0 | 4.42e-4 | 0 | 0 |
| Calcaire | 5.0e-4 | 50 | 1 | 5.0e-4 | 50 | 1 |
| Marne | 5.0e-4 | 0 | 0 | 5.0e-4 | 0 | 0 |

Table 2: *Coefficients de diffusion/dispersion : éléments radioactifs pour couplex1.*

## 1.5 Description du cas-test

### 1.5.1 Géométrie du cas-test

Nous nous sommes restreints –dans le cadre du projet Andra au Cemracs– à l'étude du couplage entre le voisinage immédiat du stockage et la couche d'argile qui le contient. [2]

Le domaine d'étude est un zoom de couplex1 au niveau du stockage (cf. figure 2).

Le domaine s'étend sur toute la hauteur de la couche d'argile (supposée rectangulaire). Le stockage est défini de façon plus précise que dans couplex1, mais reste simplifié à l'extrême par rapport à une géométrie réaliste : il est constitué de 50 alvéoles carrées de béton, entourées d'argile gonflante et disposées dans l'argile à intervalles réguliers (période spatiale : 20 mètres).

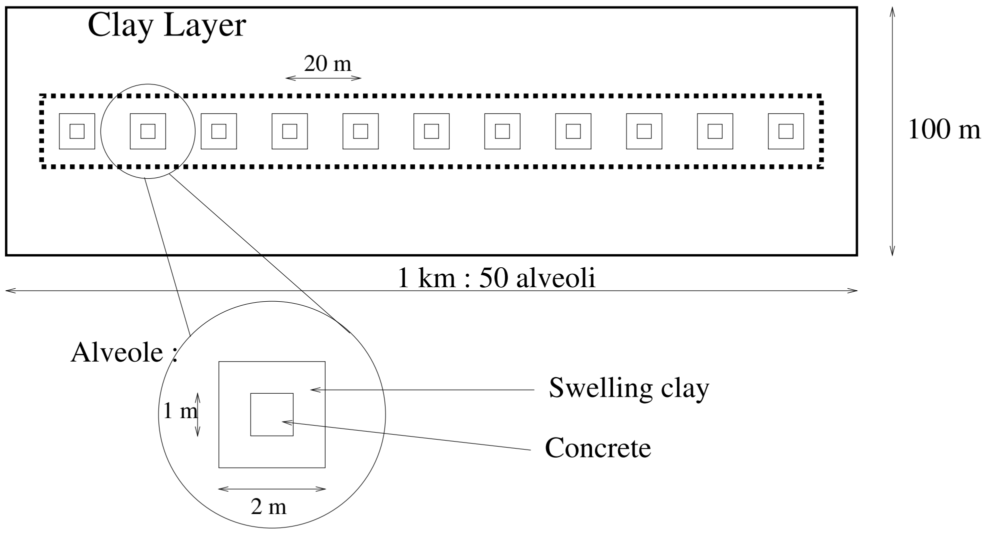


Figure 2: *Géométrie du cas-test Andra : proximité du stockage.*

On décompose le domaine ainsi défini en deux sous-domaines sans recouvrement (c'est-à-dire d'intersection vide). Le premier comprend le voisinage immédiat du stockage (délimité par des pointillés). Le second est l'ensemble de la couche géologique privé du premier sous-domaine : c'est un ouvert contenant un trou.

Le premier devra avoir un maillage très fin, alors que le second sera maillé de façon plus grossière.

On cherche donc à coupler efficacement ces deux sous-domaines qui auront des maillages non raccordés.

### 1.5.2 Equations et hypothèses

A terme, le but est de résoudre l'équation de convection-diffusion (3) dans la couche d'argile. Cependant, comme l'argile est un matériau très peu perméable et que les écoulements en son

[2] Le but n'est ici que de valider la méthode.

sein y sont extrêmement lents, une hypothèse simplificatrice peut être faite :

**Hypothèse 1** *Dans l'argile, on néglige la vitesse de Darcy dans l'équation de transport : la convection est ignorée.*

Avec cette hypothèse, on obtient l'équation de diffusion, avec un terme d'ordre 0, suivante :

$$R_i\omega(\frac{\partial C_i}{\partial t} + \lambda_i C_i) - \nabla \cdot (\mathbf{D}_i \nabla C_i) = f_i \quad \text{dans le milieu poreux.} \tag{4}$$

avec les mêmes notations que pour l'équation (3).

Cependant, l'hypothèse permet de simplifier également $\mathbf{D}_i$ : le tenseur de diffusion / dispersion effective s'écrit maintenant :

$$\mathbf{D}_i = d_{ei} I$$

Pour les calculs de validation, on fait une deuxième hypothèes simplificatrice :

**Hypothèse 2** *On suppose atteindre un régime stationnaire. Le transport est indépendant du temps.*

Finalement, avec les hypothèses (1) et (2), on obtient l'équation de type elliptique suivante :

$$R_i\omega\lambda_i C_i - \nabla \cdot (\mathbf{D}_i \nabla C_i) = f_i \quad \text{dans le milieu poreux.} \tag{5}$$

avec les mêmes notations que pour l'équation (3).

### 1.5.3 Remarque sur le découpage en sous-domaines

Il était envisageable a priori de faire une décomposition de domaine du cas-test de deux manières distinctes :

1. soit on décompose de la façon décrite sur la figure 2. La décomposition est qualifiée de *géométriquement non-conforme*. Les deux sous-domaines sont *emboîtés* l'un dans l'autre.

2. soit on fait un découpage *géométriquement conforme* en sous-domaines *adjacents*, comme sur la figure 3.

Ces deux possibilités ne sont pas équivalentes numériquement. En effet, la première nécessite un maillage *non structuré* pour pouvoir pouvoir prendre en compte un trou, alors que la seconde peut garder un maillage complètement structuré.

Par contre, la deuxième option présente un inconvénient majeur : on crée artificiellement des sous-domaines qui ont des formes très allongées et sans signification physique. De plus, on ne peut pas simplement répéter l'opération de décomposition : il n'est pas trivial dans ce cas de redécomposer le voisinage du stockage en plusieurs sous-domaines correspondant à chaque alvéole. Une telle décomposition de domaines est beaucoup moins souple.

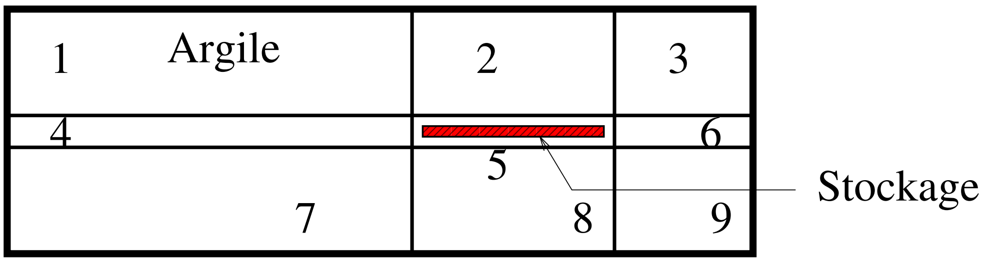


Figure 3: *Couche d'argile décomposée en 9 sous-domaines adjacents autour du domaine.*

C'est pour éviter ce problème qu'on a choisi de se placer dans le premier cas et qu'on a opté pour une géométrie non-conforme et des maillages non structurés.

Une précaution est cependant nécessaire avec de telles décompositions : nous nous sommes toujours limités au cas où le maillage du sous-domaine externe coïncide avec les limites du sous-domaine interne.

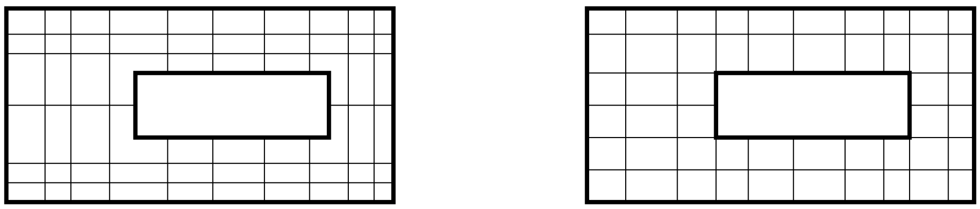

Figure 4: *Maillages non-coïncidant (gauche) et coïncidant (droite) pour une décomposition géométriquement non-conforme.*

# 2 Décomposition de domaine non conforme avec volumes finis

Nous présentons brièvement dans cette section la méthode de discrétisation utilisée (les volumes finis centrés sur les mailles) et la méthode de raccordement entre les sous-domaines (raccord par conditions de Robin).

## 2.1 Volumes finis centrés sur les mailles

On se réfèrera par exemple au manuel (2) pour une présentation des volumes finis.

L'hypothèse suivante a été faite :

**Hypothèse 3** *On se limite dans l'implémentation à un maillage constitué uniquement de rectangles.* [3]

### 2.1.1 Maillage

On se donne donc un maillage *rectangulaire* d'un domaine $\Omega$ borné polygonal de $I\!R^2$. La géométrie du domaine peut être relativement quelconque : en particulier, le domaine n'est pas supposé rectangulaire a priori (voir la figure 5).

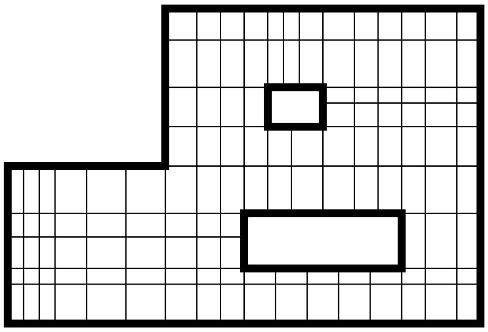

Figure 5: *Exemple de maillage rectangulaire d'un domaine avec 2 trous.*

### 2.1.2 Equation de conservation discrète

Après intégration sur chaque maille $K$ de l'équation de conservation (5), puis application du théorème de Green, on obtient le problème discrétisé suivant :

$$\sum_{\sigma \in \partial K} \Phi^i_{K,\,\sigma} + |K| R_i \omega \lambda_i \, C^i_K = |K| f^i_K \quad \forall K, \tag{6}$$

avec :

[3] L'implémentation des volumes finis avec des maillages quadrilatéraux quelconques ou triangulaires est beaucoup plus complexe et ne s'avère pas nécessaire dans le cadre de ce projet.

- $\sigma$ une arête de la maille $K$,
- $|K|$ la mesure de K,
- $f_K^i$ la moyenne de $f_i$ sur $K$,
- $C_K^i$ la moyenne de $C_i$ sur $K$,
- $\Phi_{K,\,\sigma}^i = -\int_\sigma D^i \nabla C^i \cdot \vec{n}_\sigma$ le flux sortant de $K$ à travers l'arête $\sigma$.

### 2.1.3 Expression du flux

Les $C_K^i$ sont les inconnues principales du problème. Il reste à exprimer les flux $\Phi_{K,\,\sigma}^i$ en fonction des $C_K^i$. Pour chaque maille $K$, on approxime le flux à travers une de ses arêtes $\sigma$ par la formule :

$$\Phi_{K,\,\sigma}^i = -|\sigma|\xi_{K,\,\sigma}(C_\sigma^i - C_K^i) \quad \forall \sigma \in \partial K. \tag{7}$$

On a appellé $\xi$ le rapport de la diffusivité par la distance $d$ :

$$\xi_{K,\,\sigma} = \frac{D_{K,\,\sigma}^i}{d_{K,\,\sigma}} \tag{8}$$

où, en notant $\vec{n}_{K,\,\sigma}$ la normale sortante de $K$ sur l'arête $\sigma$ et $||\cdot||$ la norme euclidienne, on a défini :

$$D_{K,\,\sigma}^i = ||D^i \cdot \vec{n}_{K,\,\sigma}|| \quad \text{et :} \quad d_{K,\,\sigma} = \text{dist}\,(\text{centre}(K), \sigma).$$

**Remarque 1** *On a ajouté ici une inconnue secondaire sur l'arête : $C_\sigma^i$. Ce terme s'interprète en* **éléments finis mixtes hybrides** *comme la* **trace de concentration** *sur l'arête. Par continuité de la concentration, deux mailles voisines ont la même trace de concentration sur leur arête commune. Dans ces conditions, on ne définit qu'une seule trace de concentration $C_\sigma^i$ par arête $\sigma$.*

Il reste à éliminer les inconnues secondaires sur les arêtes $C_\sigma^i$ pour obtenir le système linéaire en $C_K^i$.

### 2.1.4 Flux sur les arêtes internes : continuité des concentrations et des flux

La continuité des flux entre deux mailles $K$ et $L$ voisines s'écrit :

$$\Phi_{K,\,\sigma}^i = -\Phi_{L,\,\sigma}^i \quad \forall \sigma = K \cap L. \tag{9}$$

Avec cette équation (9) et la continuité de la trace de concentration (cf. remarque 1), on peut alors éliminer toutes les inconnues $C_\sigma^i$ sur les arêtes internes. On obtient ainsi l'expression du flux de la maille $K$ vers la maille $L$ à travers l'arête $\sigma$ uniquement en fonction des concentrations dans ces mailles $K$ et $L$ :

$$\Phi^i_{K,\,\sigma} = -|\sigma| \frac{1}{\frac{d_{K,\,\sigma}}{D^i_{K,\,\sigma}} + \frac{d_{L,\,\sigma}}{D^i_{L,\,\sigma}}} (C^i_L - C^i_K) \quad \forall \sigma = K \cap L,\ \sigma \text{ arête interne.} \tag{10}$$

En remarquant que la fraction dans (10) est la moyenne harmonique des $\xi$ définis au (8), on réécrit (10) :

$$\begin{aligned} \Phi^i_{K,\,\sigma} &= -|\sigma| \xi_{K,\,L} (C^i_L - C^i_K) \quad \forall \sigma = K \cap L \quad \text{avec :} \\ \frac{1}{\xi_{K,\,L}} &= \frac{1}{\xi_{K,\,\sigma}} + \frac{1}{\xi_{L,\,\sigma}} \end{aligned} \tag{11}$$

#### 2.1.5 Flux sur les arêtes de bord : prise en compte des conditions aux limites

Il reste maintenant à calculer le flux sur les arêtes de bord en tenant compte des conditions aux limites.

