Ecole Nationale des Ponts et Chaussées

2004

Stage scientifique

Jérémie Simon

Aurélien Boffy

Une aide à l'apprentissage de la méthode des éléments finis

FEJavaDemo

Du 5 avril au 25 juin 2004

Tuteur : Doz. Dr. Michael Jung

Responsables du stage

Prof. Dr. Wolfgang Walter (TU Dresden)

M. Renaud Keriven (ENPC)

Remerciements

Nous tenons tout d'abord à remercier notre tuteur Doz. Dr. Michael Jung pour sa grande disponibilité, son aide et sa sympathie tout au long de notre stage, ainsi que pour son soutien lors de nos diverses prises d'initiatives.

Nous remercions également Professeur Walter, directeur de l'Institut de Calcul Scientifique, qui a tout fait pour que notre stage se passe pour le mieux.

Kshitij Kulshreshtha, collaborateur à l'Institut de Calcul Scientifique, nous a aidé en ce qui concerne la programmation de notre logiciel et nous lui en sommes vivement reconnaissant.

Nos remerciements vont aussi à toutes les personnes de l'Institut de Calcul Scientifique, et en particulier à Madame Gudrun Heinisch pour son affabilité.

En France, nous remercions les personnes qui ont permis et soutenu notre stage :

Monsieur Jean-Jacques Colleu et l'ensemble du Département de la Formation Alternée de l'ENPC.
Monsieur Renaud Keriven, responsable de ce stage.
La Fondation de l'Ecole Nationale des Ponts et Chaussées.

Résumé

Le stage consiste en la programmation en Java d'un logiciel (*FEJavaDemo*) permettant de résoudre des problèmes de conductivité thermique en utilisant la méthode des éléments finis. Le programme a pour but d'illustrer l'ouvrage *Methode der finiten Elemente für Ingenieure* dont notre tuteur Monsieur Michael Jung est co-auteur. Ce logiciel est destiné à l'apprentissage : l'utilisateur peut suivre les différents processus d'assemblage et de résolution grâce à des animations et à l'affichage des algorithmes implémentés. FEJavaDemo utilise des fonctions de forme linéaires ou quadratiques sur des triangles ou des quadrilatères mais a été conçu dans l'optique de gérer ultérieurement d'autres types d'objets. Pour rendre le programme utilisable par le plus grand nombre, il est disponible en trois langues et accompagné d'une documentation complète.

Mots clés : FEM, MEF, méthode des éléments finis, problème de conductivité thermique, Java, programme d'apprentissage, Cuthill McKee, calcul scientifique

Abstract

The purpose of the training was the development, in Java, of a software (FEJavaDemo) able to solve problems of thermal conductivity using the finite elements method. The aim of the program is to illustrate the book "Methode der finiten Elemente für Ingenieure" which was written by our tutor M. Jung. The software is a teach ware: the user can follow the different assembling and solving processes thanks to animations and the display showing the algorithms used by the program. FEJavaDemo uses linear or quadratic shape functions over triangles or quadrilaterals, but as been made in such a way that other object can be added to it in the future. To make it usable by as many people as possible, it is distributed with a complete documentation in three languages.

Keywords: FEM, Finite Element Method, heat conduction problem, Java, teach ware, Cuthill McKee, scientific computing

Table des matières

1. Présentation du stage

1.1. Historique de l'Université de Dresde

L'Université Technique de Dresde est l'une des plus anciennes Universités Techniques d'Allemagne.

Lorsque le centre de formation technique de Dresde, précurseur de l'université actuelle, fut créé en 1828, la révolution industrielle en Saxe n'en était qu'à ses débuts.
C'est grâce à cette institution éducative que la Saxe eut la possibilité de former très rapidement des mécaniciens et techniciens indispensables pour l'industrie du pays et par la même de promouvoir le développement industriel qui s'en suivit.

En 1871, elle obtint la reconnaissance en tant qu'Ecole Royale de Techniciens Supérieurs de la Saxe (Königlich-Sächsisches Polytechnikum), premier pas vers le statut d'université technique.
Déjà à l'époque, une filière générale avait été établie, regroupant l'économie nationale et les statistiques, l'allemand et la littérature ainsi que l'histoire des arts.

En 1865 commença la formation de professeurs de mathématiques, de science naturelle et de technologie. Les départements spécialisés de la construction mécanique, du génie civil et de chimie se profilèrent bientôt vers un axe plus scientifique et accréditèrent l'Ecole au delà les frontières de la Saxe.

Enfin en 1890, on lui décerna le statut d'Université Technique. La fin du XIX^{ème} siècle et les premières décennies du XX^{ème} apportèrent à l'Ecole ses orientations scientifiques déterminantes. De nouveaux instituts furent liés à l'activité de nombreux savants reconnus bien au-delà de l'Université de Dresde.

Le profil scientifique de l'Université Technique de Dresde n'a cessé d'accroître son caractère universel et universitaire. A côté d'un grand panel de disciplines d'ingénierie, les sciences

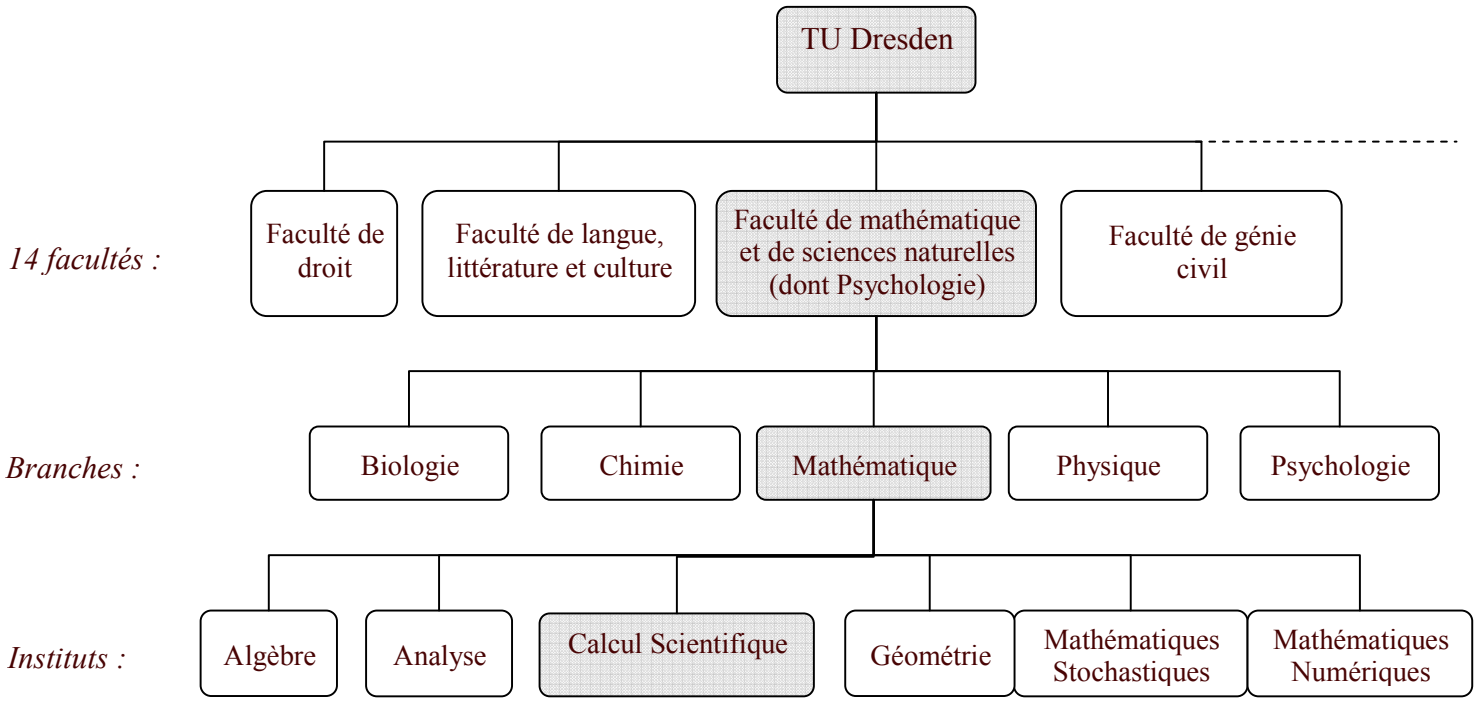
économiques, les lettres, la pédagogie et les sciences naturelles aussi font partie du potentiel scientifique de cette université. Cela a justifié en 1961 la nouvelle appellation d' « Université Technique de Dresde ».

Depuis la réunification allemande, de nouvelles facultés dans le domaine des lettres, des sciences sociales et des sciences économiques ainsi que la médecine furent ajoutées aux disciplines traditionnelles d'ingénierie et de sciences naturelles. Ainsi, l'Université Technique de Dresde dispose aujourd'hui d'un spectre scientifique en matière d'étude et de recherche, que l'on ne peut trouver, étant donné son envergure et sa diversité, que dans très peu d'universités allemandes.

Avec 30500 étudiants, environ 4500 collaborateurs (sans la faculté de médecine) et à peu près 480 professeurs, elle est aujourd'hui la plus grande université de Saxe. Depuis 1994, c'est une université complète avec 14 facultés. Son profil scientifique regroupe tous les domaines de l'ingénierie, des lettres et des sciences sociales, des sciences naturelles et de la médecine. A côté de ce large spectre, 16 cursus internationaux ont été aménagés.
La collaboration interdisciplinaire de toutes les branches constitue une des forces de l'université. Les rapports avec l'industrie sont multiples. L'Université Technique de Dresde encourage particulièrement les chercheurs de l'école, assure leur protection et se soucie du transfert rapide des inventions en produits commercialisables. Pour ses diplômés et tous les intéressés, l'Université propose des formations continues individuelles et adaptées à la pratique professionnelle.

1.2. L'Institut de calcul scientifique

Nous avons travaillé au sein de l'Institut de Calcul Scientifique, qui fait partie de la branche Mathématique de l'université. Cette section a été fondée en 1968 et est composée aujourd'hui de six instituts.



Les principaux axes de recherche de cet institut sont :

- Arithmétique d'ordinateur et langages de programmation scientifiques
- Méthodes multi-niveaux pour la simulation de processus physiques
- Calcul parallèle et outils logiciels
- Traitement d'images et vision

Le tuteur de notre stage s'appelle Michael Jung. Il est maître de conférences dans le domaine du calcul scientifique. Ses domaines de recherches sont, entre autres :

- Théorie de la convergence des méthodes multi-niveaux
- Assemblage et préconditionnement en utilisant des techniques multi-niveaux
- Combinaison de méthodes adapatives et multi-niveaux
- Implémentation de méthodes multi-niveaux en calcul parallèle
- Application de méthodes multi-niveaux à de problèmes pratiques
- Couplage de méthodes d'éléments finis avec des méthodes d'éléments de frontière

Ses spécialités sont les méthodes multi-niveaux et le calcul parallèle. Il donne des cours destinés aux ingénieurs concernant les éléments finis et les résolutions numériques d'équations différentielles partielles.

1.3. Le sujet de notre stage

1.3.1. Le livre de notre tuteur

Michael Jung est l'auteur, avec son collègue Ulrich Langer de l'Université Kepler de Linz en Autriche, d'un ouvrage relatif à la méthode des éléments finis : *Methode der finiten Elemente für Ingenieure. Eine Einführung in die numerischen Grundlagen und Computersimulation.* Ce manuel est fondé sur des cours traitant d'équations différentielles partielles que les auteurs ont donné à des élèves en formation d'ingénieur. Il constitue une introduction à la résolution numérique d'équations différentielles partielles et aux outils nécessaires issus des mathématiques numériques. Le cheminement permettant de passer du phénomène physique au modèle mathématique est décrit en s'appuyant sur l'exemple du problème de diffusion thermique. Les équations correspondant à ce problème servent de support à la description de la méthode des éléments finis.
Nous nous sommes particulièrement intéressés au chapitre 4 qui était consacré à la méthode des éléments finis appliquée à des problèmes elliptiques avec conditions aux limites dans des domaines à plusieurs dimensions. Ce chapitre propose tout d'abord les méthodes de Ritz et de Galerkin comme méthodes de discrétisation possibles, puis explique la méthode des éléments finis comme une méthode Ritz-Galerkin particulière. Les différentes étapes de cette méthode y

sont décrites avec précision à travers un exemple construit avec des fonctions linéaires sur des triangles.

Sur la page internet du livre cité se trouve une implémentation, librement téléchargeable, de la méthode des éléments finis : FEM2D. Ce programme illustre le livre car il utilise le même exemple que ce dernier et détaille les différentes étapes de la méthode. Il permet de résoudre des problèmes de conductivité thermique.

Ce logiciel est utilisé par les élèves, mais peut aussi aider les professeurs à préparer un cours sur les éléments finis car il permet de sauvegarder des images au format Jpeg de la matrice et du maillage.

Le but de notre stage était de reprogrammer ce logiciel car il comporte certains inconvénients :

- Il n'est pas modulable, c'est-à-dire qu'il est très difficile d'y ajouter un composant. Par exemple, le programme ne peut travailler qu'avec des fonctions linéaires sur des triangles et implémenter les fonctions de forme quadratiques reviendrait à revoir toute la logique du code.
- Il ne fonctionne que sous Windows.
- L'interface est grandement améliorable.
- Le langage de programmation (Fortran) est peu utilisé par rapport à C++ ou Java.

1.3.2. Le problème de conductivité thermique

On cherche le champ de température stationnaire $u(x)$ dans un domaine bidimensionnel Ω, qui peut être composé de plusieurs matériaux ayant des coefficients de conductivité $\Lambda(x)$ différents. Le domaine peut contenir des sources de chaleur qui sont décrites par une fonction f.

Trois types de conditions aux limites du domaine sont pris en compte :

- Les conditions de Dirichlet (frontière Γ_1) : La température $g_1(x)$ est fixée.
- Les conditions de Neumann (frontière Γ_2): Le flux de chaleur $g_2(x)$ à travers la frontière est fixé ($g_2(x)=0$ permet par exemple de simuler une isolation thermique)
- Les conditions de Fourier / Robin (frontière Γ_3): Modélisation d'un échange libre de chaleur avec l'extérieur (le flux de chaleur est proportionnel (coefficient $\alpha(x)$) à la différence entre la température sur la frontière $u(x)$ et la température extérieure $u_A(x)$)

Formulation classique :

On cherche $u \in C^2(\Omega) \cap C^1(\Omega \cup \Gamma_2 \cup \Gamma_3) \cap C(\bar{\Omega})$, tel que

$$
\begin{aligned}
\text{-div}(\Lambda(x)\ \text{grad}\ u(x)) &= f(x) && \forall x \in \Omega\ , \\
u(x) &= g_1(x) && \forall x \in \Gamma_1\ , \\
\frac{\partial u}{\partial N} &= g_2(x) && \forall x \in \Gamma_2\ , \\
\frac{\partial u}{\partial N} &= \alpha(x)\ \left[u_A(x) - u(x)\right] && \forall x \in \Gamma_3\ ,
\end{aligned}
$$

où $\Gamma = \partial\Omega = \bar{\Gamma}_1 \cup \bar{\Gamma}_2 \cup \bar{\Gamma}_3$ et $\Gamma_i \cap \Gamma_j = \emptyset$ pour $i \neq j$

et $\Lambda(x) = \begin{pmatrix} \lambda_1(x) & 0 \\ 0 & \lambda_2(x) \end{pmatrix}$ avec $\dfrac{\partial u}{\partial N} = \lambda_1(x)\dfrac{\partial u}{\partial x_1} n_1 + \lambda_2(x)\dfrac{\partial u}{\partial x_2} n_2$

et $\vec{n}(x) = \begin{pmatrix} n_1(x) \\ n_2(x) \end{pmatrix}$: vecteur unitaire sortant au point x $\in \Gamma$.

Formulation variationnelle du problème :

On cherche $u \in V_{g_1}$, tel que

$$a(u,v) = \langle F, v \rangle \qquad \forall v \in V_0$$

où

$$a(u,v) = \int_\Omega (\text{grad } v(x))^T \Lambda(x) \text{ grad } v(x)\, dx + \int_{\Gamma_3} \alpha(x) u(x) v(x) ds ,$$

$$\langle F, v \rangle = \int_\Omega f(x) v(x) dx + \int_{\Gamma_2} g_2(x) v(x) ds + \int_{\Gamma_3} \alpha(x) u_A(x) v(x) ds ,$$

$$V_{g_1} = \left\{ u \in H^1(\Omega) : u(x) = g_1(x) \text{ auf } \Gamma_1 \right\},$$

$$V_0 = \left\{ v \in H^1(\Omega) : v(x) = 0 \text{ auf } \Gamma_1 \right\}.$$

1.3.3. Les objectifs du stage

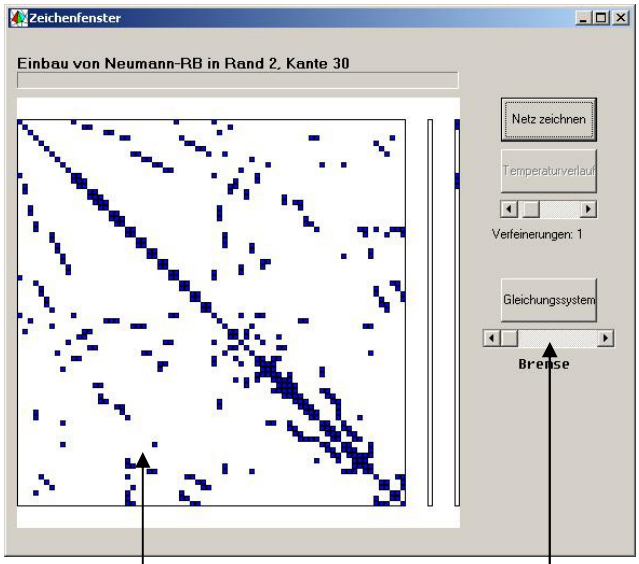
Le programme que nous devions réaliser avait pour but de traiter le problème de conductivité thermique. Il devait au moins travailler avec des fonctions de forme linéaires sur des triangles, ainsi qu'être en mesure de s'adapter, par la suite, à de nouveaux types de fonctions de forme et d'éléments.

Afin de résoudre un problème, l'ancien programme avait besoin de deux fichiers [Voir Annexe 2 – Structure des fichiers d'entrée] :

- Un fichier .net contenant les informations sur le maillage, c'est-à-dire les coordonnées des nœuds, les nœuds extrémités de chaque élément, etc.
- Un fichier .dat contenant les coefficients de conductivité thermique, les types de conditions aux limites sur les bordures, les valeurs des paramètres correspondants, ainsi que la fonction *f*.

