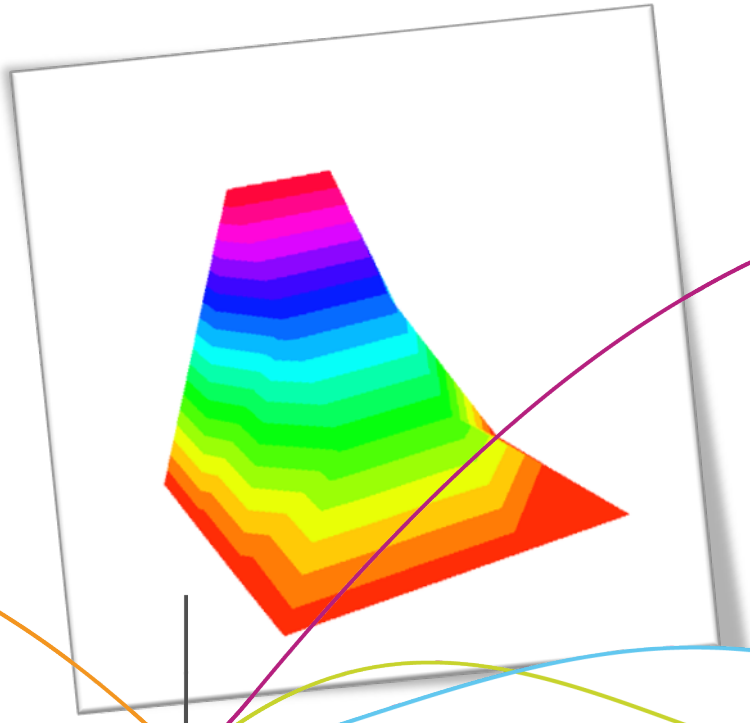
INSA ROUEN
À taille humaine,
à l'échelle du monde

Projet de Physique P6
STPI/P6/2014-nº 56

# Etude et détermination d'une surface minimale




Enseignant responsable
Nicolas HECHT

Étudiants :
- Laurent DEBRIL
- Da Chen LI
- Xin LIANG
- Hélène LUU
-Fatime-Zahra MEDOUAR

**Date de remise du rapport :** 16/06/14

**Référence du projet :** STPI[1]/P6/2014 – n° 56

**Intitulé du projet :** Etude et détermination d'une surface minimale

**Type de projet :** *modélisation et bibliographique*

**Objectifs du projet :**

L'objectif du projet est de déterminer la surface minimale de courbes fermées de $\mathbb{R}^3$ et de les visualiser grâce aux logiciels FreeFem++, qui permet de définir différents types de maillages et d'OpenGL, qui sert à la visualisation des surfaces. Pour cela, des méthodes d'analyse numérique ont été étudiées et sont implémentées, soit par défaut, soit par notre encadrant M. Hecht, dans ces logiciels. Par exemple, FreeFem++ calcule les surfaces minimales à l'aide de la méthode du gradient conjugué à pas fixe.

1. INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE ROUEN
DÉPARTEMENT SCIENCES ET TECHNIQUES POUR L'INGÉNIEUR
685 AVENUE DE L'UNIVERSITÉ BP 08- 76801 SAINT-ETIENNE-DU-ROUVRAY
TÉL : 33 2 32 95 66 21 - FAX : 33 2 32 95 66 31

# Table des matières

# Introduction

Dans le cadre de l'EC P6 « Projet de Physique », une liste d'une cinquantaine de sujets était proposée, parmi laquelle nous devions sélectionner cinq sujets potentiels. Par intérêt pour les mathématiques appliquées à un problème concret de physique, nous nous sommes retrouvés dans le même groupe avec le projet suivant :"Etude et détermination d'une surface minimale". Contrairement aux autres projets que nous étions amenés à entreprendre au cours de notre année de STPI 2, nous avons dû mettre en place une dynamique de groupe avec de nouveaux collègues que nous ne connaissions pas, situation courante dans la vie professionnelle d'un ingénieur. La mise en place de l'organisation et de moyens d'échange entre les différents éléments du groupe a donc été primordiale.