On considérera les conditions aux limites de Robin généralisées qui permettent d'avoir simplement et en une seule fois toutes les conditions aux limites usuelles :

$$\beta\ D^i \nabla C^i \cdot \vec{n} + \alpha\ C^i = \alpha\ \bar{u} \quad \text{sur le bord.} \tag{12}$$

avec $\beta$ et $\alpha$ définissant le type de condition aux limites (cf. la table 3) et $\bar{u}$ donnant la valeur de cette condition aux limites.

| $\alpha$ | $\beta$ | Condition aux limites |
|---|---|---|
| 0 | $> 0$ | Neumann |
| $> 0$ | 0 | Dirichlet |
| $> 0$ | 1 | Robin |

Table 3: *Type de Condition aux Limites en fonction des coefficients $\alpha$ et $\beta$.*

**Remarque 2** *Il faut prendre une petite précaution avec la condition de Neumann exprimée de cette façon : écrite sous cette forme, on impose obligatoirement une condition homogène (car $\alpha = 0$). Pour avoir des conditions non homogènes, il suffit en fait fixer directement le produit ($\alpha\ \bar{u}$) qui seul a un sens et qui est non nul a priori.*

La condition continue (12) se réécrit sous la forme discrète :

$$-\beta\ \Phi^i_{K,\,\tau} + |\tau| \alpha\ C^i_\tau = |\tau| \alpha\ \bar{u} \quad \forall \tau \in \partial K,\ \tau \text{ arête de bord.} \tag{13}$$

On utilise cette équation (13) et l'équation (7) pour éliminer l'inconnue sur l'arête $C^i_\tau$. On obtient finalement :

$$\Phi^i_{K,\,\tau} = -|\tau| \frac{\alpha\ \xi_{K,\,\sigma}}{\alpha + \beta\ \xi_{K,\,\sigma}} (\bar{u} - C^i_K) \quad \forall \tau \in \partial K,\ \tau \text{ arête de bord.} \tag{14}$$

### 2.1.6 Système linéaire

Finalement, on remplace les flux donnés par les formules (11) et (14) dans l'équation de conservation (6). Il ne reste comme inconnues que les concentrations $C_K^i$ constantes par mailles.

Le système linéaire obtenu est connu pour être symétrique défini positif.

**Remarque 3** *Pour obtenir un système symétrique, il est nécessaire d'éliminer toutes les inconnues sur les arêtes, en particulier celles du bord.* Si on ne le fait pas, les conditions aux limites introduisent, si on ne prend pas de précaution, des équations qui détruisent la symétrie de la matrice volumes finis.

## 2.2 Décomposition de domaines avec grilles non raccordées

On se réfèrera à l'article (1) pour une description complète de la méthode de décomposition de domaine non conforme utilisée, ainsi que pour la bibliographie sur les autres méthodes de décomposition de domaine non conforme.

### 2.2.1 Principe

On se donne $\Omega$ un domaine borné de $\mathbb{R}^d$ $(d > 0)$. On cherche à résoudre le problème :

- trouver $p$ tel que :

$$\begin{aligned} \eta\, p - \nabla \cdot (\mathbf{D}\nabla p) &= f \quad \text{dans } \Omega, \\ p &= 0 \quad \text{sur } \partial\Omega, \end{aligned} \tag{15}$$

avec $\eta \geq \eta_0 > 0$ et $D \geq D_0 > 0$ réels.

Le domaine est décomposé en $N$ sous-domaines sans recouvrement : $\bar{\Omega} = \cap_{1\leq k\leq N}\bar{\Omega}_i$. Soit $\alpha > 0$, le problème précédent se reformule comme un problème de décomposition de domaines avec raccord par conditions de type Robin :

- trouver $p_k$ tel que :

$$\eta\, p_k - \nabla \cdot (\mathbf{D}\nabla p_k) = f \quad \text{dans } \Omega_k, \tag{16}$$

$$p_k = 0 \quad \text{sur } \partial\Omega \cap \partial\Omega_k,$$

$$\frac{\partial p_k}{\partial n_k} + \alpha p_k = -\frac{\partial p_l}{\partial n_l} + \alpha p_l \quad \text{sur } \partial\Omega_k \cap \partial\Omega_l, \tag{17}$$

Une méthode itérative simple pour résoudre ce problème de décomposition de domaine est :

- étant donnés $p_l^n$ $\forall l = 1..N$, on calcule $p_k^{n+1}$ dans chaque sous-domaine $\Omega_k$, en résolvant :

$$
\begin{aligned}
\eta\, p_k^{n+1} - \nabla\cdot(\mathbf{D}\nabla p_k^{n+1}) &= f \quad \text{dans } \Omega_k,\\
p_k^{n+1} &= 0 \quad \text{sur } \partial\Omega\cap\partial\Omega_k,\\
\frac{\partial p_k^{n+1}}{\partial n_k} + \alpha p_k^{n+1} &= -\frac{\partial p_l^n}{\partial n_l} + \alpha p_l^n \quad \text{sur } \partial\Omega_k\cap\partial\Omega_l.
\end{aligned}
\tag{18}
$$

Cet algorithme très simple est implémenté dans le code. Il est accéléré par avec BiCGStab afin de converger plus rapidement (voir l'article de Van der Vorst (3) pour une description de BiCGStab). D'autres méthodes de Krylov pour des problèmes non symétriques (CG-S, GMRes, etc.) sont également envisageables.

Nous utilisons la discrétisation par volumes finis, décrite dans la section précédente, avec des maillages dans les sous-domaines non raccordés.

### 2.2.2 Explicitation de l'algorithme pour 2 sous-domaines

On se place dans le cas de 2 sous-domaines pour expliciter l'algorithme ; le principe est le même pour $n \geq 2$ sous-domaines.

Soit $T_1$ et $T_2$ les opérateurs définis sur l'interface $\Gamma = \partial\Omega_1\cap\partial\Omega_2$ par :

$$
T_k(\lambda_k|_\Gamma, f) = \left(-\frac{\partial p_k}{\partial n_k} + \alpha p_k\right)|_\Gamma\ , \tag{19}
$$

où $p_k$ est défini par :

$$
\begin{aligned}
\eta\, p_k - \nabla\cdot(\mathbf{D}\nabla p_k) &= f \quad \text{dans } \Omega_k, && (20)\\
p_k &= 0 \quad \text{sur } \partial\Omega\cap\partial\Omega_k,\\
\frac{\partial p_k}{\partial n_k} + \alpha p_k &= \lambda_k \quad \text{sur } \Gamma = \partial\Omega_1\cap\partial\Omega_2. && (21)
\end{aligned}
$$

On note $\lambda_2^{(n+1)}$ la valeur retournée par l'opérateur $T_1$ prenant comme argument $(\lambda_1^n, f)$. Ce $\lambda_2^{(n+1)}$ sert à l'itéré suivant de valeur d'entrée pour l'operateur $T_2$. De même : $\lambda_1^{(n+1)} = T_2(\lambda_2^n, f)$. De cette façon, l'algorithme (18) s'écrit :

$$
\begin{bmatrix} \lambda_1^{(n+1)} \\ \lambda_2^{(n+1)} \end{bmatrix} = \begin{bmatrix} T_2(\lambda_2^n, f) \\ T_1(\lambda_1^n, f) \end{bmatrix} \tag{22}
$$

Par bilinéarité des opérateurs $T_k$, k = 1 et 2, on peut séparer le terme source du terme d'interface pour obtenir l'équation :

$$
\begin{bmatrix} \lambda_1^{(n+1)} \\ \lambda_2^{(n+1)} \end{bmatrix} = \begin{bmatrix} 0 & T_2(\cdot,0) \\ T_1(\cdot,0) & 0 \end{bmatrix} \begin{bmatrix} \lambda_1^n \\ \lambda_2^n \end{bmatrix} + \begin{bmatrix} T_2(0,f) \\ T_1(0,f) \end{bmatrix}. \tag{23}
$$

Cet algorithme de type point fixe correspond à la résolution du système linéaire en $[\lambda_1\ \lambda_2]$ suivant :

$$
\left[ Id - \begin{pmatrix} 0 & T_2(\cdot,0) \\ T_1(\cdot,0) & 0 \end{pmatrix} \right] \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} T_2(0,f) \\ T_1(0,f) \end{bmatrix}. \tag{24}
$$

C'est ce système qui, une fois discrétisé, peut être résolu par une méthode de Krylov pour matrices non symétriques.

### 2.2.3 Discrétisation de la condition d'interface

La condition de Robin (17) est imposée de façon discrète en utilisant l'équation (13) avec $\beta = 1$ :

$$-\frac{\Phi_{K,\,\tau}}{|\tau|} + \alpha\, C_\tau = \sum_{m \neq k} \sum_{\text{arêtes } i_m} \frac{|\tau_m^{i_m} \cap \tau|}{|\tau|} \left( \frac{\Phi_{K_m,\,\tau_m^{i_m}}}{|\tau_m^{i_m}|} + \alpha\, C_{\tau_m^{i_m}} \right) \quad \forall \tau \in \partial K,\ \tau \text{ arête d'interface.} \tag{25}$$

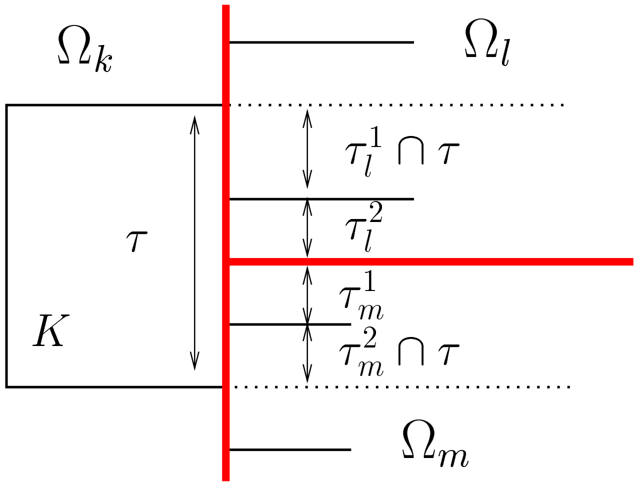


Figure 6: *Conditions d'interface : le maillage.*

Dans cette formule (25), la maille $K$ est située dans le sous-domaine $\Omega_k$ (voir figure 6). Son arête $\tau$ est une arête d'interface, commune aux sous-domaines $\Omega_m,\ m \neq k$. Les $C_\tau$ sont les traces de concentrations et les $\Phi_{K,\,\tau}$ sont les flux sur l'arête $\tau$.

En définitive, la valeur de la condition de Robin s'écrit comme la moyenne pondérée par les longueurs d'arête communes des conditions de Robin (à une modification de signe près) des autres sous-domaines.

### 2.2.4 Conclusion

Dans (1), les auteurs démontrent que l'algorithme (18) avec les conditions d'interface (25) converge pour un $\alpha > 0$ quelconque (pouvant dépendre du maillage).

De plus, l'analyse d'erreur y est faite : il est prouvé –sous certaines hypothèses géométriques[4], et en imposant une condition sur $\alpha$– que l'erreur discrète croît en $O(h^{2-\gamma})$, avec un $\gamma \geq 0$ dépendant du $\alpha$ choisi. Ainsi, pour un $\alpha$ constant, l'erreur $H^1$ discrète est en $O(h)$.

Des illustrations numériques du comportement de l'algorithme justifient les estimations théoriques.

[4]Ces hypothèses sont respectées dans le cas de maillages rectangulaires

# 3 Implementation du modèle

## 3.1 Cahier des charges

L'implémentation informatique du modèle a été réalisée en C++. Plusieurs raisons expliquent le choix d'une programmation orientée objet :

- la souplesse et la facilité de gestion des variables dans le code : les possibilités offertes par le C++ pour concevoir des classes d'objets et les relier entre elles (par héritage par exemple) permettent aux développeurs de construire des structures répondant à des besoins précis. Une fois conçues, ces structures peuvent être dupliquées, manipulées et détruites très facilement,

- la facilité de développement du modèle : les deux principaux obstacles à l'écriture d'un code sont :

  1. la correction des erreurs, la mise à jour des fichiers sources et l'intégration des différentes versions dans le cas de co-développement,
  2. l'extensibilité du code par divers développeurs.

  Pour que ces deux points ne soient pas des facteurs limitants du modèle, il faut s'assurer que le langage de programmation retenu permette, d'une part, une facile maintenance des fichiers sources et, d'autre part, autorise la création de nouveaux modules et ce de façon tout à fait indépendante du reste du code. Le C++ est le langage adapté pour répondre à ces besoins,

- le contrôle que peuvent avoir les développeurs sur les variables informatiques grâce à l'utilisation d'un puissant langage informatique tel que le C++.

Au niveau de l'écriture du modèle, les critères à respecter sont les suivants :

1. implémentation d'une méthode de volumes finis. Cette méthode, basée sur Achdou et al. (1), permet d'utiliser des grilles non raccordées.

2. utilisation de mailles rectangulaires non uniformes,

3. gestion des sous-grilles : grilles adjacentes et grilles emboîtées. Il est notamment important de pouvoir traiter une grille emboitée comme un trou par rapport à la grille dans laquelle elle se trouve. Ceci permet de s'affranchir de découpages inutiles en sous-domaines adjacents. *Ceci requiert un maillage non structuré*, afin de pouvoir traiter des sous-domaines troués (cf. exemple figure 5).

4. souplesse dans la manipulation de ces sous-grilles,

5. conditions aux limites généralisées : utilisation d'une condition de Fourier généralisée pour traiter des conditions de type Dirichlet, Neumann ou Robin.

Dans un premier temps, nous décrirons la structuration des données avant de présenter l'algorithme de résolution du problème.

## 3.2 Organisation des données

La structuration et l'organisation des données suit le découpage suivant :

- la géométrie des sous-domaines,
- les propriétés physiques et chimiques du milieu et des composants radioactifs,
- les conditions aux limites,
- un objet mathématique sous-domaine contenant des données géometriques, physico-chimiques et mathématiques.

Ce découpage est clairement justifié par le type de problème à résoudre. En effet, l'utilisateur du modèle veut pouvoir résoudre un problème de convection-diffusion dans un domaine préalablement défini. Ce domaine posséde une géométrie discrétisée à l'aide d'éléments rectangulaires non uniformes. De plus, les propriétés physico-chimiques de ce domaine peuvent varier d'un point de l'espace à l'autre.

D'autre part, la nature du polluant dont on veut étudier la dispersion varie d'un cas à l'autre. Et donc ses propriétés physico-chimiques varient d'un cas à l'autre.

Enfin, le type de conditions aux limites varie suivant si le bord est une frontière externe (conditions aux limites “classiques”), ou si le bord est en fait une interface entre sous-domaines (conditions aux limites de type Fourier avec un raccord spécifique).

Chacune de ces structures de données remplit une fonction bien précise et peut être elle-même l'objet d'un redécoupage interne plus spécifique.

La figure 7 illustre de façon globale l'organisation des classes les unes par rapport aux autres à l'intérieur du modèle NestedVF. Ces classes sont décrites dans les sections suivantes.

### 3.2.1 La géometrie

La géométrie du modèle comporte plusieurs échelles. Un sous-domaine (qui pourra être également appelé domaine suivant le contexte) est en effet constitué de points, d'arêtes et de cellules.

**Remarque 4** *Une différence est faite entre cellule, qui constitue dans le présent modèle une unité géométrique, et élément qui est une entité géométrique possédant des informations supplémentaires sur le milieu.*

Des classes correspondants aux divers niveaux de géométrie ont donc été implémentées. Pour ne pas rendre l'information redondante, ces classes sont grossièrement reliées de la facon suivante :

- une cellule est un ensemble d'arêtes,
- une arête est représentée par deux points,
- un point est défini par des coordonnées en espace.

Nous détaillons à présent plus en détail chacune de ces classes en commençant par l'entité de base, le point, pour finir par la cellule.