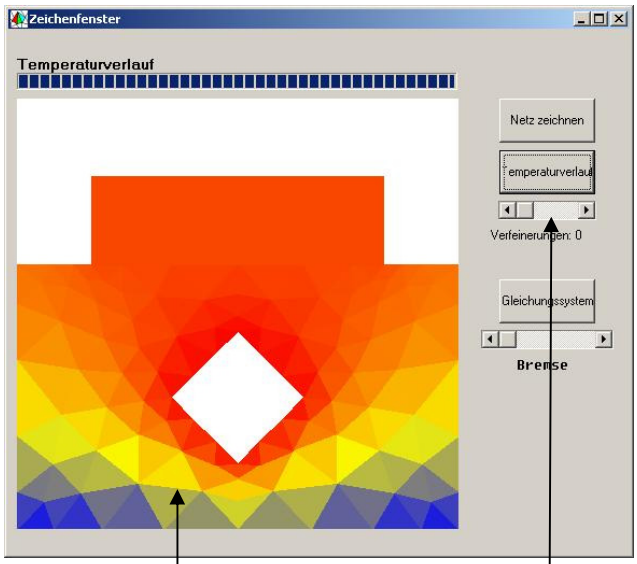
Le nouveau programme devait être compatible avec ces fichiers afin de pouvoir travailler avec les anciens exemples.

Captures d'écran de l'ancien programme :



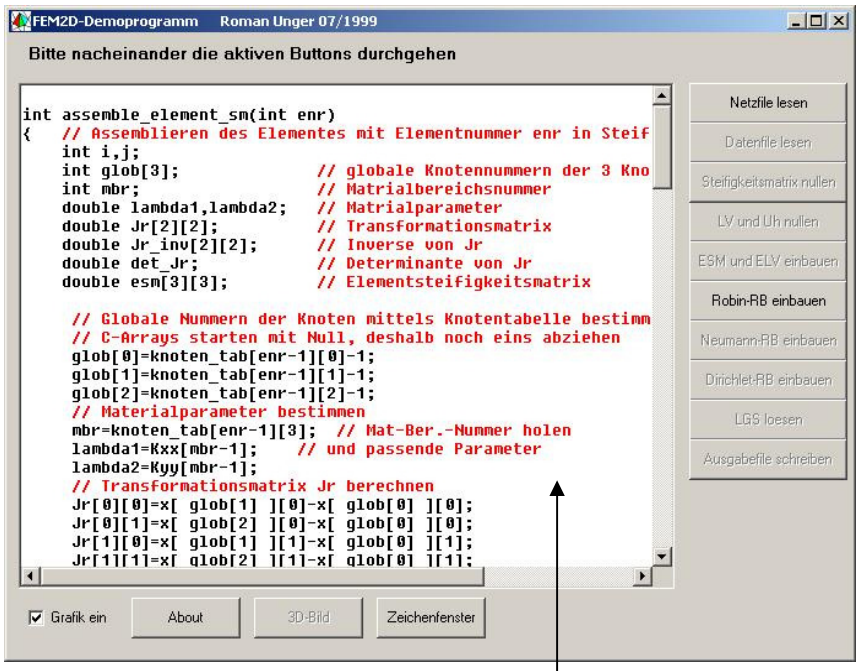
Matrice et vecteur second membre

Ascenseur pour ralentir l'affichage



Solution

Ascenseur pour raffiner le maillage (uniquement par des interpolations)



Boutons qui permettent de mettre en oeuvre les différentes étapes de la résolution

Algorithmes

Comme l'ancien logiciel, notre programme devait exposer les différentes étapes de la méthode des éléments finis. Dans une fenêtre sont représentés la matrice de rigidité et le vecteur second membre avec les éléments non nuls coloriés afin de souligner le fait que la matrice est creuse. Il est ensuite possible de mettre en œuvre les différentes étapes de l'assemblage une par une : dans un premier temps l'assemblage sans conditions aux limites, puis la prise en compte de ces conditions.

Dans une autre fenêtre, les algorithmes que le programme utilise à chaque étape sont affichés. Le maillage et la solution sont également représentés. Pour améliorer l'affichage de la solution, il est possible de diviser les triangles en quatre triangles plus petits et d'interpoler linéairement la solution sur ces nouveaux éléments.

Monsieur Michael Jung nous a également demandé de réaliser un manuel destiné à l'utilisateur ainsi qu'une documentation au cas où quelqu'un d'autre veuille plus tard ajouter des fonctionnalités au programme. Il souhaitait aussi que l'utilisateur ait la possibilité de créer des images Jpeg de la matrice et du maillage.

Nous avons décidé avec notre tuteur d'utiliser Java pour la réalisation du programme car ce langage permettait de palier à certains défauts de l'ancien logiciel. Java est en effet indépendant du système d'exploitation, librement disponible, facile à apprendre (il comporte peu de différences avec le C++ que nous connaissions déjà), utilisable sur internet (Java rend possible la réalisation d'applets, programmes dynamiques exécutés par le navigateur web) et possède une interface graphique (GUI) intégrée et simple à réaliser. De plus, Java est entièrement orienté objet ce qui permet de faciliter l'ajout de nouvelles fonctionnalités sans modifier tout le code.

2. Réalisation du travail

2.1. Quelques rappels sur la méthode des éléments finis

La méthode des éléments finis est une méthode numérique permettant de résoudre de façon approchée des systèmes d'équations différentielles partielles avec conditions aux limites.

Cette méthode consiste tout d'abord en un découpage du domaine Ω dans lequel on recherche la solution du problème, en un maillage de « petits » éléments $T^{(r)}$. On choisit comme fonctions de forme des fonctions polynomiales par morceaux qui sont nulles sur la plupart des éléments du maillage. Les solutions possibles de l'approximation sont établies grâce à une combinaison linéaire des n fonctions de forme.
En ce qui concerne la discrétisation d'un problème avec conditions aux limites dans un domaine à deux dimensions, on utilise en général des éléments triangulaires et quadrilatéraux. Ainsi on sélectionne souvent des fonctions de forme qui sont des fonctions linéaires, quadratiques ou cubiques.

Le point de départ est la formulation variationnelle du problème (voir 1.1)

On cherche $u \in V_{g_1}$: $a(u,v) = \langle F, v \rangle$ $\forall v \in V_0$.		(2.1)

Le principe de la méthode de Galerkin est de remplacer l'espace de dimension infinie dans lequel on cherche la fonction solution par un espace de dimension finie V_h :

$$V_h = \left\{ v_h \ : \ v_h = \sum_{i \in \bar{\omega}_h} v_i p_i(x) \right\} = span\{ p_i : i \in \bar{\omega}_h \} \quad \subset \quad V = H^1(\Omega)$$

où $\bar{\omega}_h$ est l'ensemble contenant les numéros de tous les nœuds du maillage.

Les fonctions p_i sont appelées « les fonctions de forme ». Elles sont linéairement indépendantes.

Comme la valeur de la solution aux points de Dirichlet est connue, on cherche une solution approchée du problème sous la forme :

$$u_h(x) = \sum_{j \in \omega_h} u_j p_j(x) \ + \sum_{j \in \gamma_h} u_{*,j} p_j(x)$$

où seuls les u_j sont inconnus.

ω_h est l'ensemble des numéros des nœuds de $\Omega \cup \Gamma_2 \cup \Gamma_3$ et γ_h l'ensemble des nœuds de $\overline{\Gamma}_1$.

($\overline{\omega}_h = \omega_h \cup \gamma_h$)

On obtient finalement le système suivant :

On cherche $\underline{u}_h = [u_j]_{j \in \omega_h} \in \mathbb{R}^{N_h}$:

$$\sum_{j \in \omega_h} u_j a(p_j, p_i) = \langle F, p_i \rangle - \sum_{j \in \gamma_h} u_{*,j} a(p_j, p_i) \qquad \forall i \in \omega_h$$

c'est-à-dire $\underline{u}_h = [u_j]_{j \in \omega_h} \in \mathbb{R}^{N_h}$ tel que :

$$K_h \underline{u}_h = \underline{f}_h$$

avec

$$K_h = [a(p_j, p_i)]_{i, j \in \omega_h}$$

$$\underline{f}_h = \left[\langle F, p_i \rangle - \sum_{j \in \gamma_h} u_{*,j} a(p_j, p_i) \right]_{i \in \omega_h} \in \mathbb{R}^{N_h}$$

N_h est le nombre de nœuds situés dans $\Omega \cup \Gamma_2 \cup \Gamma_3$

En ce qui concerne les données d'entrée, elles doivent dans notre cas satisfaire certaines contraintes :

- La fonction *f* peut avoir une définition différente pour chaque domaine et peut être définie grâce aux fonctions tan(), sin(), cos(), exp(), ln(), sqrt() et à la constante π. A titre d'exemple, la fonction f(x,y) = 2*cos(Pi*x*y) est reconnue par le programme.

- Λ est défini par λ_1 et λ_2. Ces coefficients sont des constantes qui peuvent être différentes pour chaque domaine.

- Concernant les conditions de Dirichlet, la température $g_1(x)$ n'est donnée qu'en un nombre fini de points. Ces points sont appelés les « points de Dirichlet ».

- Les autres fonctions $g_2(x)$, $\alpha(x)$ et $u_A(x)$ qui modélisent les conditions aux limites sont constantes sur chaque arête frontière. Elles sont ainsi constantes par morceaux.

2.2. Notre apprentissage de Java

Au début du stage, nous avions tous deux de solides bases en programmation, grâce en partie à l'enseignement d'informatique de première année à l'ENPC. Toutefois nous avons dû apprendre Java par la lecture d'ouvrages sur le sujet ([2], [3]).
Il nous est rapidement apparu qu'il y avait peu de différences entre Java et C++. Les deux sont par exemple orientés objets et la syntaxe est souvent comparable.

Voici les points qui ont particulièrement retenu notre attention :

- Pendant le cours de C++, nous n'avions pas abordé le concept d'héritage. L'héritage permet aux classes de transmettre leurs méthodes et leurs propriétés à leur descendance. Il était important pour notre programme de pouvoir avoir des classes parentes et des classes dérivées afin de le rendre modulaire. Par exemple, nous utilisons une classe abstraite « Elem » dont dérivent les classes « Triangle » et « Quadrilateral ».
- De même, nous ne connaissions rien du « multithreading ». Pour un programme avec une interface graphique, il est judicieux de scinder l'exécution en plusieurs processus (threads) dont le fonctionnement est indépendant. Ainsi un thread peut s'occuper d'actualiser l'écran pendant qu'un autre manipule des données. Certaines étapes du processus de résolution peuvent en effet être longues et il est important que l'utilisateur puisse interagir avec le programme pendant leur déroulement. Il est par exemple possible d'accélérer ou de ralentir le processus d'assemblage, ou encore d'interrompre un processus de raffinage trop coûteux en temps.
- Nous avions décidé de concevoir un programme multilingue, et nous avons pour cela testé plusieurs méthodes d'internationalisation. Celle que nous avons retenue permet d'ajouter de nouvelles langues de façon simple. Il suffit en effet de traduire un fichier texte qui contient les noms des boutons, des menus et les autres éléments textuels du programme.
- Afin de réaliser une interface graphique appropriée, nous avons étudié les composants « Swing » de Java. Ces composants (menus, barres de défilement, boutons, etc.) permettent

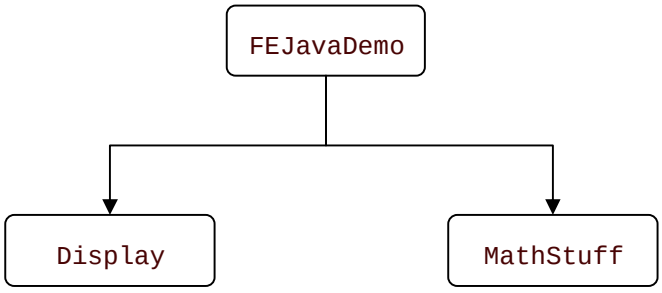
à l'utilisateur d'interagir avec le programme. Pour rendre le programme encore plus compatible, nous avons utilisé les « layout managers » qui nous ont permis d'agencer les composants sans spécifier leur position absolue. Nous avons ainsi réalisé une interface indépendante de la résolution d'écran.

- Afin rendre le programme encore plus robuste, nous nous sommes intéressés à la gestion des erreurs. Le programme devait en effet continuer à fonctionner en toutes circonstances. En Java, ceci est réalisé grâce aux exceptions. En ce qui concerne le programme, la principale source d'exceptions provient des fichiers d'entrée mais il peut aussi survenir des erreurs de mémoire lors d'opérations de raffinage avec de gros maillages.
- Comme demandé par notre tuteur, nous avons aussi réalisé une version « applet » de notre programme, c'est-à-dire une version utilisable sur internet. Ainsi il n'est plus nécessaire d'installer le logiciel sur la machine locale, l'exécution se faisant directement dans le navigateur, indépendamment du système d'exploitation de l'utilisateur. La plupart des fonctionnalités du programme est conservée dans l'applet.
- Le programme utilise plusieurs fois des fichiers : il lit des maillages et doit être capable d'écrire des fichiers .net et .dat, mais aussi de sauvegarder la solution du problème dans un fichier texte ou d'enregistrer un fichier Jpeg de la matrice. C'est pourquoi nous avons donc aussi étudié la manipulation des flux de données en Java.

2.3. Description et organisation des classes

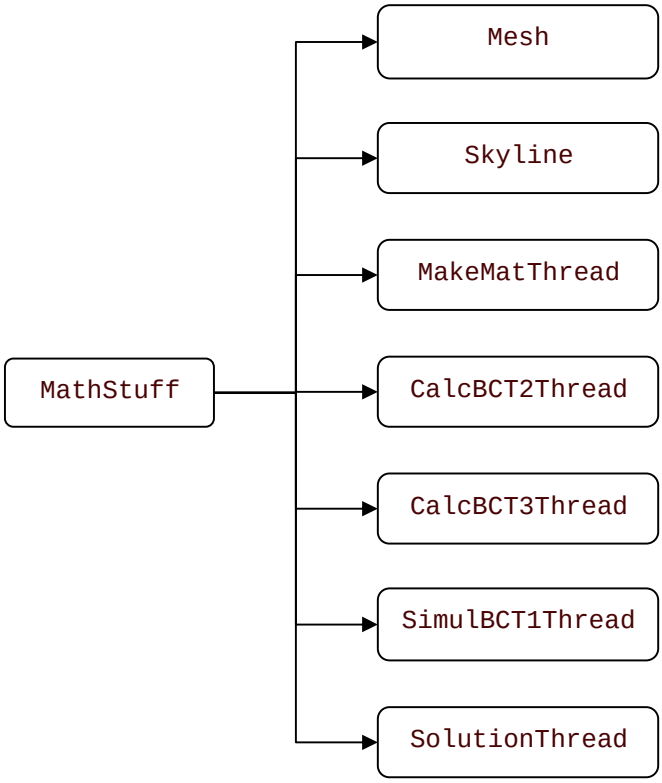
Le programme se compose d'environ 50 classes. Nous avons décidé de le décomposer en deux paquets (Package) : le premier contient tout ce qui est en rapport avec l'affichage et nous l'avons nommé « FEJDGui ». Le second, « FEJDMath », englobe toute la partie mathématique du programme. FEJavaDemo est une classe indépendante des deux paquets qui est utilisée pour démarrer le programme.
Nous allons maintenant décrire les différentes classes et détailler les liens qui les relient.

2.3.1. La classe « FEJavaDemo »



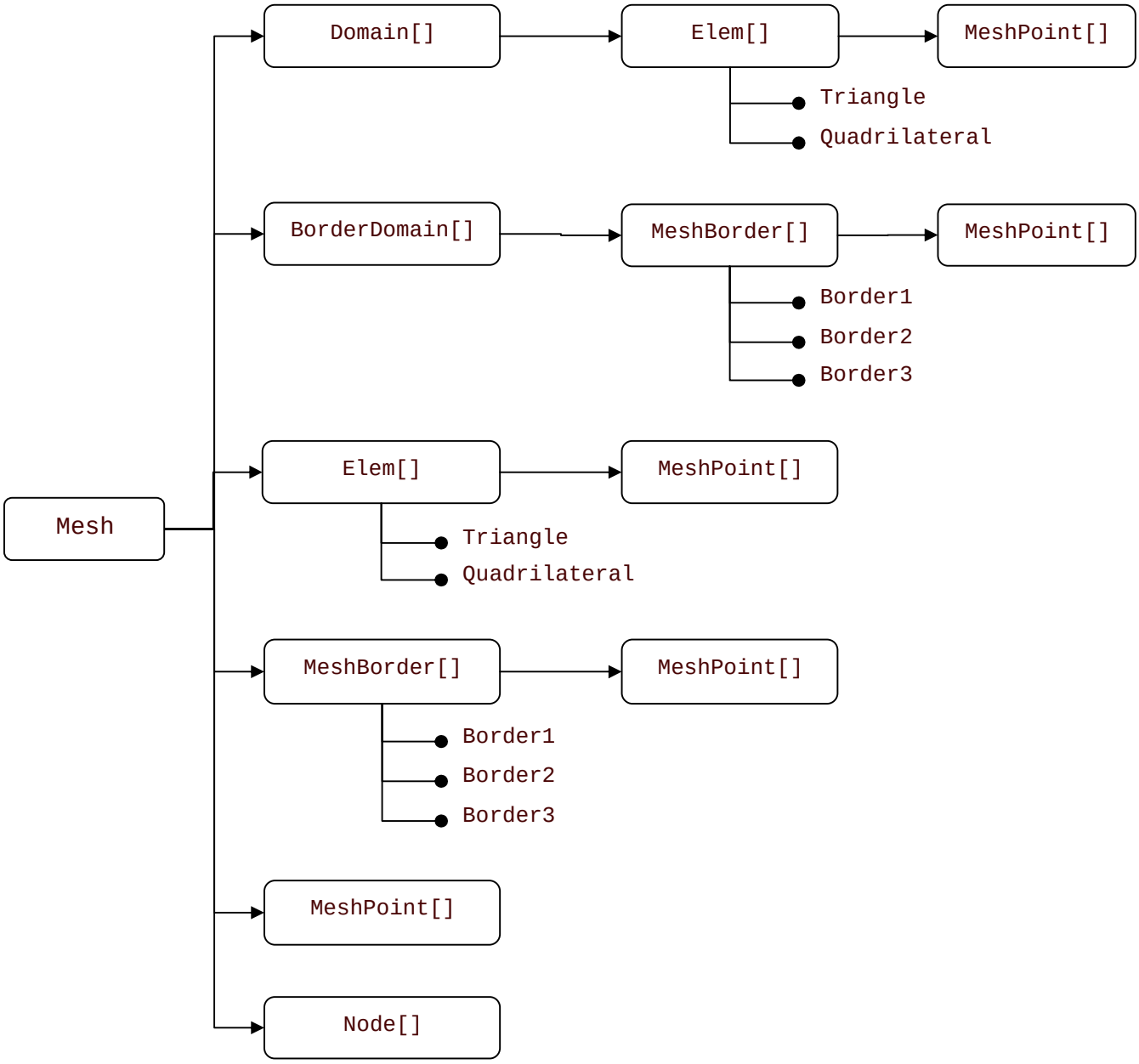
La classe FEJavaDemo initialise le programme en créant une instance des classes « MathStuff » et « Display ». Ces deux classes sont les classes principales du programme. Pour simplifier, il est possible de dire que Display et MathStuff représentent respectivement les paquets « FEJDGui » et « FEJDMath ».