A cela s'ajoute le sujet lui-même, complexe et nécessitant de nombreuses recherches bibliographiques. L'étude des surfaces minimales est en effet un domaine relatif aux mathématiques et à la physique nécessitant de poser diverses notions de géométrie différentielle et de calcul numérique.

Mais qu'est-ce qu'une surface minimale ? Il s'agit d'une surface, dont l'ensemble de points constituant le bord est fixe, et qui minimise son aire. Mise en évidence en 1744 par Leonhard EULER, la caténoïde constitue une des trois surfaces minimales élémentaires, avec l'hélicoïde et le plan.

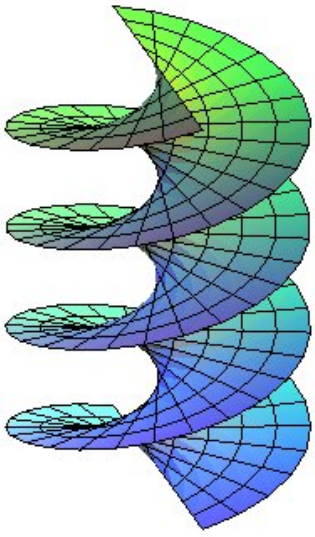


FIGURE 1 – Hélicoïde

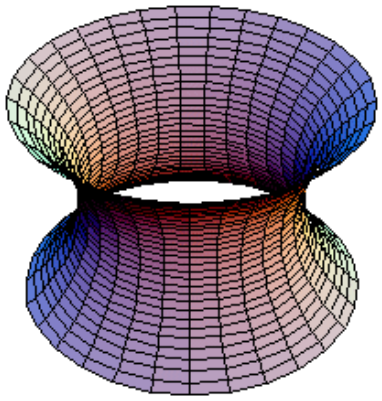


FIGURE 2 – Caténoïde

L'étude des surfaces minimales présente un intérêt, particulièrement dans le domaine de la construction, permettant ainsi de réduire les coûts en matériaux, avec comme exemples

les plus connus les colonnes de refroidissement des centrales nucléaires, ou encore le stade olympique de Munich (Allemagne).

FIGURE 3 – Colonnes de refroidissement des centrales nucléaires

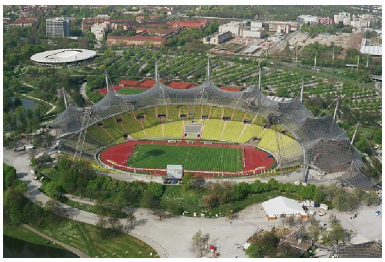


FIGURE 4 – Stade olympique de Munich

Au cours de la première moitié du semestre, notre objectif a été d'appréhender une vision globale du sujet et de pouvoir mieux comprendre les notions mathématiques nécessaires au sujet, avant d'avoir une application et une visualition plus concrète du problème.

# Chapitre 1

# Méthodologie, organisation du travail

Lors des premières séances, l'encadrant nous a introduit le sujet du projet, à savoir son objectif, les outils à notre disposition ainsi que le déroulement des différentes séances. Pendant près de la moitié du semestre, le projet consistait à faire des recherches bibliographiques, car même si l'appellation du sujet paraissait simple et compréhensible, dans le sens : "'on comprend tous les termes du sujet"', elle était tout aussi trompeuse. En effet, nous n'avions pas idée que cela ferait appel aux notions de "'maillage"', de "'discrétisation"', qui nous étaient jusque là inconnues, mais essentielles pour poursuivre l'étude de ce sujet. Les recherches durant les séances de TP étaient communes, i.e. nous partagions nos idées, nos explications entre nous mais aussi avec M. Hecht qui nous apportaient des explications et des précisions à nos interrogations.

Cependant, les approches du problème ont été différentes de celles prévues par notre encadrant. En effet, il était question de programmation d'une grosse partie du sujet en C++, car la plupart des framework sur l'étude des surfaces minimales, dont FreeFem++, était basée sur ce langage. Cette idée fut assez rapidement laissé de côté car aucun membre du groupe ne connaissait réellement ce langage, ni sa syntaxe. Par la suite, nous avons décidé de réorienter la modélisation en utilisant le code implementé par M Hecht dans FreeFem++ et faire des applications plus concrètes.