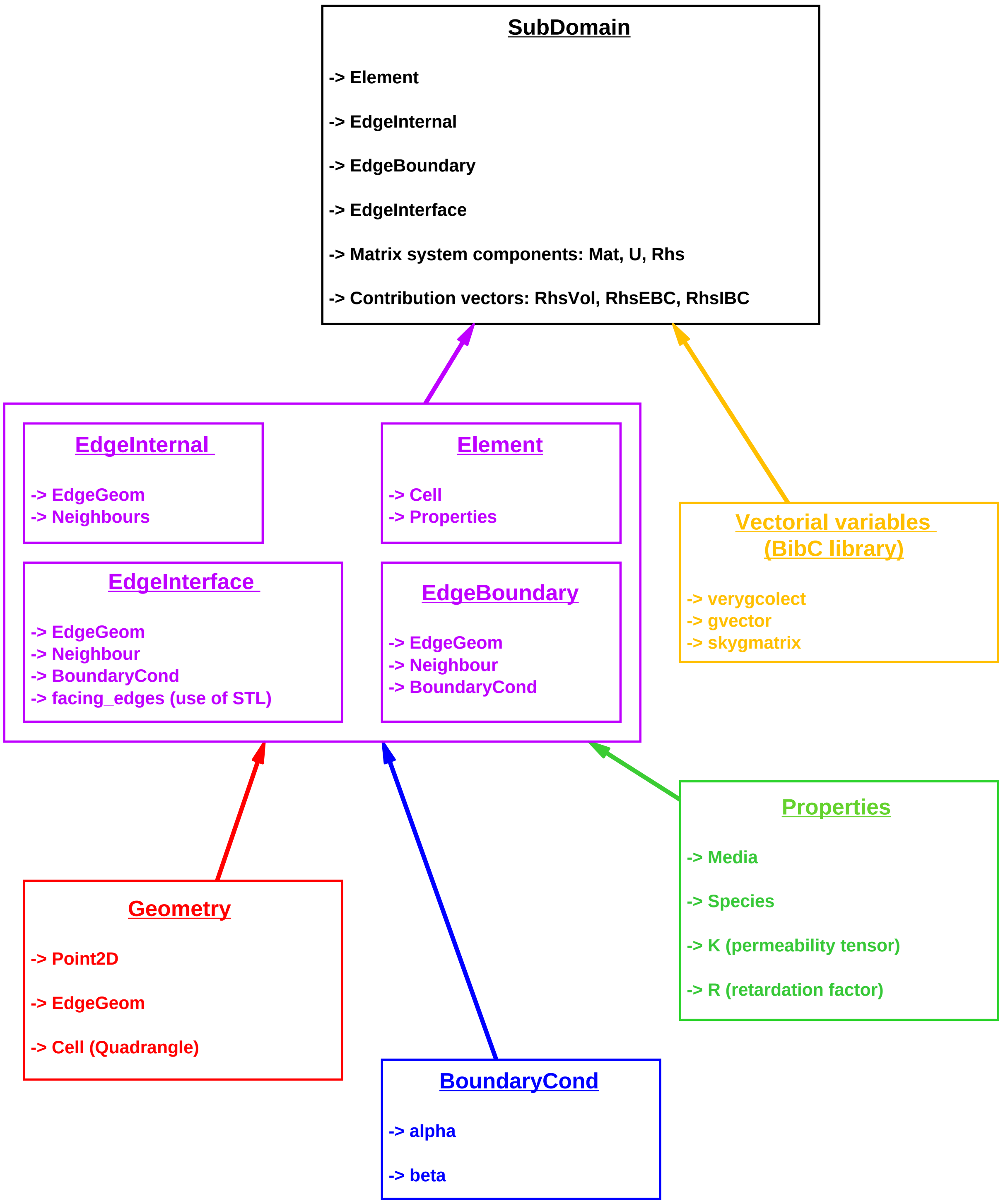


Figure 7: Architecture des classes mises en place dans le modèle NestedVF

#### 3.2.1.1 Classe Point2D

Le modèle étant un modèle à deux dimensions, un point est défini seulement par deux coordonnées en espace. Nous avons également rajouté un champ d'identification du point (numéro d'identité).

Privé :

- *double x,y* : coordonnées du point,
- *int ref* : numéro d'identité.

Les principales caractéristiques définissant un point sont à accès protégé afin d'éviter une manipulation en dehors de la structure Point2D. Pour accéder à ces champs en dehors de cette structure, des méthodes d'accès en écriture et/ou en lecture doivent être définies. Elles seront déclarées *"publiques"*.

Publique :

- *inline double & X()* : accès à la valeur de la première coordonnée avec possibilité de modification,
- *double & Y()* : accès à la valeur de la deuxième coordonnée avec possibilité de modification,
- *int & Ref()* : accès à la valeur du numéro d'identification avec possibilité de modification,
- *const double & X()* : accès à la valeur de la première coordonnée sans possibilité de modification,
- *const double & Y()* : accès à la valeur de la deuxième coordonnée sans possibilité de modification,
- *const int & Ref()* : accès à la valeur du numéro d'identification sans possibilité de modification,
- *Point2D(double xx=0,double yy=0,int rref=-1)* : constructeur,
- *Point2D(const Point2D & p)* : constructeur par copie,
- ˜*Point2D()* : destructeur,
- *Point2D & operator= (const Point2D & p)* : surcharge de l'opérateur = afin de permettre l'affectation d'un point à un autre,
- *friend ostream & operator<<(ostream & f, const Point2D & p)* : surcharge de l'opérateur d'affichage afin de permettre l'impression écran d'un objet *Point2D*.

#### 3.2.1.2 Classe EdgeGeom

Maintenant que la classe *Point2D* est définie, il est possible de construite une arête. Une arête est définie par deux points qui sont ses extrémités et par sa longueur.

Privé :

- *Point2D *beg, *end* : pointeurs vers les points extrémités de l'arête,
- *double length* : longueur de l'arête.

Une variable de type vecteur ou couple de réels pourra être déclarée si besoin afin de stocker un vecteur unitaire normal à l'arête. De même que les extrémités et la longueur de l'arête sont placées en *privé*, la méthode permettant le calcul de la longueur, et celle, si besoin, évaluant le vecteur normal seront également des membres privés de la classe EdgeGeom :

- *void calc_length()* : calcule de la longueur de l'arête.

Tout comme pour Point2D, il faut définir en mode *publique* les méthodes permettant d'avoir accès aux champs privés de la classe. Une méthode effectuant le calcul de la distance d'une arête à un point est également définie. Elle est utilisée notamment dans le calcul des flux internes et le calcul des flux sur les bords.

Publique :

- *Point2D & Begin()* : accès à la valeur de la première extrémité avec possibilité de modification,
- *Point2D & End()* : accès à la valeur de la deuxième extrémité avec possibilité de modification
- *Point2D* & Ptbeg()* : accès au pointeur sur la première extrémité avec possibilité de modification,
- *Point2D* & Ptend()* : accès au pointeur sur la deuxième extrémité avec possibilité de modification
- *double & Length()* : accès à la valeur de la longueur de l'arête avec possibilité de modification,
- *EdgeGeom()* : constructeur par défaut,
- *EdgeGeom(const EdgeGeom & ar)* : constructeur par copie,
- *EdgeGeom(Point2D* pd, Point2D* pf)* : constructeur par affectation
- *EdgeGeom(Point2D& pd, Point2D& pf)* : constructeur autorisant à la donnée de deux points seulement. La longueur (et la normale si définie) est calculée automatiquement,
- *˜EdgeGeom()* : destructeur,

- *EdgeGeom & operator= (const EdgeGeom & e)* : surcharge de l'opérateur = afin de permettre l'affectation d'une arête à une autre,

- *friend ostream & operator<<(ostream & f, const EdgeGeom & a)* : surcharge de l'opérateur d'affichage afin de permettre l'impression écran d'un objet *EdgeGeom*.

- *double dist_edgepoint(const Point2D &)* : méthode pour calculer la distance d'une arête à un point.

#### 3.2.1.3 Classe Cell

Les points et les arêtes étant connus, la cellule peut désormais être construite soit comme un ensemble de points, soit comme un ensemble d'arêtes. Une arête étant le bord d'un polygône, nous avons retenu la définition d'une cellule comme étant un ensemble d'arêtes. Dans le cas d'un maillage tridimensionnel, une cellule pourra être de la même façon définie comme un ensemble de faces, elles-mêmes définies comme un ensemble d'arêtes.

Les champs permettant la description d'une cellule sont alors son nombre de sommets, son nombre de côtés et ses arêtes. Une cellule est également caractérisée par son centre et son aire.

Il est important de noter que cette définition permet n'importe quelque type de forme de cellule. Il est donc possible de définir des classes plus spécifiques décrivant des géométries de cellules (triangles, quadrilatères, hexagones, etc.). Ces classes hériteront alors des propriétés de la classe *Cell* et pourront contenir des champs supplémentaires pour une caractérisation plus poussée. Dans le cadre de ce projet, seule une classe *Quadrangle* de quadrilatères a été construite. Elle est présentée à la suite de la classe *Cell*.

Protégé :

- *int nbpts* : nombre de points,

- *int nbedge* : nombre d'arêtes,

- *Point2D centre* : centre de la cellule,

- *EdgeGeom ** edges* : pointeurs sur les arêtes,

- *double area* : aire de la cellule.

Publique :

- *Cell()* : constructeur par défaut,

- *int & NbEdge()* : accès à la valeur du nombre d'arêtes avec possibilité de modification,

- *int & NbPts()* : accès à la valeur du nombre de points avec possibilité de modification,

- *Point2D & Centre()* : accès au point qui est au centre de la cellule avec possibilité de modification,

- *EdgeGeom** Edges()* : accès au pointeur sur les arêtes,
- *double & Area()* : accès à la valeur de la surface avec possibilité de modification,
- *friend ostream & operator<<(ostream & f, const Cell & m)* : surcharge de l'opérateur d'affichage afin de permettre l'impression écran d'un objet *Cell*.

#### 3.2.1.3.1 Classe Quadrangle

La classe *Quadrangle* hérite de la classe *Cell*. Les champs protégés de la classe mère *Cell* (nombre de sommets, d'arêtes, etc.) sont initialisés dans les constructeurs de la classe *Quadrangle*. La classe *Quadrangle* est dotée des méthodes spécifiques aux quadrilatères permettant d'initialiser les champs tels que l'aire ou le centre de la cellule.

Publique :

- *void calc_area()* : méthode calculant l'aire du quadrilatère,
- *void calc_centre()* : méthode permettant le calcul du centre de la cellule,
- *Quadrangle()* : constructeur par défaut. Seuls le nombre de sommets et le nombre d'arêtes sont initialisés. Le pointeur sur les arêtes est créé mais il ne pointe sur rien,
- *Quadrangle(Point2D & p0, Point2D & p1, Point2D & p2, Point2D & p3)* : constructeur pour lequel les quatre sommets du quadrilatère sont donnés,
- *Quadrangle(EdgeGeom & e0, EdgeGeom & e1, EdgeGeom & e2, EdgeGeom & e3)* : constructeur pour lequel les quatre côtés du quadrilatère sont donnés,
- *˜Quadrangle()* : destructeur.

**Remarque 5** *La classe Quadrangle est fondée sur une série d'hypothèses permettant de simplifier la construction d'un objet de type Quadrangle :*

1. le calcul du centre d'un quadrilatère fait l'hypothèse de cellules rectangulaires. Pour traiter des formes quelconques de quadrilatères, il est nécessaire de revoir la méthode de calcul du centre,
2. la construction d'un quadrilatère à partir de la donnée des sommets ou des arêtes se fait suivant un ordre prédéfini (fig. 8 )

### 3.2.2 Les propriétés physico-chimiques du milieu et des espèces radioactifs

Les propriétés physico-chimiques du milieu et des espèces radioactives ont été regroupées à l'intérieur de la classe *Properties*. Ces propriétés sont subdivisées entre propriétés spécifiques au milieu (classe *Media*), propriétés spécifiques á l'espèce radioactive considérée (classe *Species*) et enfin propriétés liées à la fois à l'espèce et au milieu (définies comme champs de la classe *Properties*). Un objet de type *Properties* sera ensuite affecté à chaque élément d'une grille.

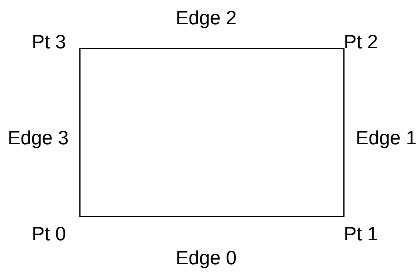


Figure 8: Numérotation par sommets et arêtes pour la classe *Quadrangle*.

**3.2.2.1 Classe Properties** La classe *Properties* est construite de la façon suivante :
Protégée :

- *Species *species* : pointeur vers un objet de la classe *Species*,
- *Media *media* : pointeur vers un objet de la classe *Media*,
- *double kxx_ , kyy_* : coefficients de diffusion suivant les directions X et Y (termes diagonaux du tenseur des perméabilités),
- *double kxy_* : coefficients de diffusion croisés (termes non-diagonaux du tenseur des perméabilités). Le tenseur des perméabilités étant symétrique (et également défini et positif), seul le terme *double kxy_* est donné,
- *double r_* : facteur de retard.

Publique :

- *Properties()* : constructeur par défaut,
- *Properties(Properties & prop)* : constructeur par copie,
- *Properties(Media * med, Species * spe, double kk_, double rr_)* : constructeur avec donnée des propriétés sur l'espèce radioactive, sur le milieu, du tenseur de perméabilité et du facteur de retard. Attention : dans ce constructeur, la perméabilité est donnée par un seul coefficient. Il convient de passer à une structure de type vectorielle pour entrer le tenseur des perméabilités,
- *˜Properties()* : destructeur,
- *Species & Species_val()* : méthode renvoyant les propriétés liées à l'espèce radioactive,
- *Media & Media_val()* : méthode renvoyant les propriétés liées au mileu poreux,
- *double Kxx_()* : méthode renvoyant la perméabilité suivant la direction X,
- *double Kyy_()* : méthode renvoyant la perméabilité suivant la direction Y,

- *double Kxy_()* : méthode renvoyant le terme perméabilité croisé,
- *double R_()* : méthode renvoyant le facteur de retard,
- *void SetProper(const double omeg, string nname, const double llife_t, const double k, const double r)* : méthode permettant l'initialisation des propriétés. Cette initialisation comprend l'initialisation des sous-classes *Species* et *Media*,
- *friend ostream & operator<<(ostream & f, const Properties & p)* : méthode d'affichage d'un objet de type *Properties*.

**Remarque 6** *On peut faire les remarques suivantes :*

1. les termes du tenseur des perméabilités sont entrés un par un dans la classe *Properties*. Il serait plus judicieux d'utiliser un objet vectoriel de type matrice pour stocker ces termes afin de respecter la nature physique et mathématique du tenseur de perméabilités.
2. dans un premier temps, la perméabilité est supposée constante suivant toutes les directions (isotropie : $k_{xx} = k_{yy}$ et $k_{xy} = k_{yx} = 0$).
3. par défaut le facteur de retard est fixé à 1 afin de conserver le terme de variation en temps et le terme de décroissance radioactive dans l'équation de convection-diffusion.