2.3.2. La classe « MathStuff »



- La classe MathStuff contient une instance de la classe « Mesh » qui recèle toutes les informations sur le maillage (voir plus bas pour une description complète de cette classe).
- MathStuff contient aussi un objet « Skyline » qui permet d'enregistrer la matrice de rigidité (voir Renumérotation de nœuds [3.1]).
- On y trouve enfin cinq threads qui réalisent les différentes étapes du processus de résolution. Les algorithmes utilisés sont décrits plus bas.

2.3.3. La classe « Mesh »



La classe de base servant à décrire le maillage est « MeshPoint ».

- Elle représente un point avec ses deux coordonnées, sa température, son numéro global, etc.

  Nous avons différencié les nœuds (« Nodes ») des autres points du maillage (« MeshPoints »). Les nœuds ne sont que des extrémités des éléments alors que les « MeshPoints » représentent n'importe quel point où l'on calcule la température. Par exemple si le programme travaille avec des fonctions quadratiques, chaque élément triangulaire contient six MeshPoints et trois nœuds.
  Cette différenciation est nécessaire car il faut avoir accès à la liste des nœuds pour pouvoir par exemple dessiner le maillage, mais aussi à la liste des MeshPoints, par exemple pour assembler la matrice de rigidité.

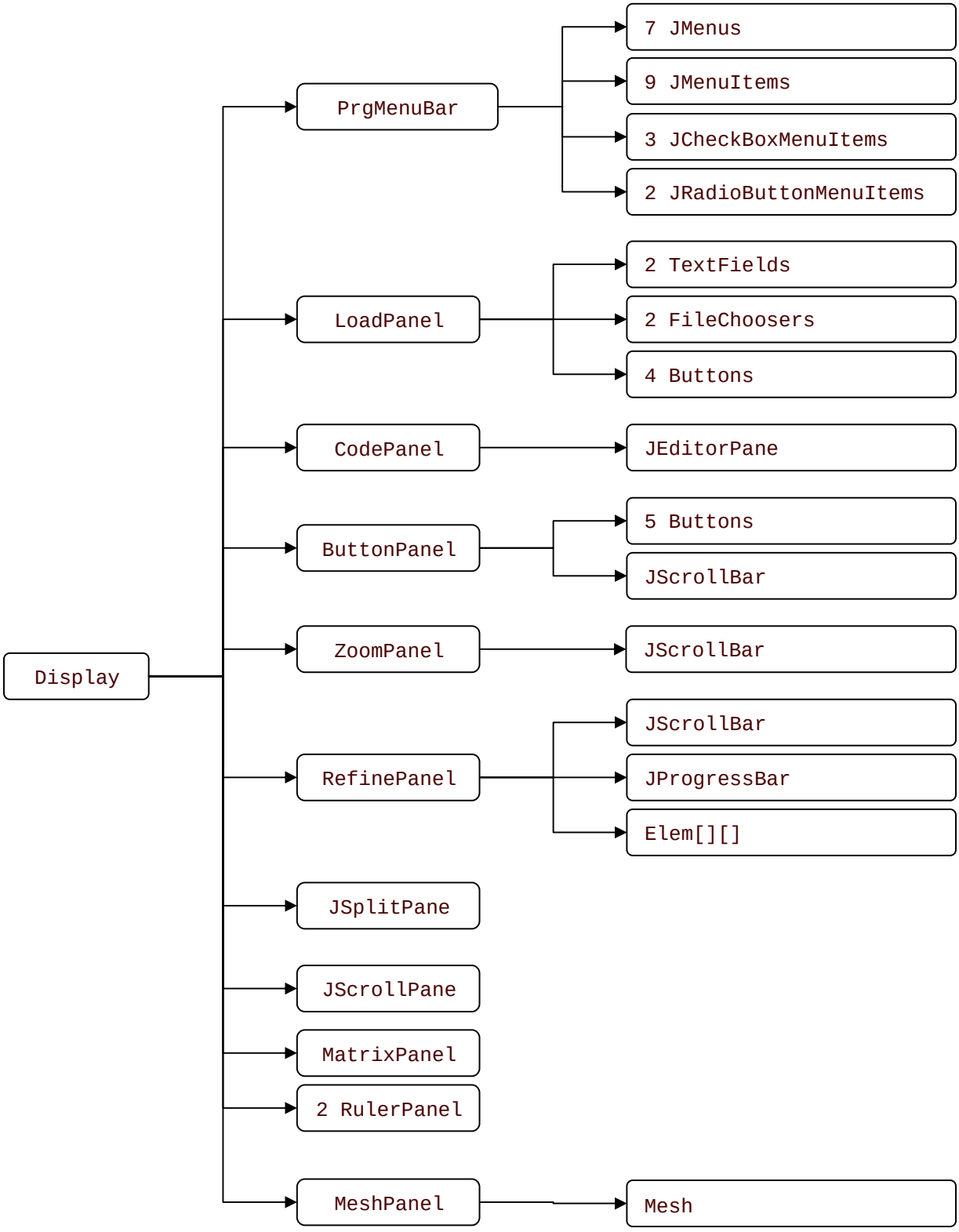
- Les éléments sont définis par un tableau de MeshPoints. « Elem » est une classe abstraite dont dérivent les classes « Triangle » et « Quadrilateral ». Chaque élément contient sa matrice de rigidité élémentaire et son vecteur second membre élémentaire, qui sont utilisés lors de l'assemblage.

- Nous avons créé une classe « Domain » qui représente le matériau. Chaque classe contient un tableau d'« Elem » et les coefficients de conductivité thermique qui correspondent au domaine.

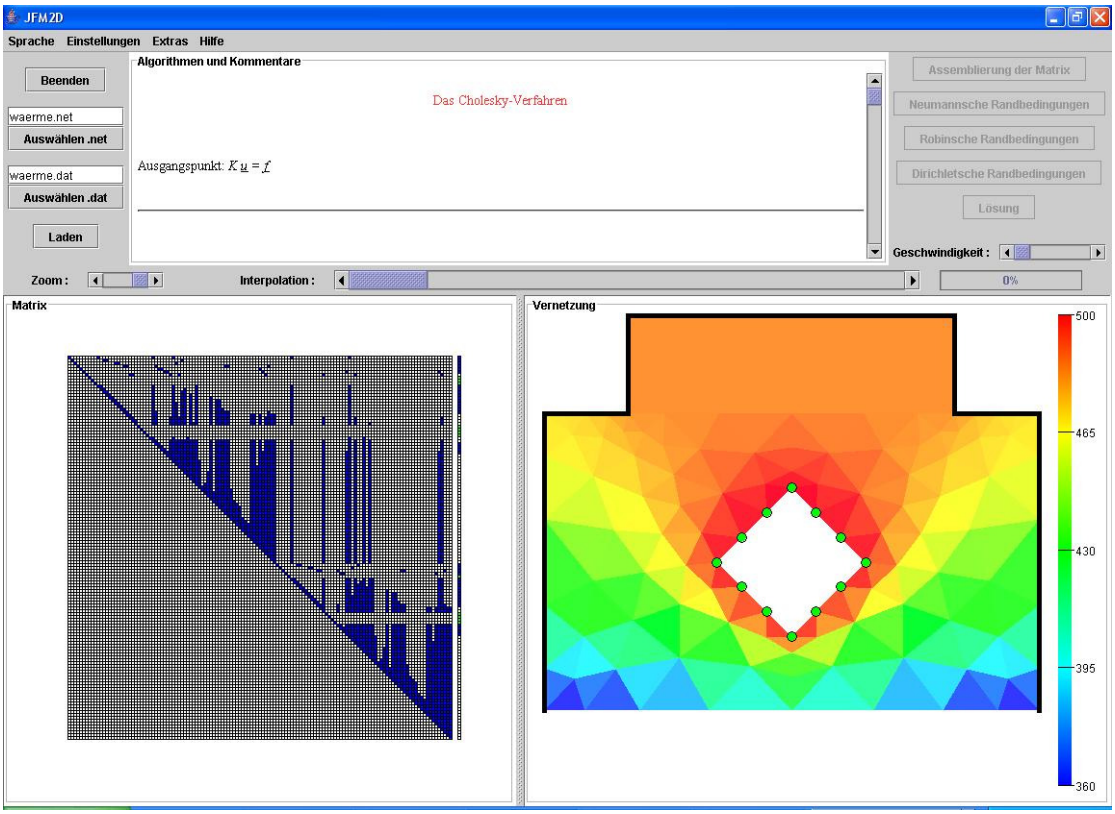
- Trois types de classes peuvent représenter les bordures : « Border1 », « Border2 » et « Border3 ». Chacune de ces classes contient un tableau de « MeshPoints » qui regroupe deux nœuds (les extrémités de la bordure) et éventuellement d'autres « MeshPoints », ainsi que les coefficients qui définissent les conditions aux limites. Pour le type « Border3 », par exemple, il y a deux valeurs : le coefficient de transfert thermique et la température de l'environnement.

- Les bordures sont regroupées en domaines frontières grâce à la classe « BorderDomain ». Une telle classe est nécessaire afin de pouvoir traiter les types de bordures les uns après les autres.

2.3.4. La classe « Display »



Display est la fenêtre du programme. Elle contient et initialise tous les composants de l'interface graphique :



- La barre de menu « PrgMenuBar » se compose de quatre menus. Le menu « langue » permet à l'utilisateur de traduire le programme en allemand, en français ou en anglais.
  Grâce au menu « Paramètres », il est possible de changer de type de fonctions de forme qui sera utilisé lors du chargement des fichiers .net et .dat. Ce menu permet aussi de renuméroter les nœuds et d'afficher la représentation des conditions aux limites ou de la matrice S^TS.
  L'option « Renumérotation automatique des nœuds avec des fonctions quadratiques » est cochée par défaut. Il est important d'utiliser cette fonctionnalité afin d'accélérer le processus de résolution. [Voir renumérotation des nœuds (3.1)].
  La dernière option de ce menu peut être utilisée afin de créer de nouveaux fichiers après une renumérotation ou un raffinage du maillage. Cela peut s'avérer utile par la suite pour charger un bon maillage directement sans avoir à répéter les mêmes étapes.
  Le menu « Accessoires » propose de créer des images Jpeg de la matrice et du maillage, de

sauvegarder la solution (les nœuds avec leurs coordonnées et leur température) dans un fichier texte, de renuméroter les nœuds et de raffiner le maillage. Pour plus d'informations sur ces deux dernières possibilités, voir le chapitre 3.

- Lors du démarrage du programme, l'utilisateur doit charger deux fichiers qui décrivent, entre autres, le maillage [Voir Annexe 2 – Structure des fichiers d'entrée]. Le « LoadPanel » offre la possibilité de choisir ces deux fichiers sur le disque. Il s'y trouve aussi un bouton qui permet de mettre fin à l'exécution de FEJavaDemo.

- Le « CodePanel » affiche les algorithmes que le programme utilise pour résoudre le problème. L'utilisateur peut ainsi découvrir ces méthodes en détail sans avoir à lire le code Java. Ceci permet également de comprendre les différentes animations grâce à ces liens avec la théorie (on peut ainsi par exemple comprendre pourquoi certains éléments sont coloriés).

- Le « ButtonPanel » est utilisé pour exécuter les différentes étapes de la résolution du problème. Un bouton lance l'assemblage de la matrice de rigidité et trois autres permettent de prendre en compte les conditions aux limites. Le dernier bouton démarre le processus de calcul de la solution. Voir plus bas pour de plus amples détails sur les algorithmes utilisés. Une barre de défilement change la vitesse de l'animation afin que l'utilisateur puisse suivre le fonctionnement du programme. L'utilisateur préfère souvent que le programme livre la solution le plus rapidement possible, c'est pourquoi les étapes de l'assemblage ne sont plus affichées lorsque le curseur est placé à l'extrême gauche.

- Le « MatrixPanel » est l'un des composants les plus importants de FEJavaDemo, puisqu'il affiche la matrice de rigidité globale et le vecteur second membre. Comme la matrice est souvent trop grande pour l'interface, ce composant contient une « Scrollpane » qui sert à ajouter deux barres de défilement si nécessaire. Deux marges indiquent les indices des lignes et de colonnes. Les éléments non nuls de la matrice sont en bleu afin de montrer que cette dernière est très creuse. Lors des différents processus d'assemblage, les cases de la matrice sur lesquelles travaille le programme sont coloriées en rouge. Les cases vertes correspondent aux nœuds de Dirichlet. En cochant la propriété adéquate, on peut observer la matrice diagonale supérieure S de la factorisation S^TS. Une petite animation symbolise les processus de descente et de remontée de l'algorithme de Cholesky.

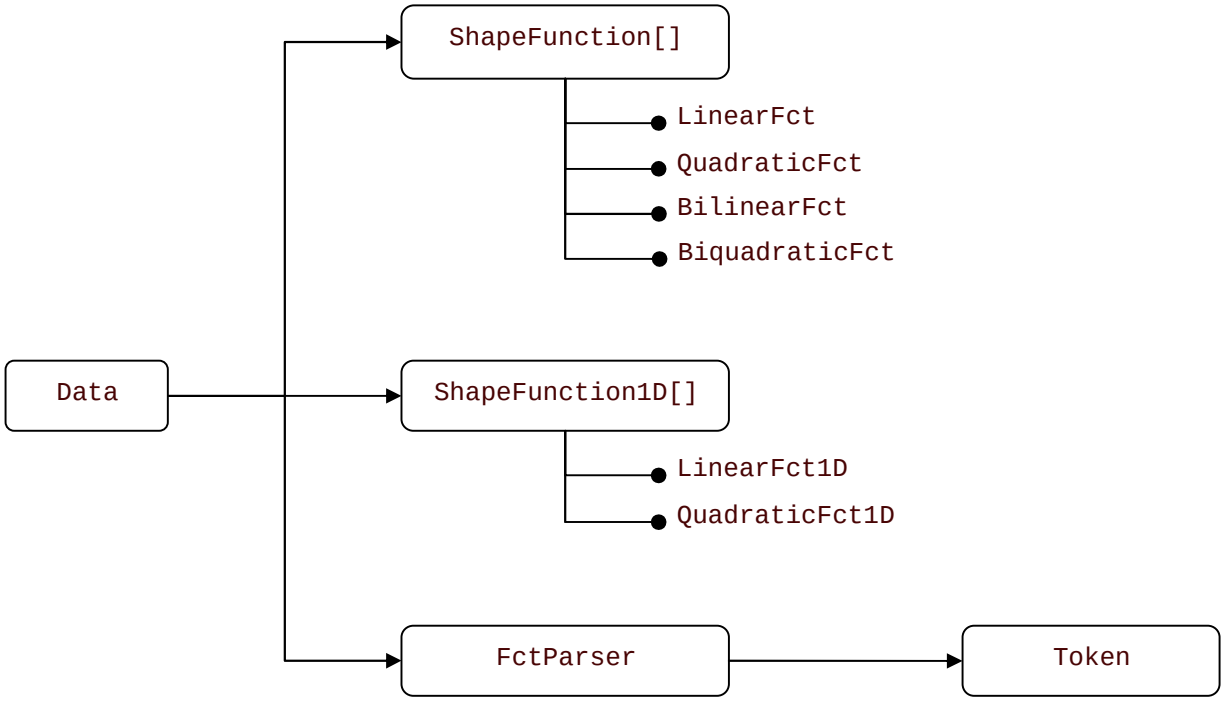
- Le « ZoomPanel » contient un ascenseur qui permet la modification du zoom du MatrixPanel. Lorsque le curseur est placé à l'extrême gauche, les valeurs des cases de la matrice sont affichées.

- Le maillage est affiché sur le « MeshPanel ». Lorsque la résolution est terminée, le champ de température est représenté, muni d'une échelle de température. Comme pour le MatrixPanel, les éléments avec lesquels travaille le programme sont coloriés en rouge. Il est alors possible de comparer les deux panneaux afin de comprendre le principe de l'assemblage. De plus, grâce au menu « Paramètres », les numéros de nœuds ainsi qu'une représentation des conditions aux limites peuvent être affichés. Cette représentation colorie en vert les nœuds de Dirichlet et surligne les bordures qui isolent thermiquement le matériau.

- Le « RefinePanel » permet d'améliorer la représentation de la solution. Les éléments peuvent être divisés et la solution est alors interpolée linéairement sur les nouveaux éléments. Comme ceci peut être assez long, une barre d'avancement permet à l'utilisateur de connaître le pourcentage du processus qui a déjà été exécuté.

- Nous avons utilisé un « SplitPanel » qui offre la possibilité de placer deux composants côte à côte tout en permettant à l'utilisateur de décider quelle proportion de l'écran il souhaite allouer à chaque fenêtre. Il était en effet intéressant de pouvoir afficher la matrice ou le maillage dans un espace plus grand.

2.3.5. La classe « Data »

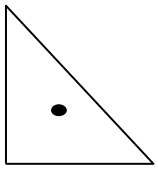
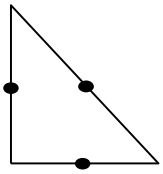
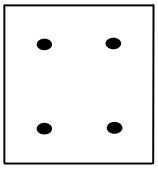
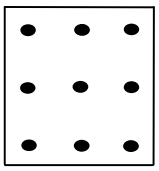


« Data » est une classe indépendante qui est utilisée pour calculer les matrices de rigidité et les vecteurs second membre élémentaires. Elle contient les coefficients des fonctions de forme et les méthodes de calcul des intégrales.