Nous décrivons à présent les sous-classes *Media* et *Species*.

**3.2.2.2 Classe Media** La classe *Media* possède les champs suivants :
Protégé :

- *double omega* : porosité effective du milieu poreux.

Publique :

- *Media()* : constructeur par défaut. Voir remarque (7),
- *Media(const Media & med)* : constructeur par copie,
- *Media(double & omeg)* : constructeur par donnée de la porosité,
- *˜Media()* : destructeur,
- *double Omega()* : méthode renvoyant la valeur $\omega$,
- *double SetOmega(double omeg)* : méthode permettant de fixer $\omega$

**Remarque 7** *Par défaut, la porosité $\omega$ est fixée à 1.*

#### 3.2.2.3 Classe Species

La classe *Species* est définie comme suit :

Protégé :

- *string name* : nom de l'espèce radioactive considérée,
- *double life_t* : temps de vie de l'espèce,
- *double lambda* : facteur représentatif du temps de vie,

Publique :

- *Species()* : constructeur par défaut.
- *Species(const Species & sp)* : constructeur par copie,
- *Species(string & nname, double & llife_t)* : constructeur avec donnée du nom de l'espèce et de son temps de vie,
- *~Species()* : destructeur,
- *string Name()* : méthode renvoyant le nom de l'espèce radioactive,
- *double Life_t()* : méthode renvoyant le temps de vie de l'espèce,
- *void calc_Lambda()* : méthode effectuant le calcul de $\lambda$,
- *const double Lambda()* : méthode renvoyant la valeur de $\lambda$,
- *void SetName(string nname)* : méthode d'initialisation du nom de l'espèce,
- *double SetLifeT(double llife_t)* : méthode d'initialisation du temps de vie de l'espèce.

**Remarque 8** *Le temps de vie d'une espèce est mis par défaut à -1 pour forcer l'utilisateur à le définir parmi les données d'entrée.*

### 3.2.3 Les conditions aux limites

Les conditions aux limites sont implémentées à l'aide de la classe *BoundaryCond*. Elles sont de type Fourier généralisé et sont définies par la donnée du couple $(\alpha, \beta)$, $\alpha$ étant le coefficient du terme d'ordre zéro et $\beta$ le coefficient du terme dérivé. Le détail de ces conditions aux limites est donné dans le paragraphe 2.1.5.

Privé :

- *double alpha* : coefficient du terme d'ordre zéro,
- *double beta* : coefficient devant la dérivée normale.

Publique :

- *double & Alpha()* :
- *double & Beta()* :
- *BoundaryCond()* : constructeur par défaut. Les valeurs de $\alpha$ et $\beta$ sont fixées à zéro,
- *BoundaryCond(const double a, const double b)* : constructeur par donnée de $\alpha$ et $\beta$,
- *BoundaryCond(const BoundaryCond & BC)* : constructeur par copie,
- *˜BoundaryCond()* : destructeur,
- *void SetBCond(const double alf, const double bet)* : méthode d'initialisation des valeurs de $\alpha$ et $\beta$,
- *friend ostream & operator<<(ostream & f, const BoundaryCond & b)* : méthode d'affichage d'un objet de type *BoundaryCond*.

### 3.2.4 La classe Element et les classes EdgeInternal, EdgeBoundary et EdgeInterface

Des classes supplémentaires ont été construites pour faire le lien entre la géometrie du problème, les propriétés du milieu, des espèces radioactives et les conditions aux limites du problème. Ce sont ces classes qui décrivent complètement l'approche mathématique du problème par la méthode des volumes finis.

Ces classes sont au nombre de quatre :

- classe *Element*,
- classe *EdgeInternal*,
- classe *EdgeBoundary*,
- classe *EdgeInterface*.

Ces classes sont décrites plus en détail dans les paragraphes suivants.

#### 3.2.4.1 Classe Element

Privé :

- *int idne* : numéro d'identité de l'élément,
- *Cell cell_* : structure géométrique de l'élément,
- *Properties * proper* : pointeur sur un objet de type *Properties*.

Publique :

- *Element()* : constructeur par défaut. Le numéro d'identité est fixé à -1 par défaut,

- *Element(Quadrangle & q, int iidne = -1)* : constructeur. La forme de la cellule géométrique est donnée comme étant un quadrilatère,
- *Element(const Element & elt)* : constructeur par copie,
- *int & Idne()* : méthode renvoyant le numéro d'identité de l'élément,
- *Cell & Cell_()* : méthode renvoyant la structure géométrique de l'élément,
- *Properties & Proper()* : renvoie un pointeur sur la propriété de l'élément,
- *Cell & setCell(Cell& c, int id)* : méthode permettant de construire la structure géométrique de l'élément après avoir créé un objet de type *Element*,
- *Properties & setProperties(Properties& p)* : méthode de réinitialisation des propriétés d'un élément,
- *˜Element()* : destructeur,
- *friend ostream & operator<<(ostream & f, const Element & e)* : affichage d'un objet de type élément.

**Remarque 9** *Les éléments qui sont dans un trou du domaine (cf. figure 5) peuvent être considérés comme des éléments ayant des propriétés particulières. Leur perméabilité (non physique) est par exemple mise à -1. Cette possibilité permet de conserver la* **structure** *éventuelle du maillage, et ainsi de simplifier le traitement informatique. Nous modéliserons de façon systématique dans nos tests les trous par des éléments "vides".*

#### 3.2.4.2 Classe EdgeInternal

La classe *EdgeInternal* est définie par la donnée d'une arête géométrique, de deux pointeurs sur les deux voisins de l'aête. Elle permet la description des arêtes se trouvant dans l'intérieur du domaine (par distinction des arêtes se trouvant sur le bord ou sur une interface).

Privé :

- *EdgeGeom * edge_* : pointeur sur une arête géométrique,
- *Element * neigh1, neigh2* : pointeurs sur les éléments voisins de l'arête interne.

Publique :

- *EdgeGeom & Edge_()* : retourne le pointeur sur l'arête géométrique correspondante,
- *EdgeGeom & setEdge(EdgeGeom& e)* : méthode permettant l'initialisation de l'arête géométrique de l'arête interne,
- *Element & Neigh1(), Element & Neigh2()* : méthode renvoyant les pointeurs sur les éléments voisins d'une arête interne,

- *EdgeInternal()* : constructeur par défaut,
- *EdgeInternal(EdgeGeom & edge,Element & n1, Element & n2)* : constructeur pour lequel la structure géométrique de l'arête ainsi que ses voisins sont donnés,
- *EdgeInternal(EdgeInternal & ei)* : constructeur par copie,
- *~EdgeInternal()* : destructeur,
- *friend ostream & operator<<(ostream & f, const EdgeInternal & edgeit)* : opérateur d'affichage d'une arête interne,
- *void setNeigh(Element & e1, Element & e2)* : méthode permettant l'initialisation des éléments voisins d'une arête interne.

#### 3.2.4.3 Classe EdgeBoundary

La classe *EdgeBoundary* est définie par la donnée d'un objet géométrique, d'un pointeur sur l'unique voisin de l'arête et d'un pointeur sur une condition limite. Elle permet la description des arêtes se trouvant sur le bord du domaine.

Privé :

- EdgeGeom * edge_ : pointeur sur une arête géométrique,
- Element * neigh : pointeur sur le voisin de l'arête interne,
- BoundaryCond * bound_ : pointeur sur une condition limite.

Publique :

- *EdgeGeom & Edge_()* : méthode renvoyant le pointeur sur la structure géométrique de l'arête,
- *EdgeGeom & setEdge(EdgeGeom& e)* : méthode permettant l'initialisation de la structure géométrique de l'arête,
- *Element & Neigh()* : méthode renvoyant le voisin de l'arête,
- *void setNeigh(Element & e)* : affectation du voisin à l'arête,
- *BoundaryCond & Bound_()* : méthode renvoyant le pointeur sur la condition limite de l'arête,
- *BoundaryCond & setBound(BoundaryCond & bc)* : affectation de la condition limite à l'arête,
- *EdgeBoundary()* : constructeur par défaut,

- *EdgeBoundary(EdgeGeom * edge, Element * elt, BoundaryCond * bound)* : constructeur avec donnée de la géométrie, du voisin et de la condition limite,
- *~EdgeBoundary()* : destructeur,
- *friend ostream & operator<<(ostream & f, const EdgeBoundary & edgeb)* : affichage d'un objet de type *EdgeBoundary*.

#### 3.2.4.4 Classe EdgeInterface

La classe *EdgeInterface* reprend la même structure que la classe *EdgeBoundary*, les conditions aux limites étant de type Fourier pour assurer le fonctionnement de l'algorithme de décomposition de domaine choisi.[5] La classe *EdgeInterface* est cependant complétée par des champs et des méthodes liés au traitement particulier des conditions d'interfaces (cf. 2.1.5). Ainsi, pour traiter l'interface, il faut :

1. connaître la table de connectivité des arêtes. En effet, chaque arête se trouvant sur une interface doit connaître le numéro du ou des sous-domaines voisins ainsi que le numéro des arêtes du ou des sous-domaines voisins ayant avec elle une partie commune. Cette connectivité est donnée par la structure de données *EdgeIntData*.
2. pour chaque arête appartenant à une interface, il faut connaître la longueur de l'intersection avec chaque arête en vis-à-vis et appartenant au(x) sous-domaine(s) voisin(s).

Privé :

- *EdgeGeom * edge_* : pointeur sur une arête géométrique,
- *Element * neigh* : pointeur sur le voisin de l'arête interne,
- *BoundaryCond * bound_* : pointeur sur une condition limite, qui sera de type Fourier,
- *vector<EdgeIntData> facing_edges* : table de connectivité (utilisée avec la librairie STL).

*EdgeIntData* est une structure contenant 3 champs décrivant comment est reliée l'arête courante à une autre arête d'un sous-domaine voisin. Dans l'exemple de la figure (6), l'arête courante est $\tau$, et un *EdgeIntData* décrit sa relation avec l'arête commune $\tau_l^1$ du sous-domaine $\Omega_l$ :

- *int NeighbDom* : numéro du sous-domaine voisin (ex. : $l$),
- *int EdgeIntNum* : indice de l'arête d'interface dans la numérotation du sous-domaine voisin (ex. : 1),
- *double SharedLength* : longueur commune de l'arête courante et de l'arête du sous-domaine voisin (ex. : $|\tau \cap \tau_l^1|$).

[5] Il est à noter qu'on peut simplement changer le type de conditions aux limites et ainsi obtenir d'autres algorithme, comme Neumann-Neumann par exemple.

Retournons à la classe *EdgeInterface*:
Publique :

- *EdgeGeom & Edge_()* : méthode renvoyant le pointeur sur la structure géométrique de l'arête,
- *EdgeGeom & setEdge(EdgeGeom & e)* : méthode permettant l'initialisation de la structure géométrique de l'arête,
- *Element & Neigh()* : méthode renvoyant le voisin de l'arête,
- *void setNeigh(Element & e)* : affectation du voisin à l'arête,
- *BoundaryCond & Bound_()* : méthode renvoyant le pointeur sur la condition limite de l'arête,
- *vector<EdgeIntData> & oppositeedges()* : méthode renvoyant la table de connectivité de l'arête,
- *EdgeInterface()* : constructeur par défaut,
- *EdgeInterface(EdgeGeom & edge, Element & elt, BoundaryCond & bc)* : constructeur par donnée des champs de la classe.
- *BoundaryCond & setBound(BoundaryCond & bc)* : affectation de la condition limite à l'arête,
- ~*EdgeInterface()* : destructeur,
- *friend ostream & operator<<(ostream & f, const EdgeInterface & edgeif)* : affichage d'un objet de type *EdgeInterface*.

### 3.2.5 La librairie BibC

La librairie BibC a été fournie par Frédéric Nataf. Elle contient des classes et des méthodes permettant une gestion simple et sûre d'objets algébriques tels que des vecteurs ou des matrices. Parmi les fonctions particulièrement utiles de cette librairie, notons la possibilité de contrôler la dimension de ces objets algébriques et la possibilité de les redimensionner et d'affecter l'indice de départ et l'indice de fin des tableaux à des valeurs quelconques. Cette dernière fonctionnalité est notamment très utile lorsque des tableaux sont extraits de structures vectorielles plus grandes. En conservant les indices globaux qui étaient affectés à ces tableaux, il est ainsi possible de le replacer très facilement dans des structures indexées de la même façon que les structures vectorielles dont ces vecteurs sont originiaires.

### 3.2.6 La classe SubDomain

La classe *SubDomain* est la classe dans laquelle se fait véritablement le lien entre toutes les classes qui ont été présentées jusqu'à présent. La résolution numérique du problème se fait en effet au niveau du sous-domaine par la nature même du problème. En effet si les effets de convection peuvent être résolus localement par connaissance des seuls voisins d'un élément, les phénomènes de diffusion sont globaux et imposent la résolution d'un système linéaire. Cette classe contient donc les objets et les méthodes nécessaires à la résolution du problème sur un sous-domaine. Ces objets peuvent être de type algébrique (vecteurs, matrices, etc.) ou bien physiques (propriétés du milieu et des espèces radioactives). Les méthodes les plus importantes concernent la construction du maillage, l'assemblage de la matrice du système et l'assemblage du second membre de ce même système.

Notons dès à présent que la formulation mathématique du problème nous permet d'éliminer les inconnues de type flux ou concentration sur les bords du domaine. De ce fait, la matrice du système est symétrique (cf. remarque 3).

#### 3.2.6.1 Construction du maillage

La construction du maillage que nous avons réalisé suit la numérotation lexicographique décrite par les figures 9 et 10. La construction des points, des arêtes et des cellules se fait de façon structurée. Cependant, en utilisant des numéros d'identification (comme implémenté dans les classes), il est possible d'utiliser un mailleur et une numérotation quelconque.

#### 3.2.6.2 Assemblage de la matrice

L'assemblage de la matrice correspondant à l'équation de conservation (6) où les flux sont donnés par (11) et (14), se fait en quatre temps. Les notations sont celles de la section (2.1).

1. Boucle sur les éléments : assemblage des termes d'ordre zéro (termes en $\omega \cdot \lambda \cdot |K|$).

   **Remarque 10** *Cette boucle est effectuée sur tous les éléments, y compris ceux qui sont dans un trou du domaine. On appelle ces éléments fictifs des éléments* **"vides"**, *pour lesquels la perméabilité a été mise à une valeur négative (non physique). Si un de ces éléments vides est rencontré dans la boucle, le terme diagonal correspondant est mis à 1 pour éviter des problèmes de conditionnement.*

2. Boucle sur les arêtes internes : on calcule le terme $-|\sigma| \frac{1}{\frac{d_{K,\,\sigma}}{D^i_{K,\,\sigma}} + \frac{d_{L,\,\sigma}}{D^i_{L,\,\sigma}}}$ du flux interne $\Phi^i_{K,\,\sigma}$.

   La contribution des arêtes internes est ensuite introduite sur les termes diagonaux et sur les composantes de la matrice liées au terme diagonal (connectivité entre éléments voisins).

3. Boucle sur les arêtes de bord : on commence par calculer le coefficient $-|\tau| \frac{\alpha\,\xi_{K,\,\sigma}}{\alpha + \beta\,\xi_{K,\,\sigma}}$ du terme $\Phi^i_{K,\,\tau}$, afin d'obtenir la contribution des arêtes de bord. Puis, on corrige les terme diagonaux correspondant aux éléments liés à ces arêtes.


Figure 9: Numérotation lexicographique utilisée dans NestedVF

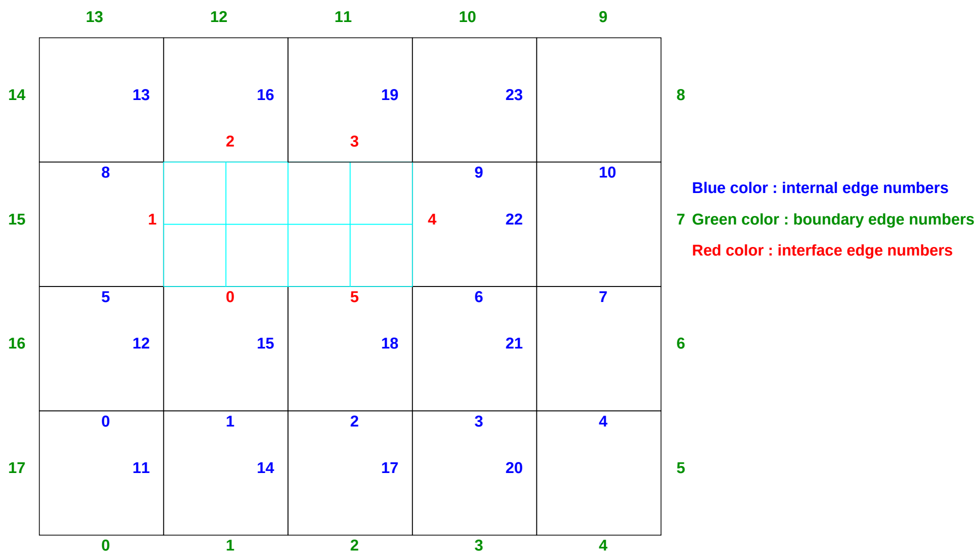


Figure 10: Numérotation lexicographique des arêtes internes, des arêtes de bord et des arêtes à l'interface.

4. Boucle sur les arêtes d'interface : on commence par calculer le coefficient $-|\tau|\frac{\alpha\,\xi_{K,\,\sigma}}{\alpha+\beta\,\xi_{K,\,\sigma}}$ du terme $\Phi^{i}_{K,\,\tau}$, afin d'obtenir la contribution des arêtes d'interface. Puis, on corrige les terme diagonaux correspondant aux éléments liés à ces arêtes.

#### 3.2.6.3 Assemblage du second membre

Cet assemblage se fait en trois temps.

1. Boucle sur les éléments : assemblage des sources volumiques.

   **Remarque 11** *De même que pour l'assemblage de la matrice, cette boucle sur les éléments est effectuée sur tous les éléments, y compris les éléments vides (pour lesquels la perméabilité à été mise à une valeur négative). Si un de ces éléments vides est rencontré dans la boucle, le terme du second membre correspondant est mis à 0. La solution sera donc* 0 *dans cette maille "vide" artificielle.*

2. Boucle sur les arêtes de bord : suivant la valeur de $\alpha$, les termes du second membre correspondant aux arêtes de bord sont calculés différemment. La distinction est faite entre les conditions aux limites de type Neumann et les autres (cf. remarque 2) : si $\alpha = 0$ (Neumann), on met dans le second membre le flux (produit a priori non nul $(\alpha\ \bar{u})$). Si $\alpha$ est strictement positif, alors la valeur de $\bar{u}$ est multipliée par son facteur $\frac{\alpha\,\xi_{K,\,\sigma}}{\alpha+\beta\,\xi_{K,\,\sigma}}$.

3. Boucle sur les arêtes d'interface : les termes du second membre correspondants aux arêtes d'interface sont calculés exactement de la même façon que pour les arêtes de bord.

#### 3.2.6.4 Contenu de la classe SubDomain

Privé :

- *int idns* : numéro d'identification du sous-domaine,
- *int nbpt* : nombre total de points dans le sous-domaine,
- *int nbedgeom* : nombre total d'arêtes géométriques,
- *int nbelt* : nombre total d'éléments,
- *int nbedgbound* : nombre d'arêtes de bord,
- *int nbedginterf* : nombre d'arêtes d'interface,
- *int nbedginternal* : nombre d'arêtes internes,
- *int nbprop* : nombre de propriétés définies sur le sous-domaine,
- *int hbdwidth* : demi-largeur de bande de la matrice du système,

Publique :

- *verygcolect< Point2D > pt* : liste des points du sous-domaine,
- *verygcolect< EdgeGeom > edge* : liste des arêtes géométriques du sous-domaine,
- *verygcolect< Quadrangle > quad* : liste des quadrilatères du sous-domaine,
- *verygcolect< Element > elt* : liste des éléments du sous-domaine,
- *verygcolect< EdgeBoundary > edgeb* : liste des arêtes de bord du sous-domaine,
- *verygcolect< EdgeInterface > edgeif* : liste des arêtes d'interface du sous-domaine,
- *verygcolect< EdgeInternal > edgeit* : liste des arêtes internes au sous-domaine,
- *verygcolect< Properties > prop* : liste des propriétés définies sur le sous-domaine,
- *skygmatrix<double> mat* : matrice du système,
- *gvector<double> U* : vecteur des inconnues,
- *gvector<double> RHS* : vecteur second membre du système ($mat \cdot U = RHS$),
- *gvector<double> RHSaux* : vecteur auxiliaire dans lequel sont stockés tous les termes du second membre qui sont invariables au cours de la résolution du problème de décomposition de domaine. Il contient donc les contributions des termes source volumiques (RhsVol) et de conditions aux limites "externes" (c'est-à-dire ne venant pas de l'interface : RhsEBC).
- *gvector<double> RhsVol* : vecteur des contributions des sources volumiques,
- *gvector<double> RhsEBC* : vecteur des contributions des conditions aux limites (bords "externes" uniquement, qui n'incluent pas l'interface avec d'autres sous-domaines),
- *gvector<double> InIBC* : vecteur des contributions des conditions aux limites sur les interfaces. Contributions en *INPUT*, c'est-à-dire en *entrée*, pour l'opérateur $T_k$ du domaine courant $\Omega_k$. Ce terme correspond au $\lambda_k$ de l'équation (21). Donc par définition, pour le sous-domaine $k$ :

$$\text{InIBC}(k) = \left(\frac{\partial p_k}{\partial n_k} + \alpha p_k\right)|_{Interface(\Omega_k)}. \tag{26}$$

- *gvector<double> OutIBC* : vecteur des contributions des conditions aux limites sur les interfaces. Contributions en *OUTPUT*, c'est-à-dire en *sortie*, pour l'opérateur $T_k$ du domaine courant $\Omega_k$. Ce terme correspond au terme de droite de l'équation (19). Donc par définition, pour le sous-domaine $k$ :

$$\text{OutIBC}(k) = \left(-\frac{\partial p_k}{\partial n_k} + \alpha p_k\right)|_{Interface(\Omega_k)}. \tag{27}$$

- *gvector<double> OutIBCaux* : vecteur auxiliaire permettant de conserver la valeur de *OutIBC* d'une itération à la suivante.
- *SubDomain()* : constructeur par défaut,
- *SubDomain(SubDomain & sub)* : constructeur par copie,
- *~SubDomain()* : destructeur,
- *void SimpleBuildPoint2D(gvector< double > x, gvector< double > y)* : méthode permettant la construction des points d'une grille simple structurée,
- *void BuildEdgeGeom(verygcolect< EdgeGeom > & egde)* : méthode permettant la construction des arêtes géométriques d'une grille simple structurée,
- *void BuildQuadrangle(verygcolect< Quadrangle > & quad)* : méthode permettant la construction des quadrilatères d'une grille simple structurée,
- *void BuildElement(verygcolect< Element > & elt)* : méthode permettant la construction des arêtes géométriques d'une grille simple structurée,
- *void BuildMesh(gvector<double > x, gvector< double > y)* : méthode permettant de construire le maillage du sous-domaine,
- *void BuildEdgeInternal()* : méthode permettant la construction des arêtes internes dans le cas où le domaine ne contient aucun trou,
- *void BuildEdgeInternalWithHole(icouple indBL, icouple indTR)* : méthode permettant la construction des arêtes internes, dans le cas où le domaine contient un trou,
- *void BuildEdgeInterface()* : méthode permettant la construction des arêtes d'interface,
- *void BuildEdgeBoundaryOut()* : méthode permettant la construction des arêtes de bord extérieures,
- *void BuildEdgeBoundaryIn(icouple indBL, icouple indTR)* : méthode permettant la construction des arêtes de bord internes
- *void setDimensions()* : méthode d'initialisation des dimensions du problème sur le sous-domaine,
- *void FillMat()* : méthode de remplissage de la matrice du système,

- *void Facto()* : méthode de factorisation LU du système,
- *void Solve()* : méthode permettant la résolution du système,
- *void FillRhsSourceAndEBC()* : méthode permettant de remplir le second membre auxiliaire RHSaux provenant des termes source et des conditions aux limites “externes” (vecteurs *RhsVol* et *RhsEBC*).
- *void UpdateInIBC( verygcolect< SubDomain * > & psubdom )* : méthode mettant à jour le vecteur d'entrée *InIBC* à partir des contributions des interfaces de tous les sous-domaines voisins (terme *OutIBCaux*).
- *void FillRhsInterface()* : remplit le vecteur second membre *RHS* à partir uniquement des contributions de l'interface (terme *InIBC*).
- *void ComputeOutIBC()* : méthode qui calcule la sortie (*OutIBC*) de l'opérateur $T_k$, à partir du vecteur solution (*U*) et des conditions aux limites d'entrée (*InIBC*).
- *void ComputeErrorInterfaceLinf()* : méthode qui calcule l'erreur maximale sur l'interface. Cette erreur est définie par la valeur absolue de la différence entre (*OutIBC*) et (*OutIBCaux*).
- *void PrintMat()* : méthode d'affichage de la matrice du système,
- *void PrintVector(gvector< double > vec)* : méthode d'affichage d'un vecteur du système,
- *friend ostream & operator « (ostream & f, const SubDomain & s)* : affichage d'un objet de type *SubDomain*.

D'autres méthodes ont été également définies pour permettre l'accès aux champs privés de la classe *SubDomain*.

## 3.3 Description de l'algorithme

Nous décrivons brièvement l'implémentation de l'initialisation et du produit (matrice $\times$ vecteur).

On se donne $N(\geq 2)$ sous-domaines $\Omega_k,\ k = 1..N$. Soit un vecteur global d'interface $X$ donné. Il se décompose en $N$ sous-vecteurs $X_k$ qui correspondent chacun aux degrés de liberté sur l'interface du sous-domaine $\Omega_k$.

Nous cherchons à résoudre le système matriciel $A * X = B$ décrit par (24) dans le cas de 2 sous-domaines (cf. section 2.2.2). La matrice $A$, pleine a priori, n'est pas explicitement formée. Pour la résolution, il suffit de pouvoir calculer le second membre $B$ dans une phase d'initialisation et de fournir le produit (matrice $\times$ vecteur) à une méthode de type Krylov (BiCGStab par exemple).

### 3.3.1 Phase d'initialisation

On calcule le second membre $B$ de l'équation (24).

For subdomain $k = 1..N$ :

$$\left| \begin{array}{l} \text{Form the sparse matrix } \textit{mat}(\text{k}), \\ \text{Factorize and store the matrix } \textit{mat}(\text{k}), \\ \text{Compute the } \textit{RHS}(\text{k}) \text{ vector, using only the Boundary Conditions and the source terms,} \\ \text{Solve the subdomain problem : } U(k) = mat(k) \backslash RHS(k), \\ \text{Compute the interface } \textit{OutIBC}(\text{k}) \text{ vector, using } U(k). \\ \text{Extension to the global vector : } B \longleftarrow \textit{OutIBC}(k). \end{array} \right. \tag{28}$$

On a utilisé les notations précédemment définies : pour les définitions de *InIBC et OutIBC*, voir les équations (26) et (27).

### 3.3.2 Produit matrice vecteur

On calcule le produit matrice vecteur : $Y = A * X$ correspondant à l'équation (24) dans le cas de 2 sous-domaines (cf. section 2.2.2). On ne construit pas explicitement la matrice $A$.

For $k = 1..N$ :

$$\left| \begin{array}{l} \text{Restriction to the interface of } \Omega_k : X_k \longleftarrow X \\ \textit{OutIBCaux}(\text{k}) \longleftarrow X_k \end{array} \right. \tag{29}$$

For $k = 1..N$ :

$$\left| \begin{array}{l} \text{Update } \textit{InIBC}(\text{k}) \text{ using the neigbouring } \textit{OutIBCaux}(\text{j} \neq \text{k}), \\ \text{Fill the right hand side vector } \textit{RHS}(\text{k}) \text{ using only the interface term } \textit{InIBC}(\text{k}), \\ \text{Solve the subdomain problem : } U(k) = mat(k) \backslash RHS(k), \\ \text{Compute the interface } \textit{OutIBC}(\text{k}) \text{ vector, using } U(k) \text{ and } \textit{InIBC}(\text{k}), \end{array} \right. \tag{30}$$

For $k = 1..N$ :

$$\left| \begin{array}{l} \text{Compute : } Y_k = \textit{OutIBCaux}(k) - \textit{OutIBC}(k), \\ \text{Extension to the global vector : } Y \longleftarrow Y_k \end{array} \right. \tag{31}$$

# 4 Validation du modèle et tests numériques

Le code a été testé sur les problèmes de type suivant :

$$R_i \omega \lambda_i C_i - \nabla \cdot (\mathbf{D}_i \nabla C_i) + = f_i \quad \text{dans } \Omega_k,$$
$$\text{Conditions limites sur } \partial\Omega_k$$

Le problème à résoudre est un problème stationnaire. Les conditions aux limites peuvent être de type Dirichlet, Neumann ou Robin suivant les valeurs fixées pour $\alpha$ et $\beta$ (voir 2.1.5).

## 4.1 Cas test 1

Le premier cas test correspond à un domaine unique sur lequel la construction des conditions limites, l'assemblage de la matrice, l'assemblage du second membre et l'algorithme de résolution du système linéaire sont validés. Des conditions limites de type Dirichlet sont prises sur les bords latéraux. Sur un de ces bords, la concentration de l'espèce radioactive est imposée à 1. Sur l'autre bord, la condition est mise à 0. Sur les bords inférieurs et supérieurs, des conditions de Neumann homogènes sont imposées. La figure 11 illustre la nature du problème.