Lors du calcul des éléments des matrices et des vecteurs, il n'est pas toujours possible de calculer les intégrales de façon analytique. C'est pourquoi le programme utilise des formules de quadratures dont le principe est d'approcher l'intégrale par une somme. FEJavaDemo utilise des quadratures de la forme :

$$\int_{\hat{T}} w(\xi) d\xi \approx \sum_{i=1}^{l} \alpha_i w(\xi_i)$$

Comme les polynômes que le programme doit intégrer sont toujours d'un degré relativement petit, FEJavaDemo utilise assez de points de quadrature pour calculer en fait toujours les intégrales de façon exacte :

Position des points de quadrature	Coordonnées ξ_i ou $(\xi_{i,1}, \xi_{i,2})$ des points de quadrature	Poids α_i	exact pour polynômes de degré k
	$\frac{3-\sqrt{3}}{6}, \frac{3+\sqrt{3}}{6}$	$\frac{1}{2}, \frac{1}{2}$	k = 3
	$\frac{5-\sqrt{15}}{10}, \frac{1}{2}, \frac{5+\sqrt{15}}{10}$	$\frac{5}{18}, \frac{8}{18}, \frac{5}{18}$	k = 5
	$\left(\frac{1}{3}, \frac{1}{3}\right)$	$\frac{1}{2}$	k = 1
	$\left(\frac{1}{2}, 0\right), \left(\frac{1}{2}, \frac{1}{2}\right), \left(0, \frac{1}{2}\right)$	$\frac{1}{6}, \frac{1}{6}, \frac{1}{6}$	k = 2
	$\left(\frac{3-\sqrt{3}}{6}, \frac{3+\sqrt{3}}{6}\right), \left(\frac{3+\sqrt{3}}{6}, \frac{3+\sqrt{3}}{6}\right)$ $\left(\frac{3-\sqrt{3}}{6}, \frac{3-\sqrt{3}}{6}\right), \left(\frac{3+\sqrt{3}}{6}, \frac{3-\sqrt{3}}{6}\right)$	$\frac{1}{4}, \frac{1}{4}$ $\frac{1}{4}, \frac{1}{4}$	k = 3
	$\left(\frac{5-\sqrt{15}}{10}, \frac{5+\sqrt{15}}{10}\right)\left(\frac{1}{2}, \frac{5+\sqrt{15}}{10}\right)\left(\frac{5+\sqrt{15}}{10}, \frac{5+\sqrt{15}}{10}\right)$ $\left(\frac{5-\sqrt{15}}{10}, \frac{1}{2}\right)\left(\frac{1}{2}, \frac{1}{2}\right)\left(\frac{5+\sqrt{15}}{10}, \frac{1}{2}\right)$ $\left(\frac{5-\sqrt{15}}{10}, \frac{5-\sqrt{15}}{10}\right)\left(\frac{1}{2}, \frac{5-\sqrt{15}}{10}\right)\left(\frac{5+\sqrt{15}}{10}, \frac{5-\sqrt{15}}{10}\right)$	$\frac{25}{324}, \frac{10}{81}, \frac{25}{324}$ $\frac{10}{81}, \frac{16}{81}, \frac{10}{81}$ $\frac{25}{324}, \frac{10}{81}, \frac{25}{324}$	k = 5

Formules de quadrature sur l'intervalle de référence I = [0,1] et sur l'élément de référence

Remarque : Sur les quadrilatères, les formules sont exactes pour des fonctions qui sont des polynômes de degré *k* dans chaque direction.

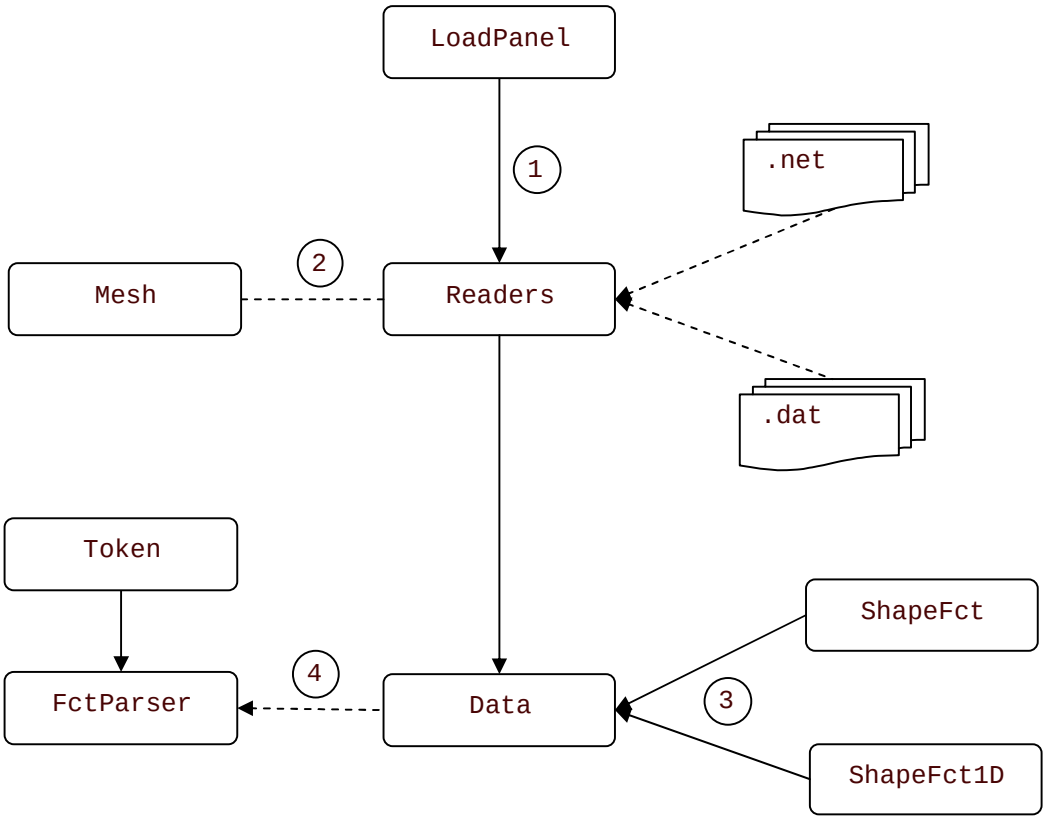
L’intégration numérique est utilisée plusieurs fois :

- Lors du calcul des matrices de rigidité et des vecteurs second membre élémentaires : Le programme procède à un changement de variables afin de n’avoir à calculer des intégrales que sur l’élément de référence (triangle, carré, etc.)
- Lors de la prise en compte des conditions aux limites : De même, FEJavaDemo ne calcule les intégrales que sur l’intervalle de référence I = [0,1].

2.4. Processus

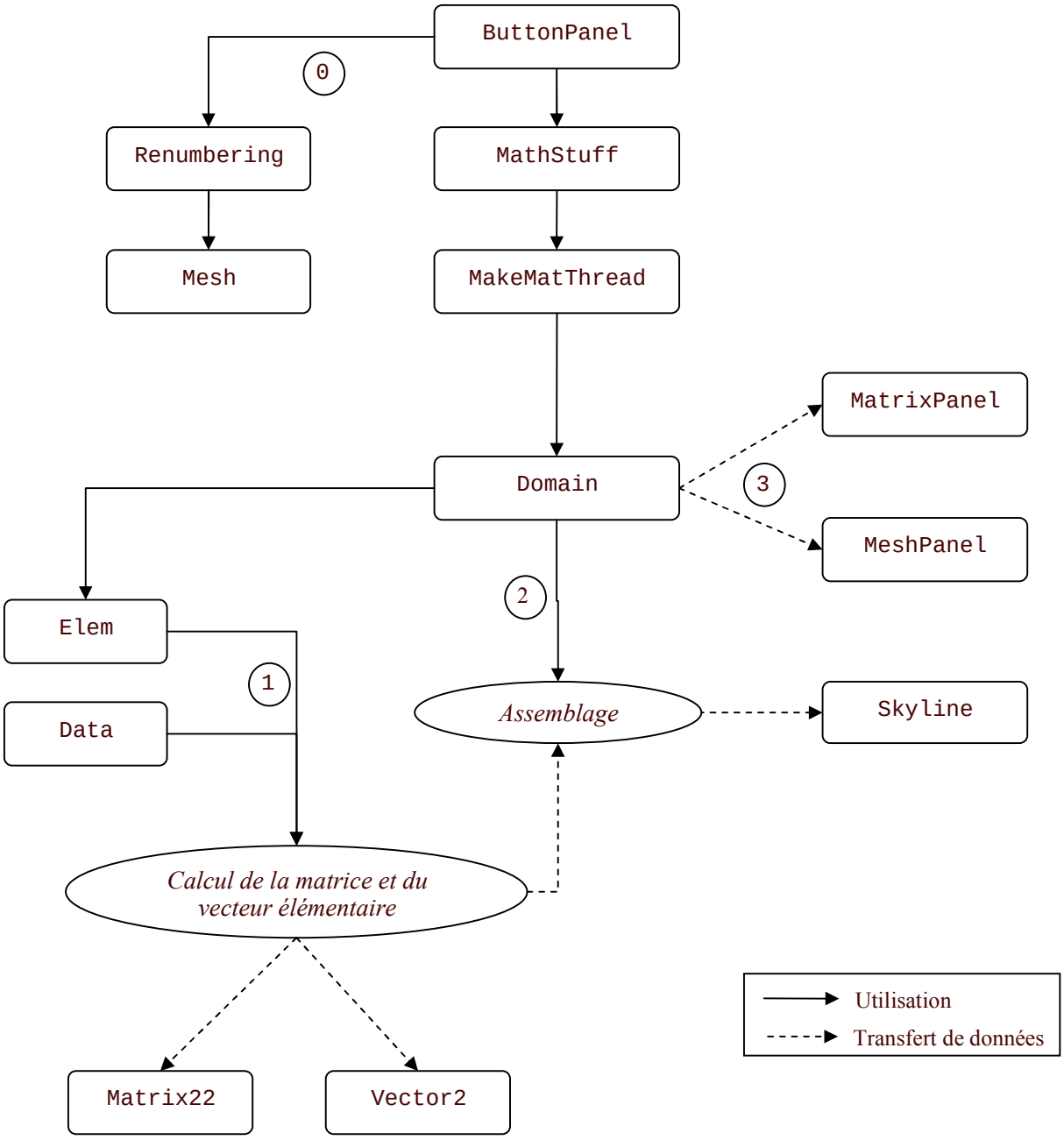
Ce chapitre détaille les interactions entre les différentes classes. Des diagrammes décrivent la succession des étapes et les classes impliquées dans chaque processus. Les algorithmes représentés par des ovales seront précisés au chapitre suivant.

2.4.1. Chargement des données



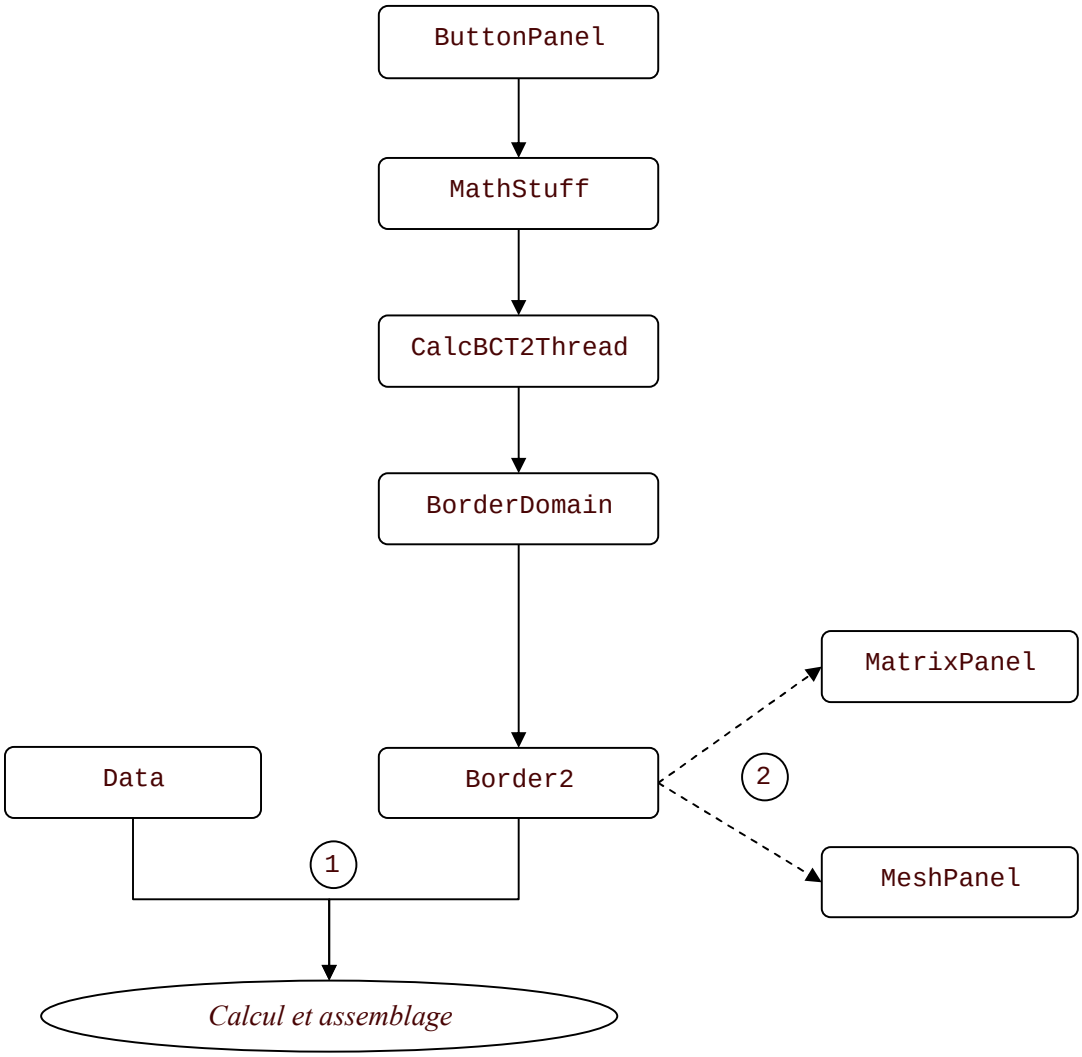
1) Le « LoadPanel » utilise un objet « Reader » pour lire les fichiers de données.
2) « Readers » remplit l'objet « Mesh » avec les informations contenues dans ces fichiers.
3) « Readers » appelle l'objet « Data », afin qu'il remplisse le tableau des valeurs des fonctions de forme en chaque point de quadrature grâce à « ShapeFct » et à « ShapeFct1D ».
4) « Data » fournit la fonction f à « FctParser » en tant qu'objet « String ». L'objet « FctParser » la découpe alors en objets « Token » compréhensibles par le programme.

2.4.2. Assemblage de la matrice



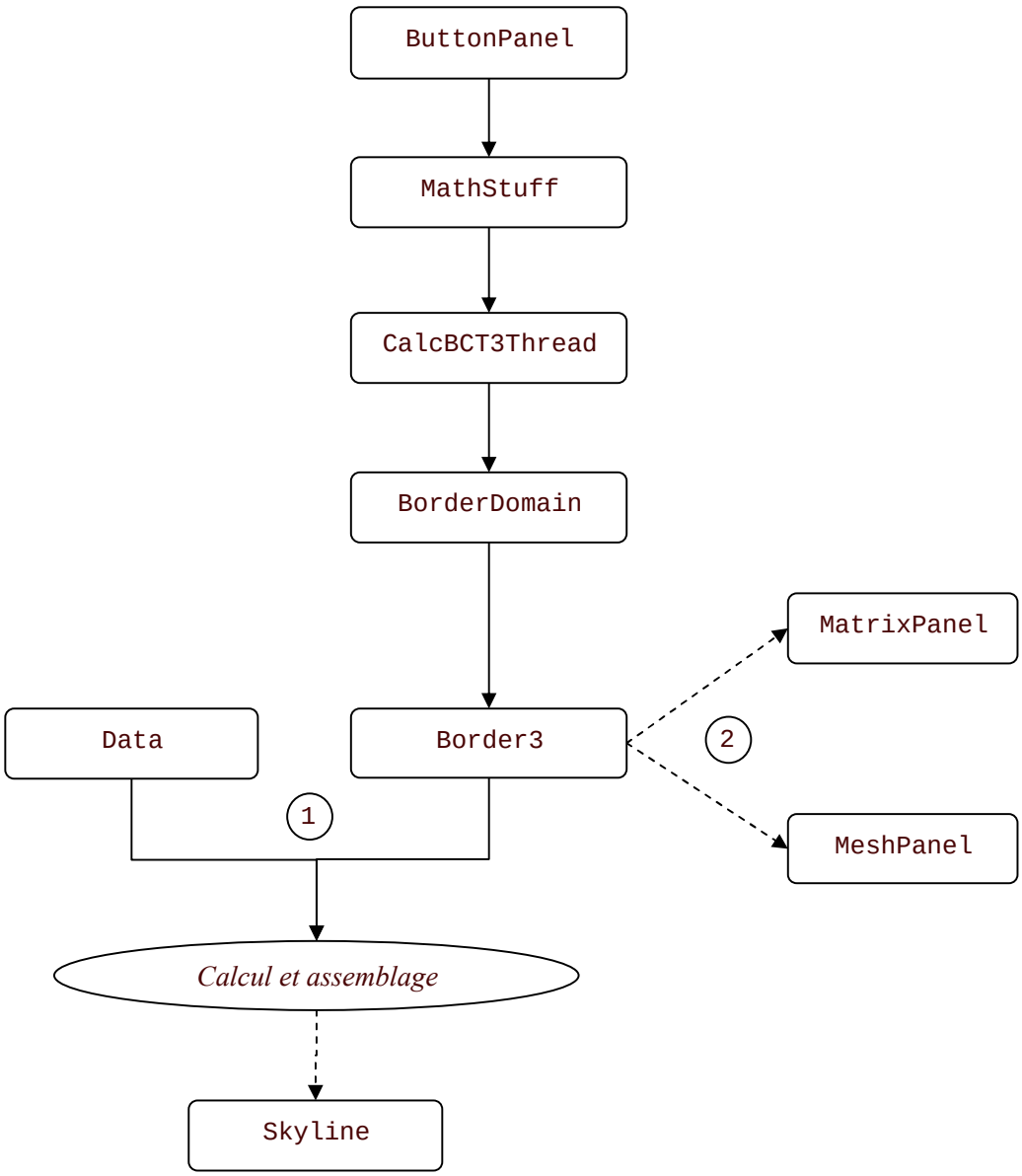
1) Si besoin est (avec des fonctions de forme quadratiques), le maillage « Mesh » est renuméroté.
2) Une boucle sur les domaines est nécessaire pour prendre en compte les différents coefficients de conductivité thermique. Le programme calcule au niveau des éléments les matrices de rigidité et les vecteurs second membre qui seront stockés dans les « Matrix22 » et les « Vector22 ». Les intégrales sont calculées grâce aux objets « Data ».
3) Après le calcul d'une matrice et d'un vecteur élémentaires, l'objet « Domain » demande l'assemblage et remplit la matrice de rigidité globale qui est stockée dans un objet « Skyline ».
4) L'objet « Domain » indique au « MeshPanel » et au « MatrixPanel » quel est l'élément en cours de traitement, afin qu'il puisse être colorié en rouge.