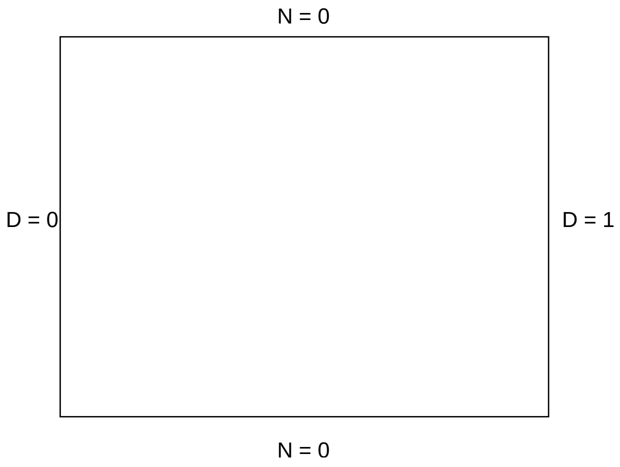


Figure 11: Configuration du cas test 1.

La solution attendue est une solution linéaire partant de zéro sur le bord du domaine sur lequel une condition de type Dirichlet homogène est imposée, et qui atteint la valeur 1 sur l'autre bord. Le maillage adopté est donné par la figure 12. Des grilles à résolution variable ont également été testées sans aucun problème.

La figure 13 illustre les résultats obtenus dans le cadre du problème 32.

Le comportement attendu est bien observé.

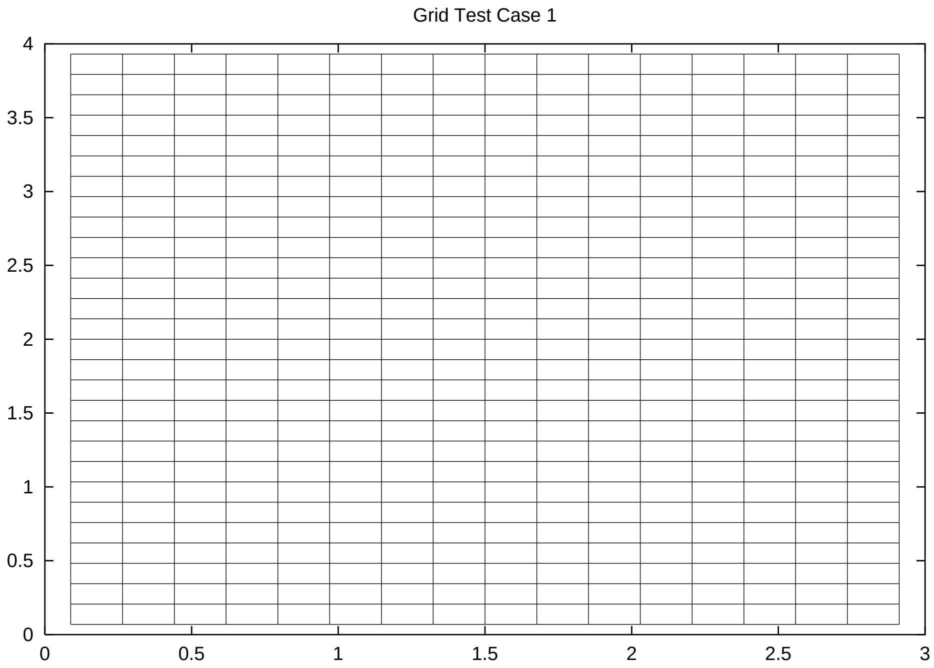


Figure 12: Discrétisation ramenée aux centre des éléments.

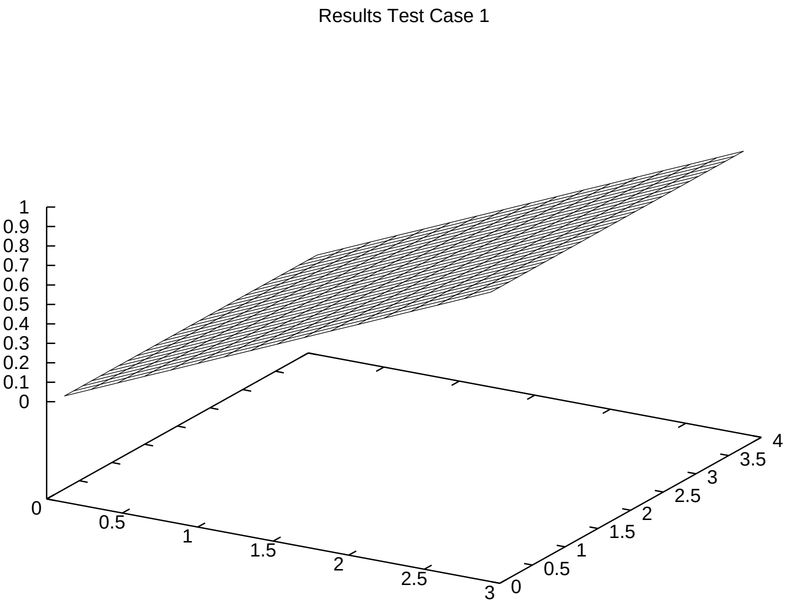


Figure 13: Résultat du cas test 1.

## 4.2 Cas test 2

Le cas test 2 est le cas d'une fracture dans le milieu. Une fracture est définie comme une zône du milieu poreux dans laquelle la porosité et la perméabilité sont plus élevées que dans le reste du domaine. La figure 14 illustre la nature du problème. La fracture est orientée suivant l'axe vertical. Des conditions limites de type Dirichlet sont prises sur les bords latéraux. Sur un de ces bords, la concentration de l'espèce radioactive est imposée à 1. Sur l'autre bord, la condition est mise à 0. Les bords inférieurs et supérieurs sont tout deux divisés en trois parties. Des conditions aux limites de type Dirichlet sont impossées sur les parties des bords correspondant à la fracture. Pour observer la diffusion le long de la fracture et dans le reste du domaine, on impose Dirichlet égal à 1 sur la partie supérieure de la fracture et Dirichlet égal à 0 sur la partie inférieure. Partout ailleurs, on fixe des conditions de type Neumann homogène. On s'attend à trouver une concentration croissant de façon pseudo-linéaire selon l'axe de la fracture et également selon l'axe perpendiculaire. Les deux évolutions à tendance linéaire se perturbent l'une l'autre pour former un comportement complètement 2 D.

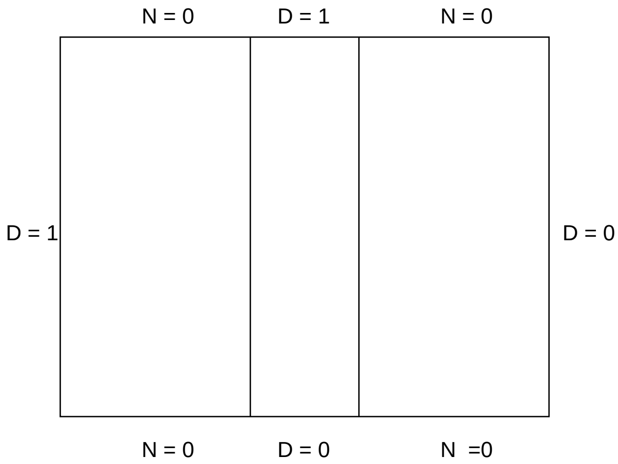


Figure 14: Configuration du cas test 2.

Le maillage adopté est donné par la figure 15. La grille est à résolution variable (la résolution est plus fine dans la fracture) mais elle ne possède de grille(s) emboîtée(s).

La figure 16 illustre les résultats fournis par le modèle NestedVF. Le comportement de la solution est exacte. Des tests ont été réalisés avec Matlab (schéma volumes finis). La comparaison des résultats donnés par Matlab et par notre code est excellente (différence point par point de l'ordre de $10^{-15}$ à $10^{-13}$ selon les maillages considérés) et par conséquent a permis de valider quantitativement notre modèle.

## 4.3 Cas test 3

Le cas test présenté ici est un carré avec un trou carré en son centre (cf. figure 17).

Le trou est introduit dans le maillage de façon virtuelle : les élements sont construits mais leur perméabilité est mise à une valeur négative pour préciser qu'ils ne rentreront pas dans le

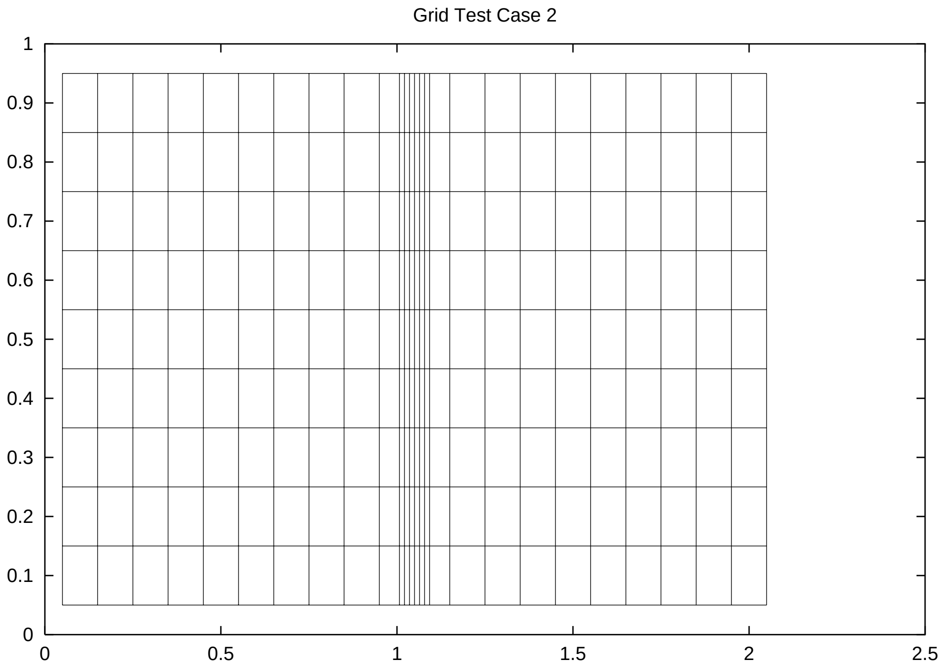


Figure 15: Discrétisation ramenée aux centre des éléments.

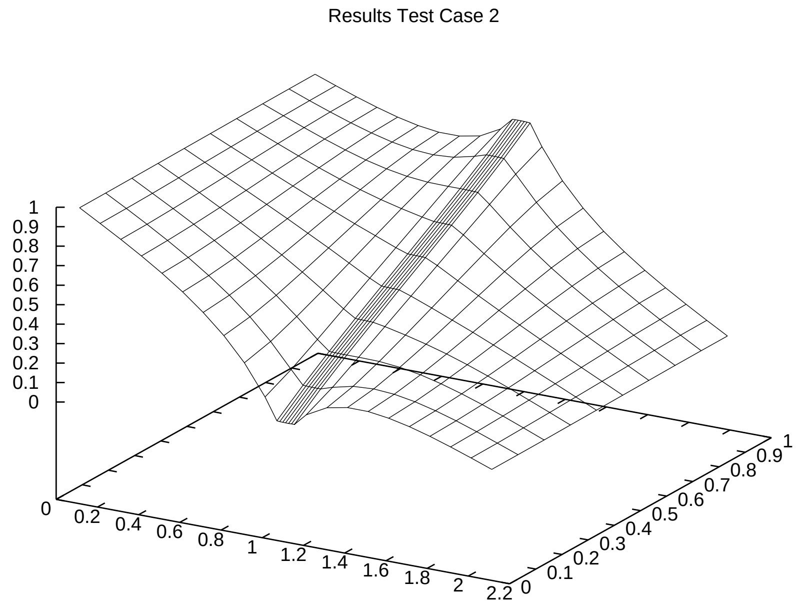


Figure 16: Résultat du cas test 2.

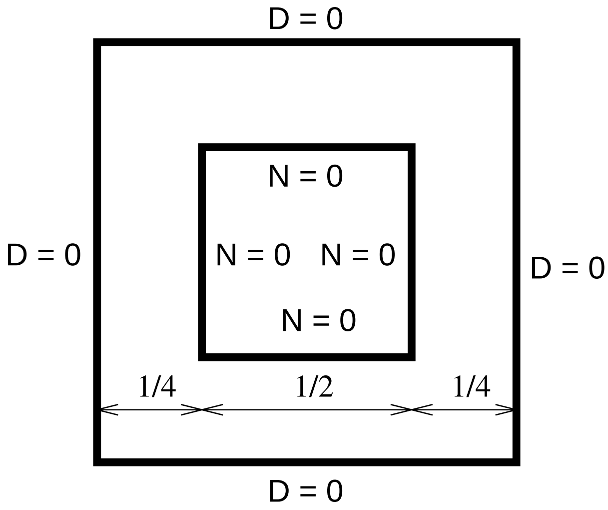


Figure 17: Description du cas test 3.

calcul de la solution.

Pour vérifier la validité de la méthode, on a choisi une solution analytique très simple :

$$u^* = \sin(2\pi x)\ \sin(2\pi y).$$

Le terme source associé est alors calculé :

$$f = -\Delta(u^*) = 8\pi^2 \sin(2\pi x)\ \sin(2\pi y).$$

On calcule également les conditions aux limites, qui sont extrêmement simples ici : Dirichlet homogène sur le bord extérieur, et sur le bord intérieur des conditions de Neumann homogènes sont imposées.

On obtient des résultats comme sur la figure (18). L'allure de la solution est tout à fait correcte. On remarque que le programme tient bien en compte le trou au milieu du domaine. La valeur affectée aux mailles dans le trou est bien celle attendue : $\bar{u} = 0$ dans le trou (cf. remarques 10 et 11).

Les résultats $\bar{u}$ sont comparés avec la solution analytique $u^*$. On retrouve la décroissance de l'erreur $L_2$ en $\mathcal{O}(h^2)$ attendue, ce qui permet de valider notre code.

## 4.4 Cas test 4

On teste l'algorithme et le code dans le cas de 2 sous-domaines avec maillages non raccordés en 2 dimensions.

### 4.4.1 Description

On résout le problème :

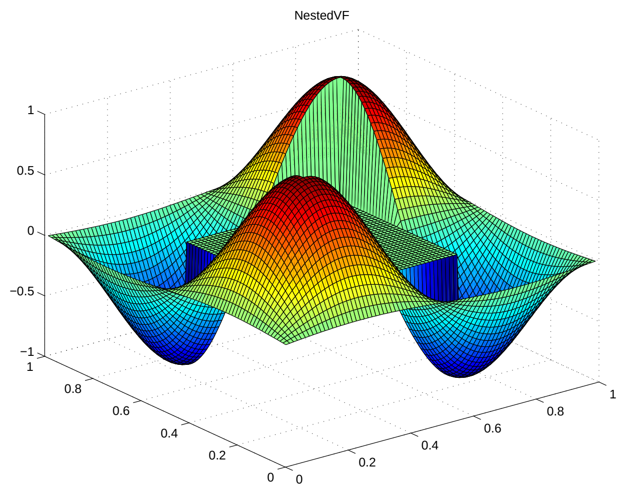


Figure 18: Résultats du cas test 3, maillage uniforme 80 x 80.

$$\begin{aligned} -\nabla \cdot (\nabla p) &= f \quad \text{dans } \Omega = [0,1] \times [0,1], \\ p &= g \quad \text{sur } \partial\Omega. \end{aligned} \tag{32}$$

Le critère d'arrêt pour l'algorithme est que la norme infinie du saut de la condition d'interface de Robin est inférieure à $10^{-8}$.

La solution numérique est comparée à la solution exacte. Nous prenons comme solution $p(x, y) = x^3y^2 + sin(xy)$. Le second membre est obtenu en appliquant l'opérateur à $u$ et en utilisant une intégration approchée : la formule de quadrature de Gauss avec quatre nœuds par cellule. De cette façon, l'erreur due à l'intégration inexacte sera petite devant l'erreur du schéma. Les normes $L_\infty$ et $H_h^1$ discrètes sont données pour une décomposition en 2 sous-domaines avec des grilles successivement raffinées.

1. Grilles $(5 \times 5 - 2 \times 2) * 2^n$, cf. table 19. Grilles emboîtées, maillages réguliers.

2. Grilles : $(5 \times 5 - 10 \times 10) * 2^n$. Grilles emboîtées, maillages réguliers.

3. Grilles : $(5 \times 5) * 2^n - (2 \times 2) * (2^n + 1)$. Grilles non emboîtées, maillages réguliers.

Les cas 1 et 2 correspondent à des maillages carrés réguliers avec un raffinement local dans le sous-domaine interne. La grille fine est un sous-maillage de la grille grossière. Les maillages sont donc qualifiés d'*emboîtés*. Le cas 3, au contraire, n'est pas *emboîté*.

Dans le premier cas, le maillage est 2 fois plus fin dans la zone raffinée que dans le reste du domaine. Dans le second, le maillage y est 10 fois plus fin. Dans ce cas, le raffinement local est donc très marqué.

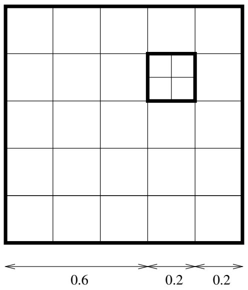


Figure 19: Description du cas test 4 : *maillage du cas 1*.

### 4.4.2 Résultats

On présente sur la figure (20) la solution du problème 2 avec $2^n = 4$. Les maillages sont réguliers : $20 \times 20$ pour le sous-domaine externe et $40 \times 40$ pour le sous-domaine interne. Il faut remarquer que l'on trace la solution comme superposition de la solution de chaque sous-domaine. Chaque solution est tracée sur la grille *entière*. Il existe donc une superposition des 2 surfaces à l'emplacement du sous-domaine interne. De plus, comme le sous-domaine externe comporte un "trou" (à l'emplacement de l'autre sous-domaine), il y a un décrochement de la surface : la solution passe brutalement à la valeur $0$ (cf. remarque $11$). D'autre part, la solution du sous-domaine interne apparaît noire car le maillage est trop fin pour la résolution de l'image (grille $40 \times 40$).

Les figures (21) et (22) montrent les erreurs en norme $L_\infty$ dans chaque sous-domaine.

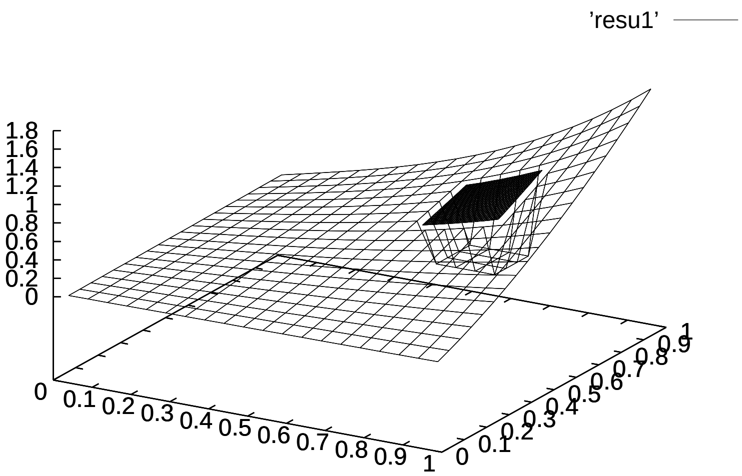


Figure 20: Résultat du cas test 4 : *cas 2*, $2^n = 4$.

On calcule l'ordre de décroissance de l'erreur $H_h^1$ d'une grille de pas $h1$ à la suivante de pas

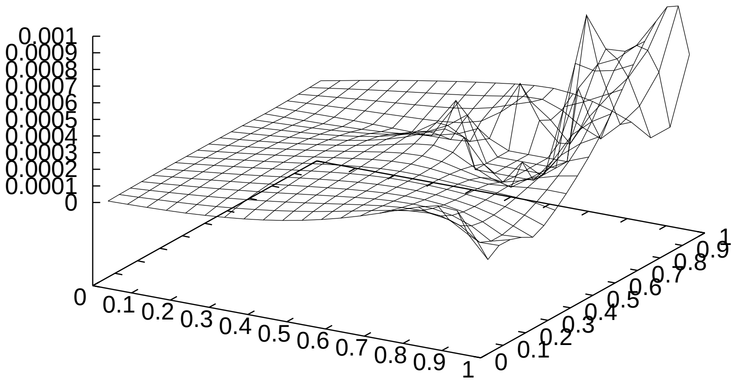


Figure 21: Erreur max du cas test 4 : *cas 2, sous-domaine externe*, $2^n = 4$.

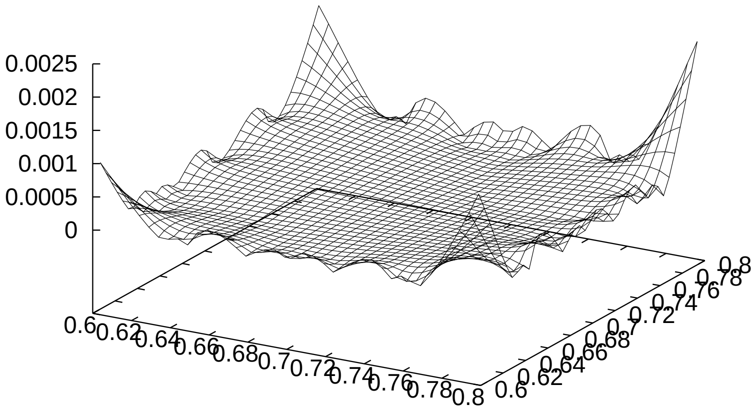


Figure 22: Erreur max du cas test 4 : *cas 2, sous-domaine interne*, $2^n = 4$.

$h2$. On appelle $\eta$ le réel tel que :

$$\frac{||\text{erreur}||_{H^1_{h1}}}{||\text{erreur}||_{H^1_{h2}}} = \left(\frac{h1}{h2}\right)^\eta$$

| $2^n$ | $1/h$ | $\|\|\cdot\|\|_\infty$ | $\|\|\cdot\|\|_{H^1_h}$ | $H^1_h$ error reduction | $\eta$ |
|---|---|---|---|---|---|
| 1 | 5 | $1.021E-2$ | $2.397E-2$ | | |
| 2 | 10 | $3.610E-3$ | $9.932E-3$ | 2.41 | 1.27 |
| 4 | 20 | $1.071E-3$ | $4.527E-3$ | 2.19 | 1.13 |
| 8 | 40 | $3.053E-4$ | $1.638E-3$ | 2.76 | 1.47 |
| 16 | 80 | $8.519E-5$ | $5.897E-4$ | 2.77 | 1.47 |
| 32 | 160 | $2.357E-5$ | $2.108E-4$ | 2.80 | 1.48 |

Table 4: Cas 1 : erreur en fonction du raffinement.

| $2^n$ | $1/h$ | $\|\|\cdot\|\|_\infty$ | $\|\|\cdot\|\|_{H^1_h}$ | $H^1_h$ error reduction | $\eta$ |
|---|---|---|---|---|---|
| 1 | 5 | $2.784E-2$ | $4.744E-2$ | | |
| 2 | 10 | $7.379E-3$ | $1.464E-2$ | 3.24 | 1.70 |
| 4 | 20 | $2.054E-3$ | $5.054E-3$ | 2.90 | 1.53 |
| 8 | 40 | $5.617E-4$ | $1.777E-3$ | 2.84 | 1.51 |
| 16 | 80 | $1.520E-4$ | $6.281E-4$ | 2.83 | 1.50 |

Table 5: Cas 2 : erreur en fonction du raffinement.

### 4.4.3 Commentaires

Les résultats numériques des cas 1 et 2 (maillages réguliers mais non-raccordés) montrent clairement une tendance à un **ordre de convergence en $O(h^{3/2})$**. La super-convergence des volumes finis qui est due à la grille *structurée* est donc conservée malgré le non-raccord des maillages. On perd toutefois un facteur $1/2$ par rapport aux volumes finis classiques à cause de ce non-raccord. Cette propriété n'a pas encore de justification théorique mais semble vérifiée numériquement. On note de plus que les résultats restent sensiblement les mêmes pour les deux tests, bien que le maillage du sous-domaine interne soit 5 fois plus fin dans le cas 2 que dans le cas 1. L'algorithme semble donc peu dépendant du rapport entre les maillages des deux sous-domaines.

Les résultats sont moins nets pour le cas 3, qui correspond à des maillages qui ne sont pas emboîtés. En effet, contrairement aux cas précédents, le maillage du sous-domaine externe n'est

| $2^n$ | $1/h$ | $\|\|\cdot\|\|_\infty$ | $\|\|\cdot\|\|_{H_h^1}$ | $H_h^1$ error reduction | $\eta$ |
|---|---|---|---|---|---|
| 1 | 5 | $1.610E-2$ | $3.99E-2$ | | |
| 2 | 10 | $1.636E-2$ | $3.22E-2$ | 1.24 | 0.31 |
| 4 | 20 | $1.164E-2$ | $2.10E-2$ | 1.53 | 0.61 |
| 8 | 40 | $5.250E-3$ | $8.78E-3$ | 2.39 | 1.26 |
| 16 | 80 | $1.730E-3$ | $2.70E-3$ | 3.25 | 1.70 |
| 32 | 160 | $4.926E-4$ | $7.43E-4$ | 3.64 | 1.86 |
| 48 | 240 | $2.287E-4$ | $3.43E-4$ | 2.17 | 1.91 |

Table 6: Cas 3 : erreur en fonction du raffinement.

pas un sous-maillage du sous-domaine interne. L'ordre de convergence augmente et semble vouloir atteindre un $O(h^2)$. Cependant, le taux de convergence part d'un niveau bien inférieur aux cas précédents. Ainsi, si on compare au cas 1 qui est le pendant emboîté du cas 3, pour un raffinement égal de $2^n = 32$, on obtient dans un cas $2.10^{-4}$ contre $7.10^{-4}$ dans l'autre. L'erreur $H_h^1$ est donc ici $3.5$ fois plus forte dans le cas non emboîté que dans le cas emboîté.

## 4.5 Cas test ANDRA

On donne les résultats de la simulation du cas test statique de l'Andra défini à la section (1.5). Précisons d'abord la géométrie et le maillage des sous-domaines : voir figure (23).

### 4.5.1 Description

On résout le problème :

$$\begin{aligned} -\nabla\cdot(\nabla p) &= f \quad \text{dans } \Omega = [0,1100]\times[0,100], \\ p &= 0 \quad \text{sur } \partial\Omega. \end{aligned} \tag{33}$$

Le terme source vaut 1 dans les cellules du stockage (parties hachurées de taille $1m \times 1m$ sur la figure 23). On a effectué une mise à l'échelle car le problème est linéaire.

On décompose le domaine en 2 sous-domaines emboîtés comme le montre la figure (23).

Le critère d'arrêt pour l'algorithme est que la norme infinie du saut de la condition d'interface de Robin est inférieure à $10^{-8}$.

### 4.5.2 Résultats

On visualise les résultats sur les figures (24, 25 et 27). La première est un modèle simplifié avec 3 sources qui permet de mieux voir le comportement de la solution. Les autres figures prennent en compte les 50 sources et ont des raffinements différents.

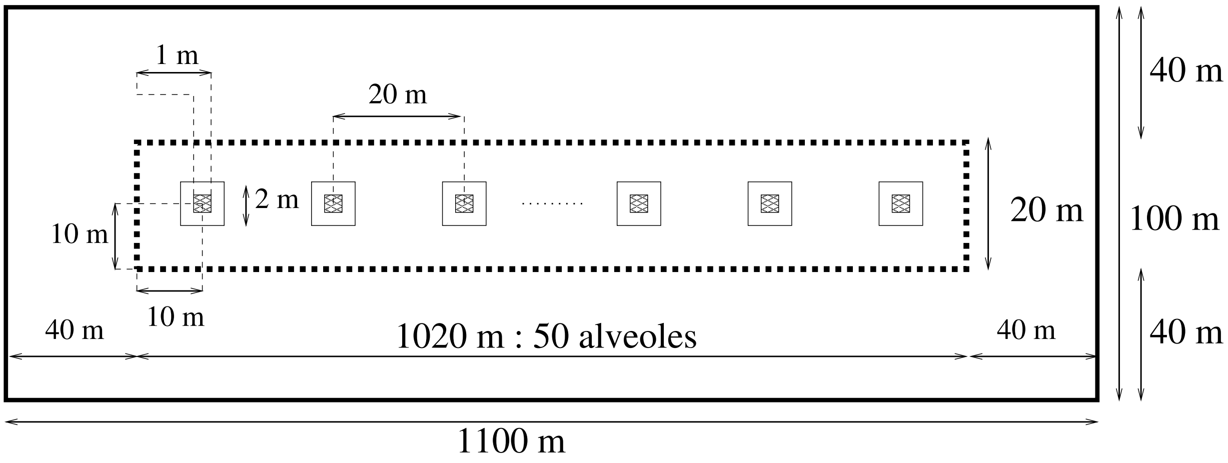


Figure 23: Géométrie du cas test Andra.

Les résultats avec deux raffinements au sein du stockage différents :

1. • Sous-domaine externe : $110 \times 10$ (mailles carrées de $10m \times 10m$).
   • Sous-domaine interne : $2040 \times 40$ (mailles carrées de $0.5m \times 0.5m$).

   Figure (25).

   BiCGStab converge en 34 itérations (moins d'une minute sur un PC à 512 Mo de RAM) pour 4368 inconnues sur l'interface et plus de $82\,000$ mailles.

2. • Sous-domaine externe : $110 \times 10$ (mailles carrées de $10m \times 10m$).
   • Sous-domaine interne : $4080 \times 80$ (mailles carrées de $0.25m \times 0.25m$).

   Figure (27) et (26).

   BiCGStab converge en 34 itérations (8 minutes sur un PC à 512 Mo de RAM) pour 8528 inconnues sur l'interface et plus de $163\,000$ mailles.

On note que le maillage externe est beaucoup moins fin que le maillage autour du site de stockage : *le maillage dans la couche géologique est 20 fois plus lâche que dans le voisinage des déchets*. Ceci n'a pas d'incidence sur la rapidité de convergence de la méthode.