2.4.3. Prise en compte des conditions aux limites de type 2



1) A l'intérieur d'une boucle sur les domaines-frontières qui contiennent des bords de type 2 (il n'y a généralement qu'un seul domaine-frontière de la sorte), le programme assemble les vecteurs élémentaires au niveau de chaque bord.
2) L'objet « Border2 » indique au « MeshPanel » et au « MatrixPanel » quel est le bord en cours de traitement, afin qu'il puisse être colorié en rouge.

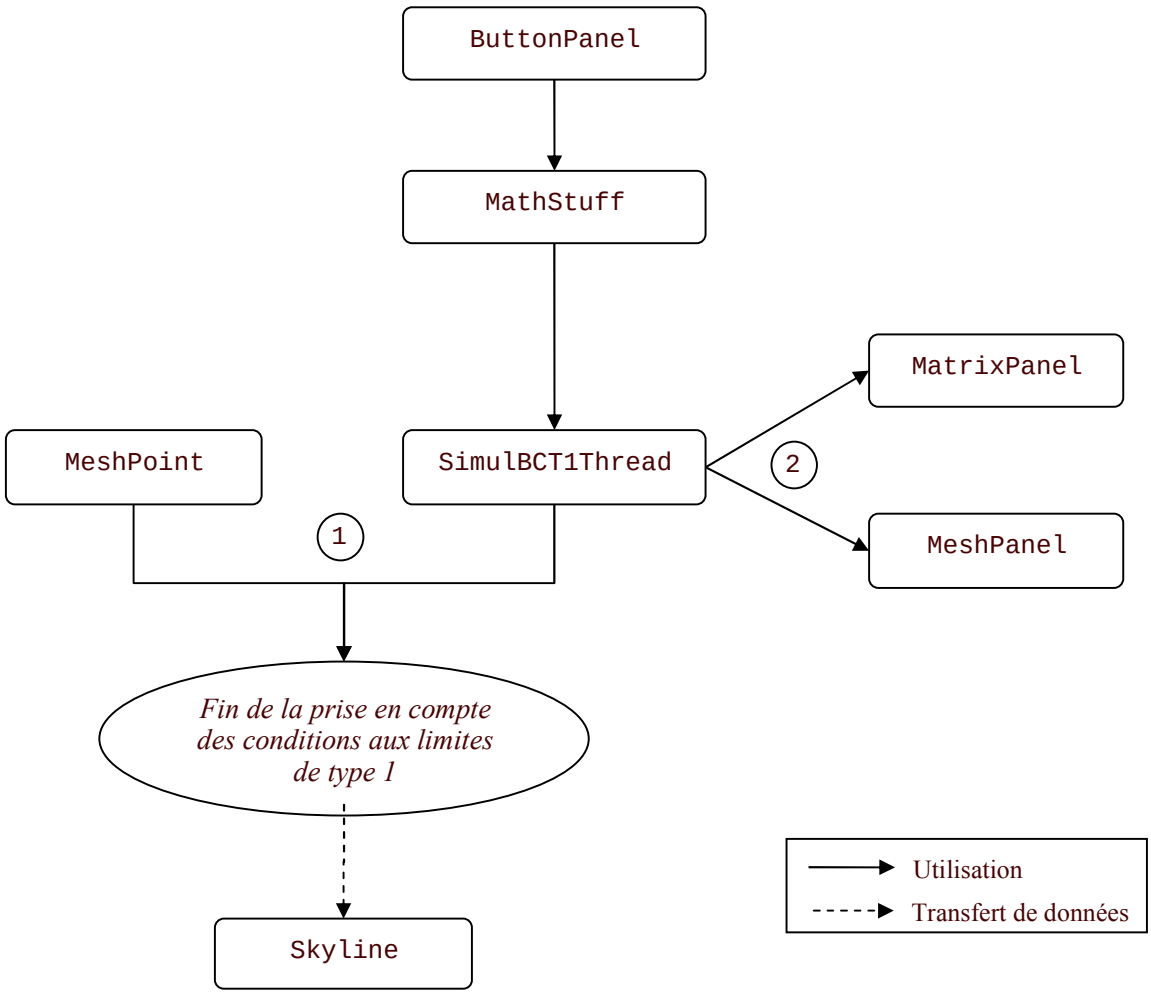
2.4.4. Prise en compte des conditions aux limites de type 3



1) A l'intérieur d'une boucle sur les domaines-frontières qui contiennent des bords de type 3 (il n'y en a généralement qu'un), le programme assemble les vecteurs et les matrices élémentaires au niveau de chaque bord, ce qui modifie la Skyline.

2) L'objet « Border3 » indique au « MeshPanel » et au « MatrixPanel » quel est le bord en cours de traitement, afin qu'il puisse être colorié en rouge.

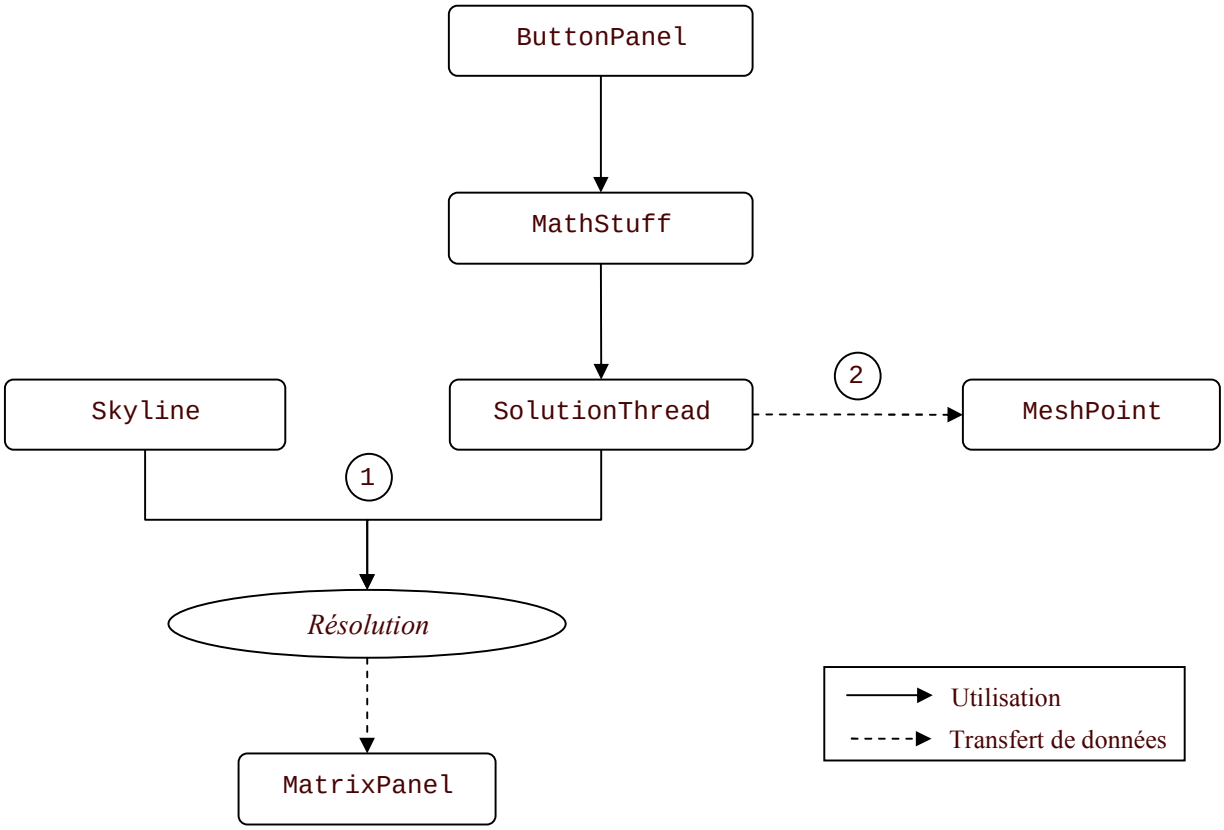
2.4.5. Prise en compte des conditions aux limites de type 1



1) La prise en compte des conditions aux limites de type 1 est traitée d'une part lors du processus d'assemblage, et d'autre part lors de la prise en compte des conditions aux limites de type 3. Il ne reste alors plus qu'à placer des « 1 » sur la diagonale de la matrice de rigidité globale et les valeurs des nœuds de Dirichlet sur le vecteur second membre. Comme les conditions de Dirichlet ne sont pas définies sur des bords mais sur des nœuds, le programme procède à une boucle sur les nœuds du maillage.

2) L'objet « SimulBCT1Thread » indique au « MeshPanel » et au « MatrixPanel » quel est le nœud en cours de traitement, afin qu'il puisse être colorié en vert.

2.4.6. Résolution du problème



1) Le thread commence par appeler l'algorithme de résolution du problème, en lui fournissant la Skyline en paramètre. L'objet « Skyline » contient la matrice de rigidité et quelques méthodes utilisées pour la manipuler : décomposition de Cholesky, résolution de systèmes d'équations linéaires, etc. Le « MatrixPanel » est alors appelé pour afficher la matrice triangulaire supérieure S de la décomposition S^TS, ainsi qu'une petite animation qui symbolise les algorithmes de substitution montante et descendante.

2) Les températures solution sont attribuées aux « MeshPoint ».

2.5. Algorithmes

2.5.1. Calcul et assemblage de la matrice de rigidité et du vecteur second membre

Le système global à résoudre est établi au niveau des éléments. Le programme assemble dans un premier temps la matrice de rigidité globale et le vecteur second membre sans prendre en compte les conditions aux limites.

Lors du calcul des matrices de rigidité élémentaires $K^{(r)}$, on procède à un changement de variables dans les intégrales afin de n'avoir à les évaluer que sur l'élément de référence $\widehat{T}$.

Matrice de rigidité élémentaire $K^{(r)}$ correspondant à l'élément $T^{(r)}$:

$$K^{(r)} = \left[\int_{\widehat{T}} \left[\left(grad_{\xi} \varphi_i(\xi) \right)^T \left(J^{(r)} \right)^{-1} \Lambda \left(J^{(r)} \right)^{-T} grad_{\xi} \varphi_j(\xi) \right] \left| \det J^{(r)} \right| d\xi \right]_{i,j=1}^{\widehat{N}}$$

où :

- ξ : coordonnées sur l'élément de référence
- φ_i et φ_j : fonctions de forme définies sur l'élément de référence
- $\Lambda = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$: coefficients de conductivité thermique du matériau
- $\widehat{N}$: nombre de nœuds par élément
- $J^{(r)}$: matrice jacobienne de la transformation qui permet de passer de l'élément de référence $\widehat{T}$ à un élément quelconque $T^{(r)}$ du maillage.

Seules les matrices $J^{(r)}$, $(J^{(r)})^{-1}$ et les dérivées des fonctions de forme définies sur l'élément de référence sont nécessaires pour le calcul des matrices de rigidité élémentaires. Les formules explicites des fonctions de forme de chaque élément $T^{(r)}$ ne sont pas requises.

Les vecteurs second membre élémentaires $f^{(r)}$ sont calculés de la même façon :

$$\underline{f}^{(r)} = \left[\int_{\bar{T}} f\left(x_{T^{(r)}}(\xi)\right) \varphi_i(\xi) \left| \det J^{(r)} \right| \mathrm{d}\xi \right]_{i=1}^{\bar{N}}$$

où : $x_{T^{(r)}}(\xi) = J^{(r)}\xi + x_1^{(r)}$ représente les coordonnées relativement à l'élément $T^{(r)}$

Lors de ce premier processus, la matrice de rigidité et le vecteur second membre sont aussi modifiés pour tenir compte des conditions aux limites de Dirichlet.

La matrice de rigidité est corrigée de la sorte :

$$K_{ij}^{(r)} = K_{ji}^{(r)} = \delta_{ij} \qquad \forall i \in \gamma_h^{(r)}, j \in \bar{\omega}_h^{(r)}$$

où $\bar{\omega}_h^{(r)}$: ensemble contenant les numéros de tous les nœuds de l'élément $T^{(r)}$

On corrige aussi le second membre :

$$f_i^{(r)} := f_i^{(r)} - \sum_{j \in \gamma_h^{(r)}} K_{ij} g_1(x_j) \qquad \forall i \in \omega_h^{(r)}$$

où :

- $g_1(x_j)$: valeur des conditions de Dirichlet au point x_j
- $\omega_h^{(r)}$: ensemble contenant les numéros des nœuds qui ne sont pas de Dirichlet dans l'élément $T^{(r)}$
- $\gamma_h^{(r)}$: ensemble contenant les numéros des nœuds de Dirichlet de l'élément $T^{(r)}$

Il aurait aussi été possible d'effectuer ces corrections à la fin du processus d'assemblage mais c'est plus facile de le faire au niveau élémentaire où les matrices et les vecteurs sont naturellement plus petits. De plus, il est coûteux avec le format « *Skyline* », une fois la matrice totalement assemblée, de trouver les lignes qui doivent être mises à zéro.

Les matrices de rigidité et les vecteurs second membre élémentaires sont alors assemblés par l'algorithme suivant qui utilise une boucle sur chaque domaine, car ceux-ci peuvent avoir des coefficients de conductivité différents.

Pour chaque domaine

Pour chaque élément $T^{(r)}$ du domaine

Calcul de $K^{(r)}$ et de $\underline{f}^{(r)}$

Prise en compte des conditions aux limites de Dirichlet : Correction de $\underline{f}^{(r)}$ et de $K^{(r)}$

Pour chaque nœud de l'élément de numéro global i et de numéro local k .

$$f_i := f_i + f_k^{(r)}$$

Pour chaque noeud de l'élément de numéro global j et de numéro local l

L'étape suivante consiste à prendre en compte les conditions aux limites :

Conditions aux limites de type 2 :

Vecteur qui doit être assemblé pour les conditions aux limites de Neumann :

$$\underline{f}^{(e_2)} = \left[\left(\int_0^1 \varphi_i(\xi) d\xi \right) g_2 \sqrt{\left(x_{n_2} - x_{n_1}\right)^2 + \left(y_{n_2} - y_{n_1}\right)^2} \right]_{i=1}^{\hat{N}_E}$$

où :

- φ_i : fonctions de forme définies sur l'intervalle de référence
- $\hat{N}_E$: nombre de nœuds par arête
- n_1 et n_2 : extrémités de l'arête
- g_2 : valeur de la condition aux limites sur l'arête

Conditions aux limites de type 3 :

Vecteur qui doit être assemblé pour les conditions aux limites de Robin :

$$\underline{f}^{(e_3)} = \left[\left(\int_0^1 \varphi_i(\xi) d\xi \right) \alpha \, u_A \sqrt{\left(x_{n_2} - x_{n_1}\right)^2 + \left(y_{n_2} - y_{n_1}\right)^2} \right]_{i=1}^{\bar{N}_E}$$

où α et u_A sont les valeurs de la condition aux limites sur l'arête

Matrice qui doit être assemblée pour les conditions aux limites de Robin :

$$K^{(e_3)} = \left[\left(\int_0^1 \varphi_i(\xi) \varphi_j(\xi) d\xi \right) \alpha \sqrt{\left(x_{n_2} - x_{n_1}\right)^2 + \left(y_{n_2} - y_{n_1}\right)^2} \right]_{i=1}^{\bar{N}_E}$$

Le programme corrige aussi bien sûr ce vecteur et cette matrice pour prendre en compte les conditions aux limites de type 1. Ils sont corrigés de façon analogue à ce qui a déjà été vu pour les matrices de rigidité et les vecteurs second membre élémentaires.

Ces matrices et vecteurs sont assemblés par le même algorithme que celui vu précédemment.

Conditions aux limites de type 1 :

Comme les conditions aux limites de Dirichlet ont déjà été en partie traitées lors du calcul de la matrice de rigidité et du vecteur second membre sans prise en compte des conditions aux limites et lors de la prise en compte des conditions aux limites de Robin, il ne reste plus désormais qu'à placer des « 1 » sur la diagonale de la matrice de rigidité globale et les valeurs des nœuds de Dirichlet sur le vecteur second membre :

Pour chaque nœud de Dirichlet de numéro global i :

$K_{ii} = 1$

$f_i = g_1(x_i)$

où $g_1(x_i)$ est la valeur de la condition de Dirichlet au point x_i.

2.5.2. Résolution du système

Les différents processus d'assemblage conduisent à l'équation linéaire suivante : $K\,u = f$.
K est la matrice de rigidité, f le vecteur second membre et u le vecteur solution cherché.

Pour résoudre une telle équation issue de la méthode des éléments finis, on peut utiliser des méthodes directes ou des méthodes itératives. Les méthodes de résolution directe consistent en des algorithmes qui renvoient le vecteur solution u après un certain nombre d'étapes. Les méthodes itératives construisent une suite (u_k), qui, à partir d'une première approximation u_0 converge vers le vecteur solution u.

Nous savons que la matrice K est symétrique définie positive. Dans ce cas, la *méthode de Cholesky*, qui fait partie des méthodes de résolution directe, est fréquemment utilisée.

La méthode de Cholesky consiste à écrire la matrice K comme le produit d'une matrice triangulaire supérieure par sa transposée : $K = S^T S$

Nous savons aussi que la matrice K est creuse. Cette propriété est conservée lors de la factorisation de Cholesky. Ainsi, les matrices K et S peuvent être sauvegardées dans des emplacements mémoire similaires. De plus, le programme peut utiliser un algorithme de décomposition particulier qui ne calcule pas les éléments nuls de S, ce qui est très bénéfique :

Algorithme de décomposition ($K = S^T S$) avec prise en compte du profil de K :

$$S_{11} = \sqrt{K_{11}}$$

Pour $j = 2, 3, \ldots, N$

Si $l_0(j)+1 < j$, alors pour $i = l_0(j)+1, l_0(j)+2, \ldots, j-1$:

$$S_{ij} = \frac{1}{S_{ii}}\left(K_{ij} - \sum_{l=\max\{l_0(i),l_0(j)\}+1}^{i-1} S_{li}S_{lj} \right)$$

$$S_{jj} = \sqrt{K_{jj} - \sum_{l=l_0(j)+1}^{j-1} S_{lj}^2}$$

(où $l_0(j)$ est le numéro de ligne pour lequel, dans la $j^{ème}$ colonne, $K_{ij} = 0$ pour $0 < i < l_0(j) + 1$ et $K_{l_0(j)+1,\,j} \neq 0$)

L'équation $Ku = f$ est alors équivalente à $S^T S\, u = f$. Le programme résout ce système en deux étapes grâce à une *descente* et à une *remontée*. Cela signifie qu'il résout d'abord $S^T y = f$ puis $S\, u = y$. Ces deux systèmes sont particulièrement faciles à résoudre car ce sont des systèmes triangulaires.

Algorithme de descente de Cholesky :

$$\begin{pmatrix} S_{11} & 0 & \cdots & 0 \\ S_{12} & S_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ S_{1N} & S_{2N} & \cdots & S_{NN} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}$$

$$y_1 = \frac{f_1}{S_{11}}$$

$$y_i = \frac{1}{S_{ii}} \left(f_i - \sum_{j=1}^{i-1} S_{ji} y_j \right) \qquad i = 2,\ 3,\ \ldots,\ N$$

Algorithme de remontée de Cholesky :

$$\begin{pmatrix} S_{11} & S_{12} & \cdots & S_{1N} \\ 0 & S_{22} & \cdots & S_{2N} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & S_{NN} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

$$u_N = \frac{y_N}{S_{NN}}$$

$$u_i = \frac{1}{S_{ii}} \left(y_i - \sum_{j=i+1}^{N} S_{ij} u_j \right) \qquad i = N-1,\ N-2,\ \ldots,\ 1$$

3. Autres fonctionnalités

Nous avons jusqu'à présent évoqué l'implémentation de la base du programme, c'est-à-dire ce qui permet de résoudre un problème. Toutefois, pendant le développement du logiciel, nous avons aussi pensé à d'autres fonctionnalités qui se sont également avérées très utiles. Même si nous avons eu seuls l'initiative de les implémenter, ce fut naturellement toujours avec l'accord de notre tuteur. Les trois fonctionnalités décrites ci-dessous sont, selon nous, celles qui enrichissent au mieux le programme.

3.1. Renumérotation des nœuds

Comme les fonctions de forme ont un support qu'on peut qualifier de "local" (quelques éléments), la matrice de rigidité est creuse. Les éléments $K_{ij} = a(p_i,p_j)$ ne sont en effet différents de zéro que lorsque l'intersection des intérieurs des supports des fonctions p_i et p_j est non vide.

De plus, avec une numérotation des nœuds appropriée, la matrice a une structure en bande. La largeur de la bande, c'est-à-dire le plus gros écart entre un élément non nul d'une colonne à l'élément de la diagonale correspondant, dépend de la numérotation des nœuds.

Il est très profitable d'utiliser ces deux propriétés pour le stockage de K. Le programme utilise le format dit « Skyline » pour le stockage de la matrice : seuls les éléments situés dans l' « enveloppe » (ou « profil ») sont stockés, c'est-à-dire entre le 1er élément non nul d'une colonne et l'élément de la diagonale correspondant.

La structure et le degré d'occupation de la matrice de rigidité dépendent de la numérotation des nœuds. Il est en effet important que les nœuds de chaque élément aient des numéros voisins car ceci conduit à un profil plus petit et donc à un nombre d'éléments stockés plus faible.

A partir d'une numérotation des nœuds donnée, le programme peut procéder à une renumérotation avec l'algorithme de Cuthill McKee pour réduire la taille du profil de la matrice.

Le programme peut utiliser des fonctions de forme quadratiques (voire cubiques) à partir de fichiers .net écrits pour des fonctions de forme linéaires. Dans ce cas, seuls les 3 sommets des

triangles sont numérotés et le programme ajoute les nœuds supplémentaires (3 par triangle dans le cas des fonctions de forme quadratiques) en leur attribuant les numéros qui suivent. Il est possible de procéder automatiquement, dès le chargement des fichiers, à une renumérotation des nœuds pour accélérer le processus de résolution du problème.

Algorithme de Cuthill McKee :

L'idée qui est à la base de l'algorithme est de prendre en compte les nœuds voisins le plus conjointement possible dans le processus de renumérotation, afin d'éviter des écarts trop importants dans les numéros des nœuds d'un même élément, qui conduisent à de mauvais profils. Ainsi par exemple, le fait de numéroter les nœuds d'un même niveau selon leur degré croissant (le degré *d(z)* d'un nœud *z* correspond au nombre de ses voisins) repose sur cette constatation triviale que les nœuds avec beaucoup de voisins doivent avoir des numéros globaux les plus grands possibles afin de conserver des écarts d'index faibles dans le niveau suivant.

1. Détermination des voisins de la racine *r*. Ces nœuds doivent être numérotés selon leur degré croissant. Lorsque des nœuds ont le même degré, un choix arbitraire est évidemment nécessaire. Les nœuds numérotés dans cette étape sont à la distance 1 de la racine *r*. Ils forment le 1^{er} niveau.

2. Les nœuds du 1^{er} niveau sont considérés successivement selon leur nouveau numéro. Pour chacun d'eux, numérotation de leurs voisins non renumérotés par degré croissant. Les nœuds qui sont numérotés lors de cette étape sont à la distance 2 de la racine et forment le $2^{ème}$ niveau du processus de renumérotation.
   On répète alors ce processus jusqu'à ce que tous les nœuds aient été renumérotés.

3. Il a été prouvé que le profil pouvait encore être relativement réduit en inversant l'ordre de numérotation des nœuds qui est généré par cet algorithme

Implémentation de l'algorithme :

(1) Recherche du nœud initial ou « racine » r. Il reçoit le numéro 1.
$S = \{r\}$

(2) Tant que tous les nœuds n'ont pas été renumérotés :
- $S_{nou} = \{\}$
- Pour chaque nœud x de S,
  - Détermination de tous les voisins du nœud x qui n'ont pas déjà été renumérotés : $\{k_1, k_2, ..., k_r\}$, et numérotation selon leur degré croissant.
  - $S_{nou} = S_{nou} + \{k_1, k_2, ..., k_r\}$
- $S = S_{nou}$

(3) Inversion de la numérotation grâce à la substitution : $k \rightarrow n+1-k$

Algorithme pour la détermination de la racine :

Pour cette étape, on divise le maillage en niveau.
$R(g) = (L_1, L_2, ..., L_r)$ s'appelle « la structure en niveaux avec racine g », lorsque :

1. $\bigcup_{i=1}^{r} L_i$ = Ensemble des nœuds du maillage
2. L_r est non vide
3. $L_1 = \{g\}$
4. Le plus court chemin entre un nœud $h \in L_i$ et g comporte i-1 arêtes

r s'appelle « la profondeur » de la structure en niveaux. Sa largeur k est le nombre de nœuds du niveau qui en contient le plus.

Gibbs, Poole et Stockmeyer ont proposé d'utiliser des structures en niveaux avec la largeur la plus faible. Celle-ci correspond en général à la structure en niveau qui a la plus grande profondeur. En effet, logiquement, plus il y a de niveaux, moins il y a de nœuds par niveaux.
Idéalement, il faudrait commencer par déterminer ce que l'on appelle « le diamètre » du maillage, qui correspond au chemin le plus court entre deux points du maillage les plus éloignés possible. Le problème est bien sûr resoluble mais l'on se contente la plupart du temps d'un algorithme « approximatif ».
Avec un tel algorithme, g n'est pas toujours l'extrémité d'un diamètre du maillage mais il s'en approche. On parle souvent dans ce cas d'un « pseudo-diamètre ».
L'algorithme suivant est très approprié à la recherche d'un bon nœud de départ, car les dépenses en terme de temps de calcul et de place mémoire sont relativement restreintes, étant donné qu'en règle générale, peu de nœuds doivent être testés pour trouver un pseudo-diamètre.

(1) Choix d'un nœud g de degré minimum

(2) Construction de la structure en niveaux $R(g) = (L_1, L_2, \ldots, L_r)$. Tri des éléments f_j du dernier niveau L_r selon leur degré croissant, c'est-à-dire $d(f_j) \leq d(f_{j+1})$

(3) Poser $j = 1$

(4) Calcul de la profondeur de la structure en niveaux $R(f_j)$: s. Si $s > r$, poser $g = f_j$ et retour à l'étape (2)

(5) Si $j < l$, incrémentation de j et retour à l'étape (4)

3.2. Parseur

Un parseur est un programme, ou une partie de programme, qui analyse syntaxiquement un texte et le découpe en éléments. Ce texte peut être une entrée de l'utilisateur (grâce à une ligne de commande par exemple), le code source d'un programme ou un fichier qui contient des instructions de formatage comme par exemple des tags HTML. Le but de notre parseur est de trouver la valeur de la fonction *f*. Cette fonction est écrite dans un fichier .net au format texte et lue sous forme d'un objet String (par exemple *f = 2*X+exp(Y)*). Nous cherchons la valeur de cette fonction aux différents points du maillage afin de calculer le vecteur second membre.

L'objet String est tout d'abord découpé en objets « Token ». Un Token représente une fonction mathématique (sin(), cos(), …), une variable (X ou Y), un nombre, un opérateur (+, -, …) ou une parenthèse. Afin de calculer la valeur de la fonction, le programme a besoin des coordonnées des points où la fonction doit être évaluée. Tout d'abord, les Tokens qui représentent une variable sont remplacés par la coordonnée correspondante. La fonction *f* est ensuite évaluée grâce à l'algorithme suivant :

Recherche des parenthèses intérieures (celles qui ne contiennent pas d'autres parenthèses)
Calcul de l'expression qui est entre ces parenthèses :
a. Remplacement des Tokens qui représentent une fonction par la valeur de cette fonction
b. Evaluation des multiplications puis des additions
c. Remplacement de l'expression entière par sa valeur
S'il reste des parenthèses, retour à l'étape 1.
Calcul de l'expression restante (elle ne comporte aucune parenthèse)

A la fin de ce processus, il ne reste plus qu'un objet Token qui contient une valeur. C'est la valeur de la fonction *f* au point considéré.

3.3. Raffinage du maillage

Lorsque le programme travaille avec des triangles et des fonctions de forme p_i linéaires par morceaux, il existe une possibilité de raffiner le maillage. Après un tel raffinage, une nouvelle procédure de résolution est exécutée avec le maillage raffiné. Lorsque l'utilisateur demande un raffinage, il doit entrer un écart de température maximal. Le programme raffine alors le maillage jusqu'à ce que l'écart de température entre les 3 sommets de chaque triangle soit inférieur à la valeur seuil souhaitée. Si l'utilisateur a entré 0, le maillage est raffiné entièrement une fois (chaque triangle est divisé en quatre).

Algorithme de raffinage du maillage :

(1) Partage des triangles pour lesquels l'écart de température entre deux sommets est trop important. Pour effectuer cette division, on crée un triangle dont les sommets sont les milieux des 3 côtés du triangle initial.

(2) Tant que des divisions de triangles sont effectuées

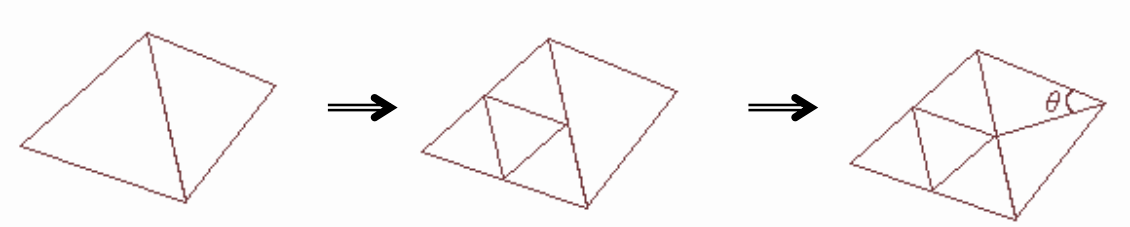
   (i) Dans le maillage raffiné, partage en quatre des triangles qui n'ont pas déjà été divisés en quatre, mais où des nœuds ont été générés sur au moins deux de leurs côtés.

   (ii) Pour chaque triangle où exactement un nœud a été généré sur un côté :

      - Si un partage en deux du triangle générerait deux triangles de mauvaise qualité (c'est-à-dire que l'angle θ serait trop petit, $\theta < 25°$), alors partage du triangle en quatre.

      - Sinon, partage du triangle en deux, en reliant ce nœud généré par une précédente division, au sommet opposé.

(3) Résolution du problème avec le maillage raffiné. S'il reste des écarts de température supérieurs à la valeur seuil donnée, retour à l'étape (1).



Raffinage de maillage avec partage en quatre des triangles

4. Bilans

4.1. Limites

Nous avons utilisé différentes méthodes qui permettent d'accélérer la résolution du problème. La Skyline prend ainsi par exemple en compte le fait que la matrice de rigidité globale soit creuse. Cependant, FEJavaDemo reste un logiciel destiné à l'apprentissage et ne peut évidemment pas rivaliser avec un « véritable » programme d'éléments finis en ce qui concerne la rapidité d'exécution. Les objectifs poursuivis sont différents et nous considérons que FEJavaDemo est suffisamment rapide pour l'utilisation qui lui est prescrite.

De plus, le programme a été développé pour résoudre des systèmes d'équations différentielles scalaires linéaires du second ordre, comme le problème de conductivité thermique stationnaire. C'est suffisant pour comprendre la méthode des éléments finis, mais c'est évidemment restrictif. D'autres problèmes peuvent cependant aussi être modélisés par de tels systèmes : des problèmes issus de la thermodynamique, de l'électrostatique, de la magnétostatique ou encore de la mécanique. FEJavaDemo pourrait ainsi être modifié afin de traiter de tels problèmes. Ce n'était pas notre priorité mais de futurs programmeurs pourront éventuellement s'atteler à ces tâches.

4.2. Améliorations possibles

Nous avons semble-t-il atteint les objectifs initiaux qui nous étaient assignés et ajouté différentes fonctionnalités, mais le programme peut encore naturellement être amélioré. Nous avons pensé à différentes possibilités que nous n'avons pas eu le temps de réaliser :

4.2.1. Améliorations en rapport avec le maillage

Nous avons longtemps essayé d'implémenter des méthodes afin de générer des maillages. Avec FEJavaDemo, il est en effet pour le moment obligatoire de fournir les coordonnées de chaque nœud alors qu'il serait tellement avantageux de n'avoir à donner que la situation des contours du

domaine, ainsi que la densité d'éléments souhaitée. On peut trouver dans la littérature un nombre conséquent de travaux sur les mailleurs. Les méthodes de type quadtree-octree, ou celles de types Voronoï sont des exemples d'algorithmes de génération automatiques de maillage. L'ouvrage de notre tuteur présente succinctement les techniques d'avancée de front (« Advancing-Front » en anglais) et nous les avons trouvés particulièrement intéressantes pour FEJavaDemo car elles sont très « visuelles », en ce sens qu'il est facile de comprendre comment elles fonctionnent en examinant le maillage se former à l'écran, depuis les bords du domaine vers l'intérieur.
Ce n'est cependant pas aussi simple de les implémenter. Nous nous sommes efforcés de nous renseigner sur ce problème par Internet et par la consultation de livres spécialisés à la bibliothèque mais n'avons trouvé aucun algorithme clair. Nous avons ainsi préféré terminer dans un premier temps l'essentiel du logiciel et nous n'avons pas eu assez de temps à la fin pour ajouter encore une telle fonctionnalité.
Il faut cependant savoir que de nombreux programmes permettent de générer des maillages et que l'on peut les utiliser conjointement avec FEJavaDemo. Il est en effet inconcevable de construire manuellement un maillage avec une géométrie compliquée.

Le choix des éléments qui sont partagés lors d'un raffinage de maillage peut aussi être amélioré. FEJavaDemo examine pour le moment les écarts de température qui existent entre les différents sommets de chaque élément. Si un écart trop important est calculé, l'élément correspondant est divisé quatre. Cette stratégie nous est apparue naturelle et notre tuteur nous a aidé à la concrétiser. Ce n'est pourtant pas toujours la meilleure technique de raffinage. On peut par exemple imaginer un pic de température au milieu d'un élément qui passerait inaperçu même après plusieurs raffinage si les sommets ont des températures voisines. De tels cas peuvent être évités grâce à des estimations d'erreurs a posteriori s'appuyant par exemple sur des méthodes de « résidus ».

4.2.2. Techniques relatives aux éléments finis

Comme nous l'avons déjà mentionné, notre tuteur tenait à ce que FEJavaDemo soit modulable, en d'autres termes que l'on puisse facilement ajouter d'autres types de fonctions de forme et d'éléments. Nous avons naturellement toujours gardé cet objectif à l'esprit et nous avons eu le temps de tester cette modularité vers la fin de notre stage. Nous avons en effet implémenté les fonctions de forme quadratiques et les quadrilatères, alors que seules les fonctions de forme linéaires sur des triangles étaient attendues par notre tuteur. Ceci nous a permis d'améliorer le code et nous avons désormais l'impression qu'il est assez simple d'ajouter encore d'autres types de fonctions et d'éléments : il n'est pas nécessaire de maîtriser l'ensemble du code source. Nous avons en outre écrit des manuels pratiques qui décrivent les différentes étapes qui permettent de telles modifications.

Nous avons déjà évoqué la différence entre les méthodes directes et les méthodes itératives pour la résolution, avec la méthode des éléments finis, de systèmes tels que celui décrit précédemment. Nous avons aussi vu qu'avec une matrice K symétrique définie positive, il était tout indiqué d'utiliser la *méthode de Cholesky* qui est une méthode directe.
Néanmoins, l'utilisation de la méthode des gradients conjugués ou d'une autre méthode itérative permettrait d'établir une comparaison avec la méthode de Cholesky. Chaque méthode a en effet ses avantages et ses inconvénients et l'utilisateur pourrait choisir la méthode qui est la plus appropriée pour ce qu'il recherche (vitesse, efficacité, encombrement mémoire, etc.). A titre d'exemple, pour un problème avec 3593 inconnues, un programme avec la méthode de Cholesky nécessite 0,11 secondes et 1,31 MB de mémoire alors qu'un programme avec une méthode SOR (méthode itérative) a besoin de 5,97 secondes et de 193,19 kB de mémoire (avec le même processeur). Pour résoudre des problèmes en grandes dimensions, les méthodes itératives sont souvent plus efficaces alors que la méthode de Cholesky est un très bon choix pour des problèmes plus « simples ».