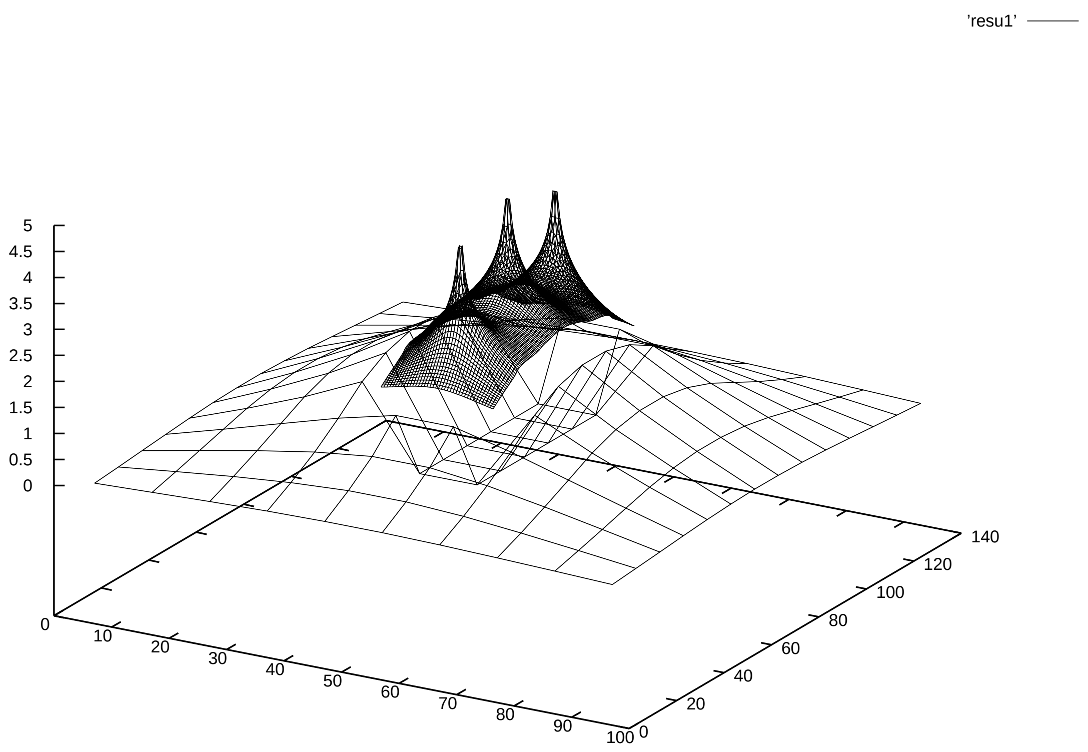


Figure 24: Solution du cas test Andra, 3 sources.

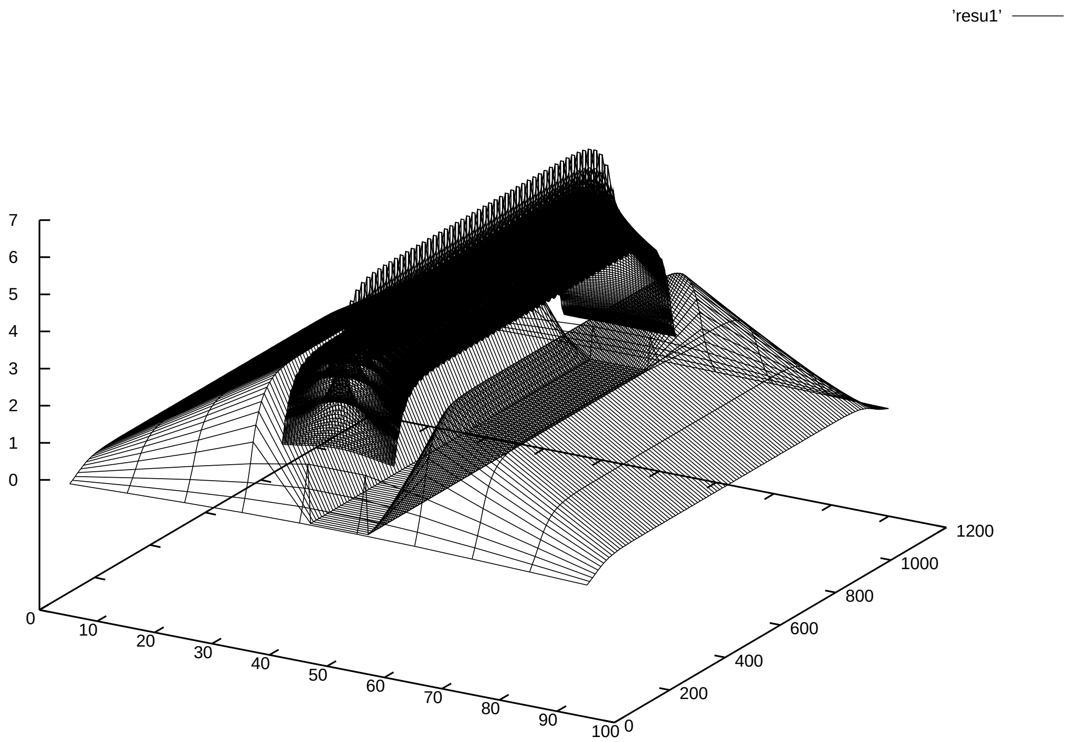


Figure 25: Solution du cas test Andra, 50 sources. Cas (110 x 10) et (2040 x 40) mailles.

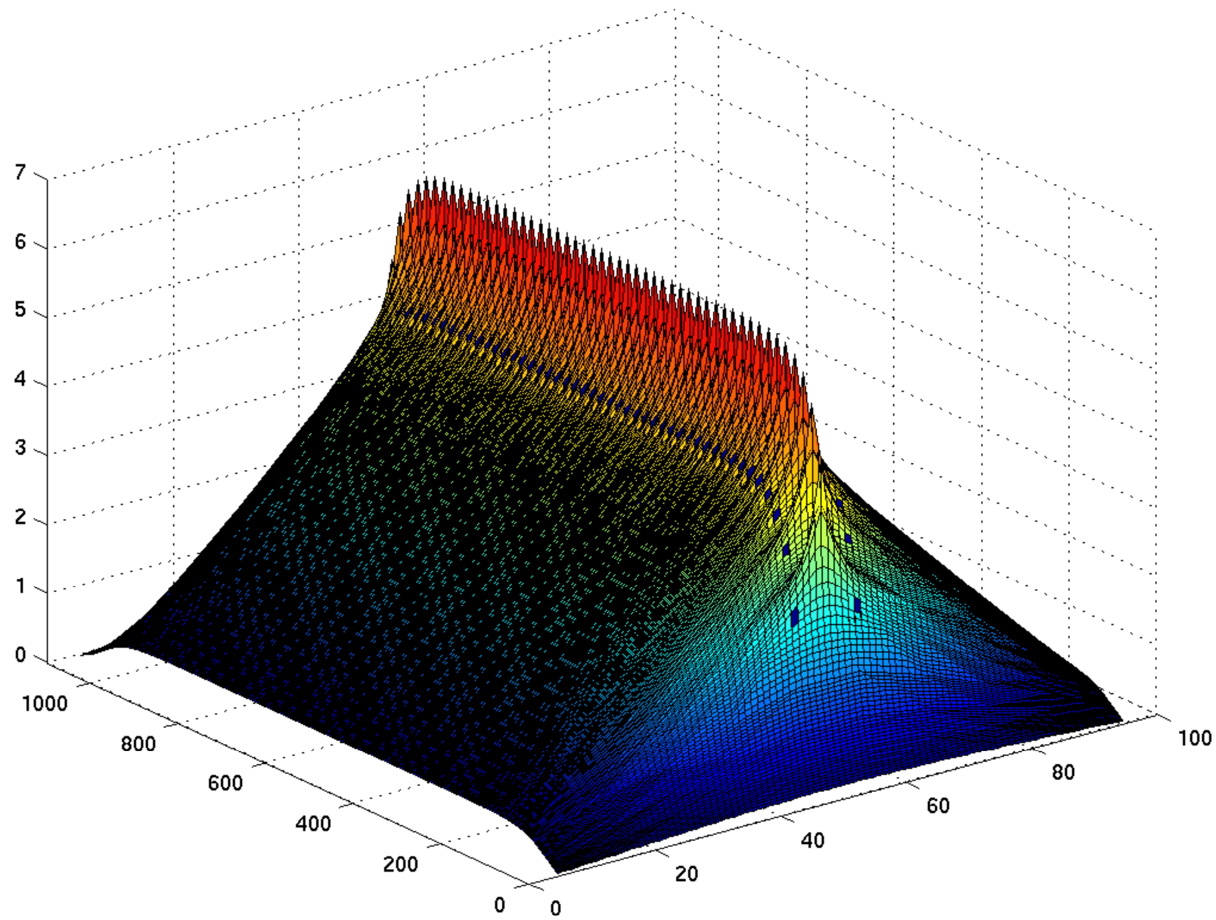


Figure 26: Solution interpolée du cas test Andra, 50 sources. Cas (110 x 10) et (2040 x 40) mailles.

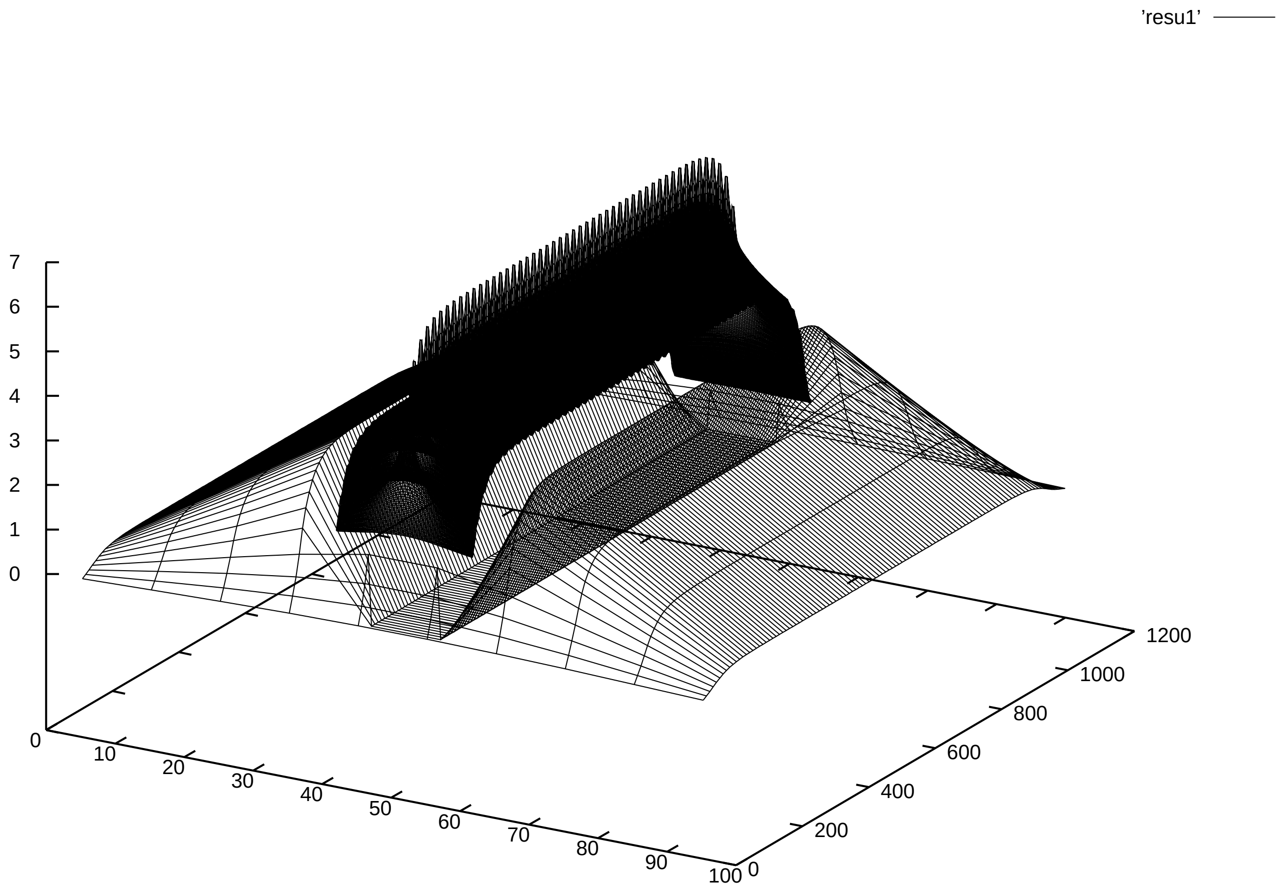


Figure 27: Solution du cas test Andra, 50 sources. Cas (110 x 10) et (4080 x 80) mailles.

# 5 Perspectives et conclusions

## 5.1 Conclusion

Les résultats obtenus montrent le bon comportement de l'algorithme étudié. La méthode générale de décomposition de domaines non-conforme semble donc validée pour l'etude du stockage profond des déchets. Elle paraît particulièrement adaptée au traitement du couplage entre le champ proche et le champ lointain. Cette méthode permet en effet de surmonter des difficultés liées aux différentes échelles spatiales du problème. D'autres études seraient utiles pour montrer sa pertinence en terme d'échelles temporelles ou encore en terme de couplages de différentes physiques.

## 5.2 Perspectives de développement

Les choix faits pour construire le modèle permettent d'obtenir lors de la procédure d'assemblage une matrice symétrique définie positive. Ceci permettra à terme de remplacer la méthode de décomposition LU, actuellement implémentée, par des méthodes de gradient conjugué, ou de Choleski par exemple.

Il faut également mettre en œuvre la boucle en temps et les termes de convection.

## 5.3 Remerciements

Tout d'abord, nous tenons à remercier Yves Achdou et Frédéric Nataf de leur aide et de leurs conseils tout au long du Cemracs 2001. Tant par des conseils de programmation, de structuration des codes, de méthodes de travail [6], nous leur sommes grandement reconnaissants.

Egalement, un grand merci à Stéphane Del Pino, grand manitou de Debian, pour ses nombreux coups de mains, conseils et débugages.

[6] **Il est d'ailleurs très utile de travailler à deux en même temps sur le même code mais surtout *sur le même fichier et sur le même écran*. C'est paradoxalement un gain de temps et d'energie!!**

# References


[1] Achdou Y., Japhet C., Maday Y. and Nataf F. *A New Cement to Glue Non-Conforming Grids with Robin Interface Conditions: the Finite Volume Case*. Internal report $n^o 419$, Ecole Polytechnique - Centre de Mathématiques Appliquées UMR CNRS 7641, May 1999. to appear in Numerische Mathematik.

[2] Eymard R., Gallouët T., Herbin R. *The Finite Volume Method*. Handbook of Numerical Analysis, Elsevier, Ph. Ciarlet J.L. Lions edition, 2000.

[3] H. A. V. D. Vorst. *Bi-CGStab: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*. SIAM J. Sci. Stat. Comput., 1992, vol. 13, pp. 631–644.