4.2.3. Affichage

Nous pensons que l'interface peut elle aussi être encore améliorée. Elle est certes bien meilleure que celle de l'ancien logiciel, mais certaines représentations et animations pourraient être ajoutées. Nous avons ainsi par exemple réfléchi à la façon de représenter les conditions aux limites. Dans sa version actuelle, FEJavaDemo permet déjà de voir quelles sont les frontières isolées thermiquement de l'extérieur. Nous avons aussi imaginé que des flèches de couleurs et de longueurs différentes pourraient représenter les flux entrants et sortants. Il serait aussi intéressant que la température soit affichée lorsque le curseur se déplace sur le maillage… Il existe encore bien d'autres possibilités d'amélioration dans ce domaine qu'un éventuel futur programmeur pourra explorer.

4.2.4. Comparaison avec la solution analytique

Pour tester la précision de la méthode et du logiciel, nous avons pensé qu'il serait intéressant de pouvoir fournir la solution analytique du problème. Grâce à cette solution et aux conditions aux limites, le programme pourrait retrouver les données initiales du problème. En lançant alors la procédure de résolution avec la méthode des éléments finis, il serait possible de comparer la solution « éléments-finis » avec la solution analytique. Cela pourrait non seulement servir à améliorer la méthode de résolution, mais cette comparaison aurait surtout des vertus pédagogiques car elle permettrait de mettre en évidence les relatives faiblesses de la MEF. L'influence de différents paramètres (type des fonctions de forme, des éléments, niveau de raffinage du maillage) sur la précision de la solution pourrait aussi être soulignée.

4.3. Perspectives

FEJavaDemo est d'ores et déjà utilisé à l'université de Dresde et nous estimons que d'autres établissements pourraient également s'en servir. Il serait notamment intéressant que les professeurs de l'ENPC travaillent avec ce logiciel. Celui-ci leur permettrait d'illustrer simplement leurs cours sur la méthode des éléments finis et aiderait les élèves à comprendre le principe de la méthode. En ce qui concerne le premier semestre à l'ENPC, nous pensons notamment aux cours de calculs scientifiques et de mécanique. De plus, il serait souhaitable que le logiciel, déjà disponible sur Internet, soit aussi à disposition de tous les élèves sur les ordinateurs de l'école.

Comme FEJavaDemo est « open-source » c'est-à-dire que quiconque peut modifier le code du programme afin de l'améliorer, nous espérons aussi que des utilisateurs en profiteront et ajouteront de nouvelles fonctions. Un autre étudiant de l'ENPC pourrait aussi éventuellement continuer notre travail lors d'un stage ultérieur.

4.4. Impressions personnelles

4.4.1. Jérémie

Ce stage scientifique à l'université technique de Dresde a été pour moi une très bonne expérience. Je pense que ce stage constituait un bon moyen de terminer la première année car il permettait de faire le lien entre la théorie et la pratique.
J'ai eu l'occasion d'améliorer mes connaissances en mathématique et en informatique mais j'ai aussi découvert le monde de l'université et de la recherche. Bien que je ne veuille pas faire de recherche plus tard, cette expérience m'a aidé à réfléchir à mon projet professionnel. Ce stage fut très enrichissant aussi bien au niveau scientifique que personnel.

Tout d'abord, je souhaiterais encore une fois remercier le professeur Walter pour nous avoir aidé à résoudre tous les problèmes pratiques de notre stage. Je souhaiterais aussi souligner la bonne ambiance de travail qui régnait à l'institut ainsi que la disponibilité de notre tuteur. Michael Jung et Kshitij Kulshreshtha nous ont en effet aidé tout au long de notre stage pour tout ce qui concernait les mathématiques et l'informatique. Nous avons aussi eu la possibilité d'assister à des cours : des cours d'allemand avec le programme d'échange Erasmus et des cours de Java avec le professeur Walter. Ces cours m'ont permis d'approfondir mes connaissances mais aussi de mieux connaître la vie à l'université.

Au début du stage, j'avais un fort intérêt pour la programmation ainsi que pour le calcul scientifique et je souhaitais effectuer un stage dans l'un de ces deux domaines. Il se trouve que ce stage combinait les deux, c'était donc pour moi un sujet idéal. J'ai aussi apprécié le fait d'avoir mené notre projet d'un bout à l'autre. L'idée principale venait naturellement de notre tuteur mais nous avons eu beaucoup de liberté en ce qui concerne les fonctionnalités et la structure du programme. J'ai l'impression que nous avons réalisé quelque chose d'utile et pas seulement une partie d'un gros projet que nous n'aurions pas entièrement compris.

En ce qui concerne le travail à deux, c'était la première fois que je devais réaliser un si gros projet en binôme mais je pense que tout s'est bien déroulé. Nous avions au début les mêmes connaissances en informatique et en calcul scientifique et cela nous a permis de progresser à la

même allure. Nous avons tout d'abord travaillé sur les mêmes algorithmes afin que chacun comprenne bien la structure du programme, nous nous sommes ensuite partagé le travail. Toutefois, nous avons toujours programmé en étroite collaboration.

Je pense qu'avec la formation de première année, nous avions de bonnes bases en informatique et calcul scientifique. Nous avons tout de même dû apprendre le Java mais cela n'était pas trop difficile avec nos connaissances en C++. Il nous a aussi été nécessaire d'approfondir le calcul scientifique en particulier en ce qui concerne les conditions aux limites et les méthodes d'assemblage. Je pense que les acquis de première année nous ont permis de comprendre rapidement ces concepts.
Pour le choix de mon département, j'ai longtemps hésité entre SEGF et IMI. Ce stage a confirmé mon intérêt pour la programmation, mais il m'a aussi montré que je ne souhaite pas faire le métier de programmeur. En conclusion, je maintiens mon choix en faveur de SEGF, mais j'espère pouvoir pendre l'an prochain quelques cours de la filière IMI.

4.4.2. Aurélien

Ce stage à l'Université Technique de Dresde fut pour moi une expérience remarquable à tous égards.

Une excellente expérience humaine car il fut très intéressant de rester trois mois dans un pays étranger et de se pénétrer d'avantage de son mode de vie. J'ai évidemment cherché à améliorer mes connaissances en allemand le plus possible, par exemple en essayant de faire un maximum de rencontres ou encore en suivant des cours d'allemand proposés par l'université. Afin de profiter pleinement de notre séjour au plan culturel, nous avons également visité, entre autres, Dresde, Berlin, Prague, Weimar et le camp de concentration de Buchenwald.

En ce qui concerne le travail en binôme, ce stage fut aussi une expérience très formatrice car je n'avais encore jamais aussi longuement collaboré à un projet avec une tierce personne. Nous

avons su, je pense, travailler souvent en commun lorsque cela s'avérait être plus productif, et d'autres fois séparer les tâches, lorsque cela s'y prêtait.

Ce stage fut avant tout une très bonne expérience professionnelle. D'abord parce que je n'exclus pas la possibilité de travailler plus tard dans le monde de la recherche, sur lequel ce stage m'a offert un premier regard. Mais aussi grâce au fait que le développement d'un logiciel du début à la fin est très enrichissant car on a vraiment le sentiment d'avoir fait quelque chose qui pourra être utilisé dans le futur. Notre tuteur nous a laissé suffisamment de liberté pour implémenter des idées de fonctions susceptibles d'améliorer le programme que nous avons eu seuls. Ce fut ainsi pour moi particulièrement stimulant de travailler sur une fonction dont nous avons eu nous-même l'initiative pour essayer de satisfaire au mieux le tuteur. J'évoque ici par exemple la renumérotation des nœuds et le raffinage du maillage.
Ce stage aurait pu ne consister qu'en de la programmation, ce qui aurait pu être un peu fastidieux. Nous avons en fait eu également la possibilité de compléter nos connaissances sur la méthode des éléments finis et ce fut ainsi une très bonne prolongation de nos cours de mécanique et de calcul scientifique de 1ère année. En ce qui concerne la programmation en elle-même, il fut souvent instructif d'étudier quelques méthodes algorithmiques, comme les différentes façons de stocker des matrices creuses. De plus, le fait d'avoir travaillé avec Java et non avec C++ nous a permis d'avoir une vision plus large et il est toujours intéressant de pouvoir comparer la structure de différents langages de programmation.

Eu égard au choix de département de ma deuxième année, j'estime avec recul qu'il m'aurait été difficile de trouver un stage plus approprié. J'hésitais en effet alors entre IMI (Ingénierie Mathématique et Informatique) car les concepts de modélisation, de simulation et de programmation me passionnent, et GMM (Génie Mécanique des Matériaux), car la mécanique m'intéresse également depuis longtemps. Ainsi, la programmation d'un logiciel qui permet de traiter des problèmes physiques par une méthode souvent utilisée en mécanique s'est avérée idéale.

Bibliographie

Jung, M.; Langer, U.: *Methode der finiten Elemente für Ingenieure* Eine Einführung in die numerischen Grundlagen und Computersimulation. B.G. Teubner Stuttgart - Leipzig - Wiesbaden 2001

Neundorf, W.; *Wissenschaftliches Rechnen* Matrizen und LGS. Ilmenau 2002
http://www.mathematik.tu-ilmenau.de/~neundorf/education/wr/wr1.pdf

Horstmann, C.; Cornell, G.: *Core Java 2*. Sun Microsystems Press 1999

SunMicrosystems; *Java 2 Platform, API Specification*. Juin 2004
http://java.sun.com/j2se/1.4.2/docs/api/

Technische Universität Dresden. Juin 2004
http://www.tu-dresden.de/

Annexes

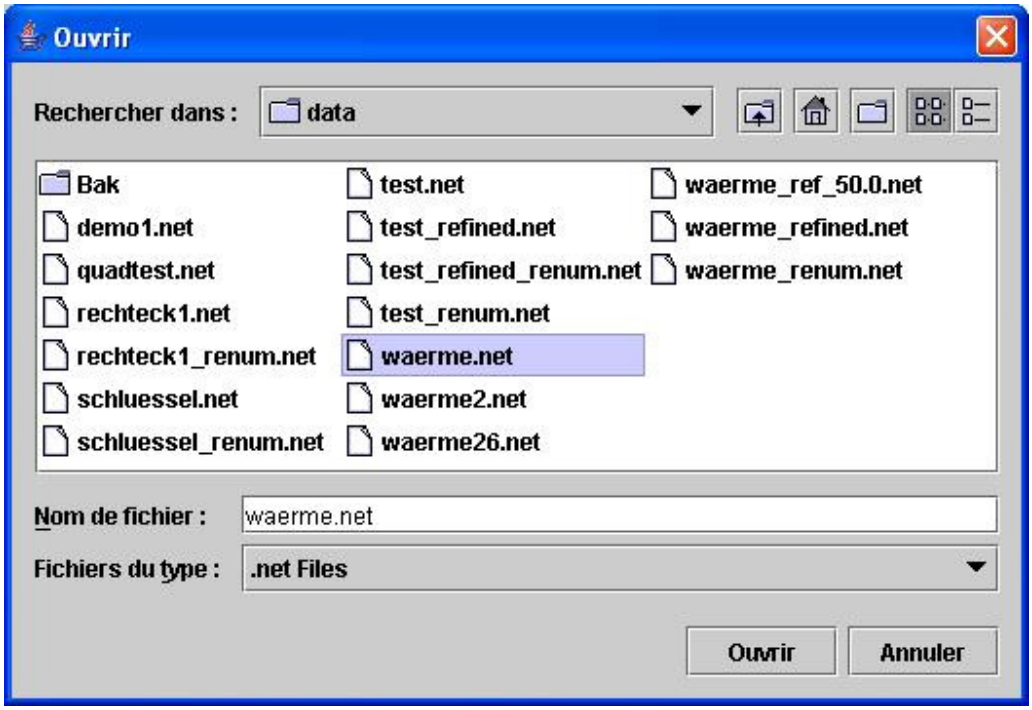
Annexe 1 – Déroulement du programme

Choix du type de fonctions de forme :

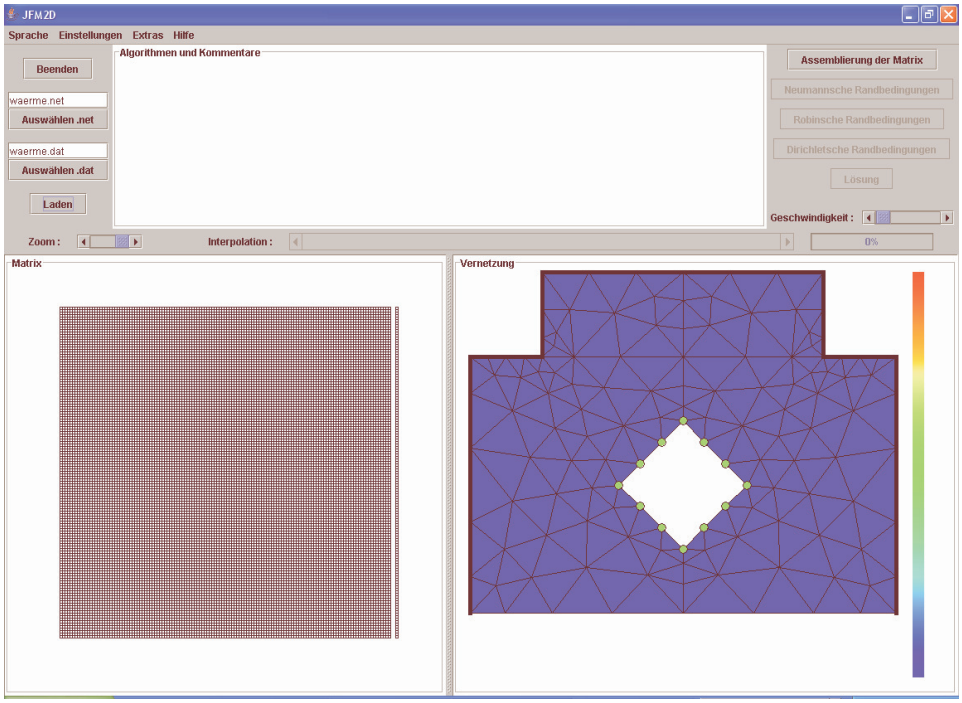
Le programme travaille par défaut avec des fonctions de forme linéaires. Grâce au menu « Propriétés → Type des fonctions de forme », on peut aussi choisir des fonctions de forme quadratiques.

Chargement des fichiers de données à utiliser :

En cliquant sur le bouton « Parcourir .net » (respectivement « Parcourir .dat »), les fichiers de maillages (respectivement les fichiers .dat) à disposition sont affichés.



Après avoir choisi les fichiers souhaités, les données seront chargées en cliquant sur le bouton « Chargement ».



Calcul de la matrice de rigidité et du vecteur second membre sans prise en compte des conditions aux limites :

Dès que le chargement des fichiers de données est terminé, le bouton « Assemblage de la matrice » devient actif. Il permet de démarrer le calcul et l'assemblage de la matrice de rigidité et du vecteur second membre sans prise en compte des conditions aux limites.

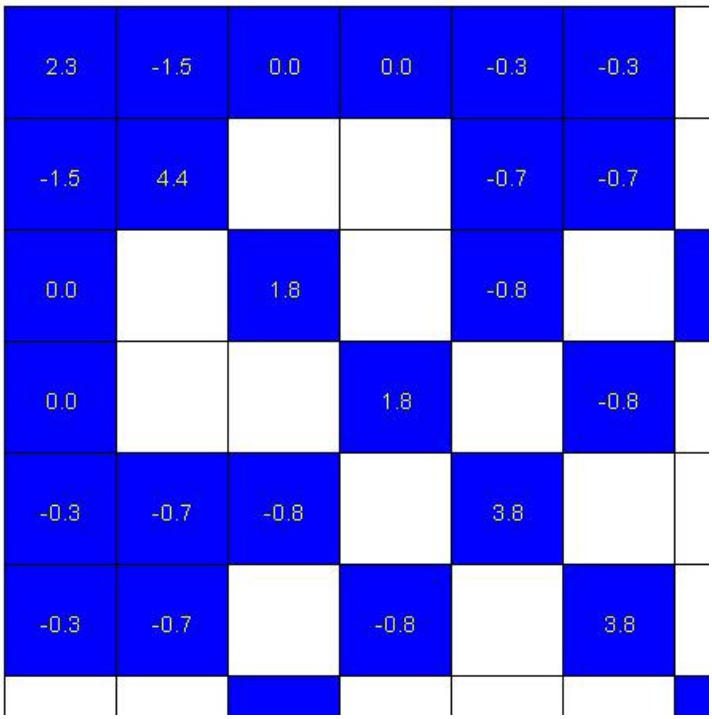
Pendant le déroulement du processus, un ascenseur permet de modifier la vitesse d'affichage. L'utilisateur préfère souvent que le programme livre la solution le plus rapidement possible. C'est pourquoi les étapes de l'assemblage ne sont plus affichées lorsque le curseur est placé à l'extrême gauche.

Prise en compte des conditions aux limites :

Les trois boutons suivants permettent la prise en compte des conditions aux limites.

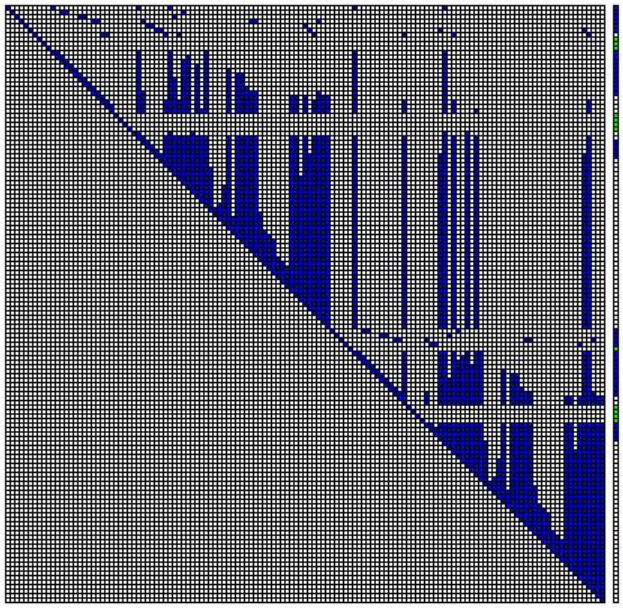
Lors des différents processus d'assemblage, les éléments du maillage sur lesquels travaillent le programme, ainsi que les cases correspondantes dans la matrice sont coloriés en rouge. Les éléments non nuls de la matrice sont en bleu afin de montrer que cette dernière est très creuse. Les cases vertes correspondent aux nœuds de Dirichlet.

Un ascenseur permet de modifier le zoom sur la matrice. Lorsque le curseur est à l'extrême gauche, les valeurs des cases de la matrice sont affichées. On peut aussi lire les numéros des lignes et des colonnes.

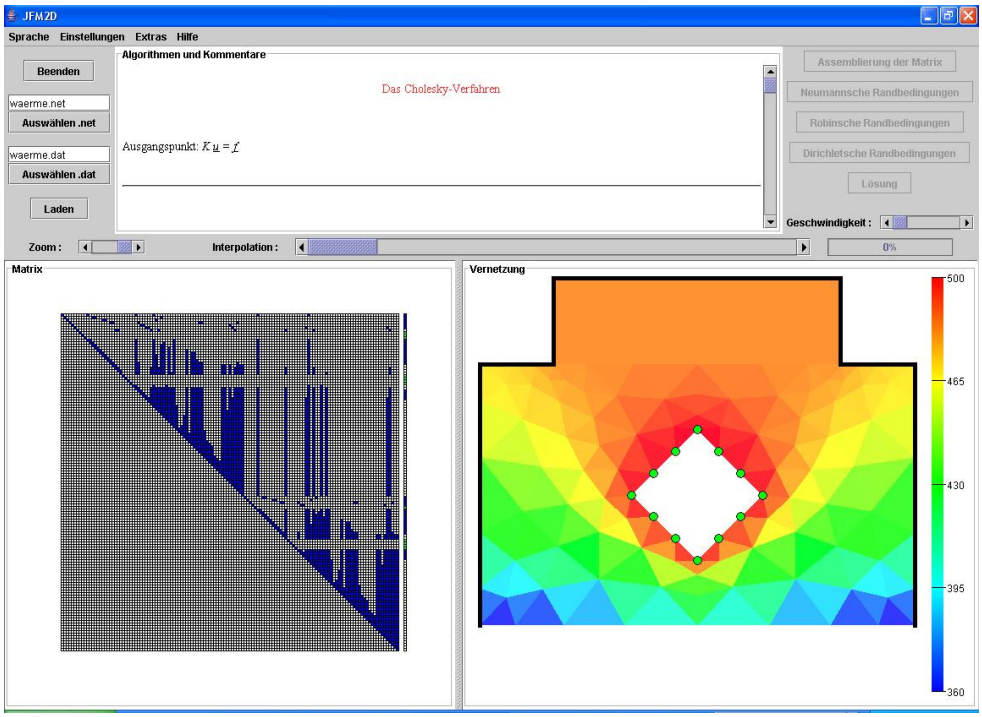


Résolution du système :

Lorsque toutes les conditions aux limites ont été prises en compte, le bouton « Résolution » devient actif. En cochant « Propriétés → Affichage → afficher la décomposition S'S », on peut observer la matrice diagonale supérieure S de la factorisation S^TS. Une petite animation symbolise le processus de descente et de remontée de Cholesky.

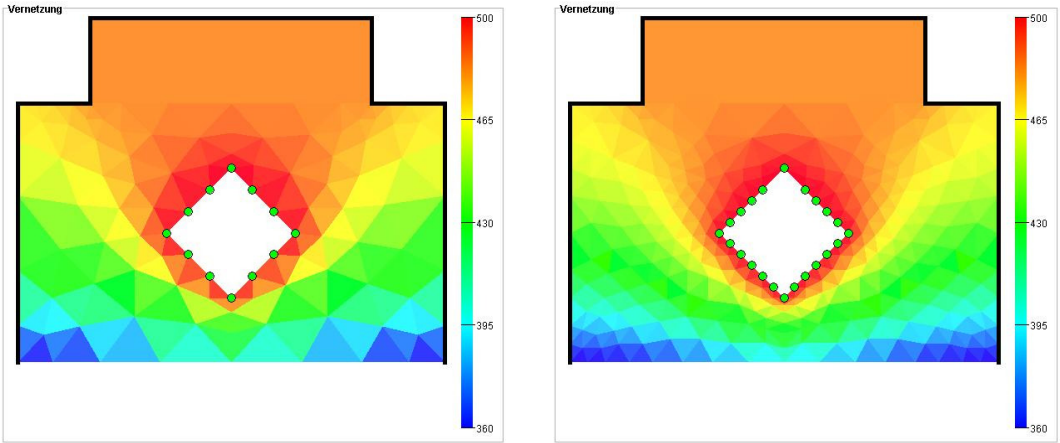


Lorsque la résolution est terminée, le champ de température est représenté, muni d'une échelle de température.



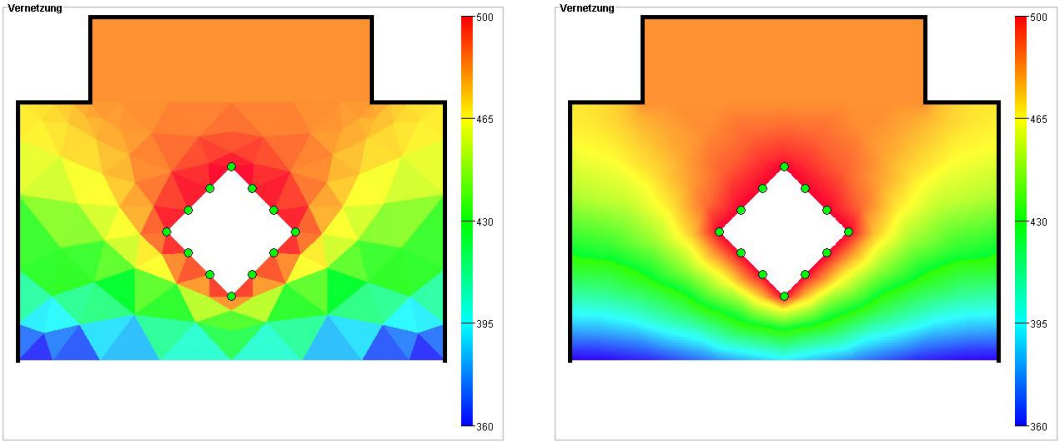
Raffinage du maillage :

Si le programme travaille avec des fonctions de forme linéaires sur des triangles, on peut demander un raffinage du maillage (« Accessoires → Raffinage du maillage »). Dans ce cas, l'utilisateur doit entrer l'écart maximal de température entre deux nœuds. S'il entre "0", le maillage sera entièrement raffiné une fois. Il est possible de sauvegarder les fichiers du nouveau maillage raffiné (« Accessoires → Création de nouveaux fichiers… → ... lors d'un raffinage »).



Interpolation de la solution :

Grâce à l'ascenseur « Interpolation », la représentation de la solution peut être améliorée: les éléments peuvent être divisés et la solution est alors interpolée linéairement aux nouveaux nœuds. Comme ceci peut être assez long, une barre d'avancement permet de connaître quel pourcentage du processus a déjà été exécuté.



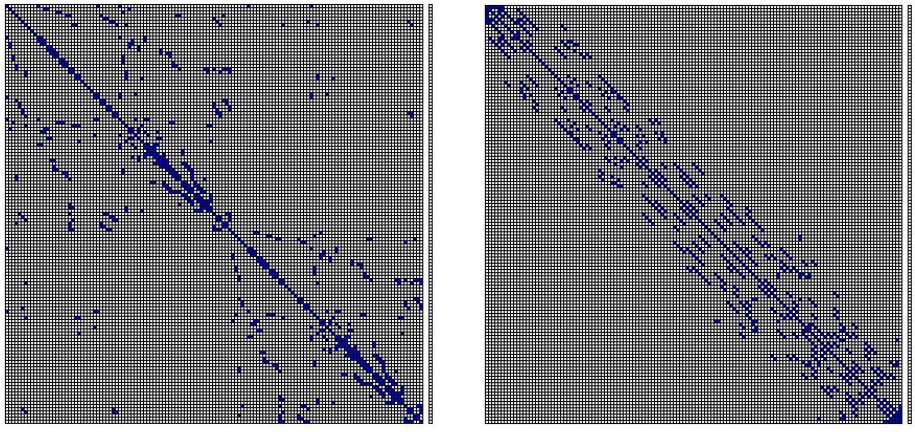
Sauvegarde de la solution :

La solution (plus précisément les coordonnées des nœuds et leur température) peut être sauvegardée par le menu « Accessoires → Sauvegarder la solution ».

Renumérotation des nœuds :

Cette possibilité est offerte par le menu « Accessoires → Renumérotation des nœuds ».

La case à cocher « Renumérotation automatique des nœuds avec les fonctions quadratiques » est choisie par défaut. Ceci permet d'accélérer le processus de résolution. Il existe aussi une possibilité de sauvegarder la nouvelle numérotation grâce au menu (« Accessoires → Création de nouveaux fichiers… → ... lors d'une renumérotation »).

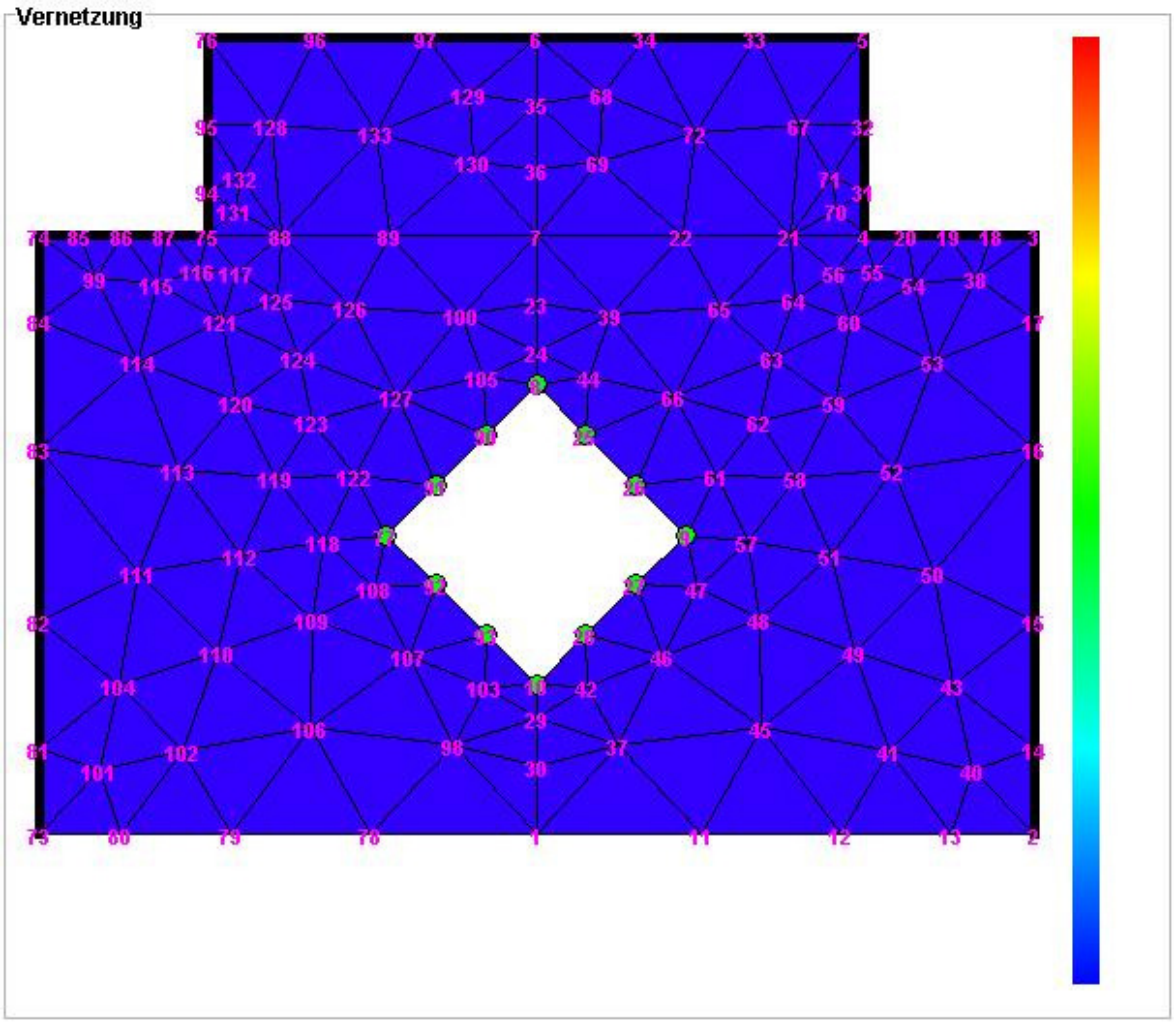


Autres possibilités :

Le logiciel peut bien sûr être traduit grâce au menu « Langue ».

Entre chaque étape, il est possible d'enregistrer des images Jpeg de la matrice et du maillage (« Accessoires → Création de Jpeg »).

Grâce au menu « Propriétés → Affichage → Afficher les conditions aux limites », il est possible de colorier les nœuds de Dirichlet en vert et de mettre en gras les bordures qui sont isolées thermiquement de l'extérieur. On peut aussi afficher les numéros globaux des nœuds (« Propriétés → Affichage → Afficher les numéros de nœuds »).



Annexe 2 – Structure des fichiers d'entrée

Pour pouvoir travailler sur un nouveau maillage, il faut écrire deux fichiers de données : un fichier ".net" et un fichier ".dat". Le premier contient les informations sur le maillage et le second sur les conditions aux limites et la fonction *f*.

Le fichier .net

Les lignes qui commencent par # sont des commentaires et sont ignorées par le programme.

La 1ère ligne décrit le type des éléments utilisés dans le maillage (1 pour triangles, 2 pour quadrilatères, etc.).

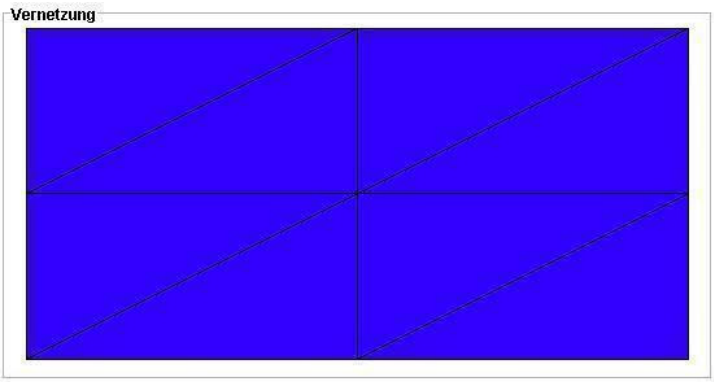
Suivent le nombre de nœuds (c'est-à-dire la dimension de la matrice de rigidité globale) et le nombre d'éléments.

Les nœuds sont alors définis grâce à 3 valeurs : leur numéro global, leur abscisse et leur ordonnée.

Ce sont ensuite les éléments qui sont définis : chacun a un numéro global suivi des numéros globaux de leurs sommets (3 numéros pour les triangles, 4 pour les quadrilatères, …) et de numéro du domaine dans lequel il se situe.

La ligne suivante contient le nombre d'arêtes-frontières. Celles-ci sont définies juste en dessous par leur numéro global ainsi que par les nœuds qui sont à leurs extrémités.

L'exemple qui suit est le fichier .net d'un maillage très simple :



```
# FEJavaDemo – Fichier .net
#
# Type des éléments (1 pour triangles, 2 pour quadrilatères, etc.)
  1
#
# Nombre de nœuds et nombre d'éléments
  9 8
#
# Numéros globaux et coordonnées des nœuds
  1  0.0  0.0
  2  0.0  1.25
  3  0.0  2.5
  4  2.5  0.0
  5  2.5  1.25
  6  2.5  2.5
  7  5.0  0.0
  8  5.0  1.25
  9  5.0  2.5
#
# Numéros globaux des sommets des éléments (et numéro du domaine)
  1  4  5  1  1
  2  2  1  5  1
  3  5  6  2  1
  4  3  2  6  1
  5  7  8  4  1
  6  5  4  8  1
  7  8  9  5  1
  8  6  5  9  1
#
# Nombre d'arêtes-frontières
  8
#
# Numéros globaux et extrémités des arêtes-frontières
  1   1  4
  2   4  7
  3   7  8
  4   8  9
  5   9  6
  6   6  3
  7   3  2
  8   2  1
```

Le fichier .dat

Le fichier commence par le nombre de domaines différents qui composent le maillage puis suivent les coefficients de conductivité λ_1 et λ_2 de chaque domaine.

Les domaines-frontières sont alors définis :

Le nombre de différents domaines-frontières est d'abord écrit.

Pour chaque domaine, suivent alors deux nombres : le nombre d'arêtes qui le composent puis le type de conditions aux limites (1 pour Dirichlet, 2 pour Neumann et 3 pour Robin).

Dans les lignes qui suivent, on peut lire les numéros globaux des arêtes-frontières qui composent chaque domaine avec, pour chaque arête, la ou les valeur(s) des conditions aux limites correspondantes :

- Deux valeurs doivent être fournies pour les conditions aux limites de Dirichlet : les températures aux deux extrémités du segment.
- En ce qui concerne les conditions aux limites de Neumann, une seule valeur est nécessaire : le flux de chaleur à travers l'arête $\left(\frac{\partial u}{\partial N} = a \right)$.
- Pour les conditions aux limites de Robin, on doit entrer les deux valeurs a et b qui caractérisent le transfert $\left(\frac{\partial u}{\partial N} = a \left[b - u(x) \right] \right)$.

Les dernières informations du fichier concernent la fonction *f*. Elle peut être différente pour chaque domaine et doit être écrite entièrement (par exemple : 2*x+tan(y)+7). Les fonctions qui peuvent être utilisées pour les définir sont les suivantes : tan(), sin(), cos(), exp(), ln(), sqrt() et la constante π.

L'exemple suivant est le fichier .dat qui correspond au fichier .net précédent :

```
# FEDemoJava – fichier .dat
#
# Nombre de domaines
  2
# Lambda1 et Lambda 2 dans le domaine 1
  200.0  200.0
# Lambda1 et Lambda 2 dans le domaine 2
  100.0  100.0
#
# Nombre de Domaines-frontières
  3
# Nb d'arêtes et type de conditions aux limites dans le domaine-frontière 1
  4  1
# Nb d'arêtes et type de conditions aux limites dans le domaine-frontière 2
  2  2
# Nb d'arêtes et type de conditions aux limites dans le domaine-frontière 3
  2  3
#
# N° globaux et données (coeff a (et b)) des arêtes du domaine-frontière 1
  1   0.   10.
  2  10.   20.
  3  20.   30.
  4  30.   40.
# N° globaux et données (coeff a (et b)) des arêtes du domaine-frontière 2
  5   5.
  6   7.
# N° globaux et données (coeff a (et b)) des arêtes du domaine-frontière 3
  7  10.   50.
  8  10.   50.
#
# Fonction F(x,y) dans le domaine 1
  3*cos(x*y*Pi)
# Fonction F(x,y) dans le domaine 2
  0
```