FIGURE 1.1 – Répartition des tâches réalisées

# Chapitre 2

# Travail réalisé et résultats

## 2.1 Maillage

Pour les besoins de notre projet nous cherchions à dessiner et à calculer l'aire d'une surface 2-D grâce au logiciel free-fem ++. On utilise le principe d'échantillonnage de la surface ou du maillage pour former et calculer l'aire d'une surface le plus rapidement possible.

### 2.1.1 Définition

Un maillage est donc un moyen de décrire une surface continue avec des éléments finis. Il existe differents types de maillage dépendants des formes utilisées et de la dimension. On se concentrera sur la 2-D où l'on retrouve des maillages triangulaire et quadrilatéral. Le maillage triangulaire, étant plus facile à manipuler et permettant la formation de surfaces complexes, est celui que nous utiliserons. Le maillage est composé de différentes parties : la Face qui est la frontière d'une cellule, un edge qui est la frontière d'une face, un Node un point de maillage et une zone un groupe de noeuds, face et/ou cell.

### 2.1.2 Maillage dans FreemFem++

Le logiciel FreeFemm++ contient des fonctions génératrices de maillage les "mesh generators". Un exemple de code qui permet de dessiner un maillage carré avec un maillage triangulaire de largeur 5 et de longueur 20 :

```
mesh Th=square(5,20)
plot(Th);
```

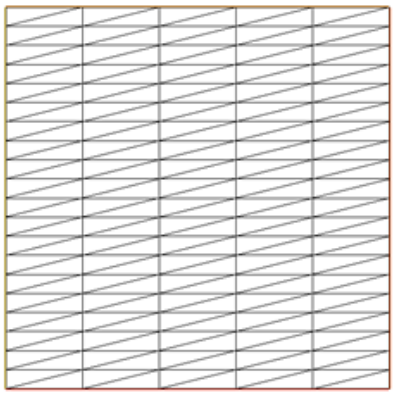


FIGURE 2.1 – Maillage d'un carré avec un motif triangulaire

Il est aussi possible de fixer des limites pour un maillage en utilisant la fonction 'border' définie par des courbes paramétrées.

### 2.1.3 Un type d'algorithme de générateur de maillage

Différents algorithmes générateurs de maillage triangulaire existent. Le plus simple est l'algorithme par avance de front : création d'un squelette de la forme qui est composé d'arêtes et de noeuds. Pour chaque arête, on cherche à former un triangle équilatéral qui nous donne la position idéale d'un point C, ensuite on prend l'arête voisine et on regarde si le point C rentre dans le triangle équilatéral composé avec la dernière arête. Si oui, on prend ce point là (cela peut être un noeud) on continue jusqu'a la dernière arête. Si plusieurs points peuvent servir de C, on prend celui qui nous fournit le triangle le plus proche d'un triangle équilatéral.

Grâce à la commande "borde", on peut donc créer des courbes fixes qui délimitent notre maillage. Pour cela on utilise des courbes paramétriques dépendant de t, par exemple un carré $20 \times 20$, s'écrit :

```
int lateral=1;
//permet de nommé les catégories des bordures
int base=2;
border C01 (t=0,20) {x=0; y=t; label=lateral;}
border C02 (t=0,20) {x=20; y=20-t; label=lateral;}
// border nomDuCôté (paramètre=bornInf.bornSup
{équation de la courbe paramétrique; label= nomCatég}
border C03 (t=0,20) {x=20-t; y=0;label=base;}
border C04 (t=0,20) {x=t; y=20;label=base;}

int n = 10;
//nombre de découpage du maillage par côté, précision des maillages
mesh Th = buildmesh(C01(-n)+C02(-n)+C03(-n)+C04(-n));
//le signe du n donne le sens, il faut impérativement que le sens
soit trigonométrique
plot(Th, wait=true);
```

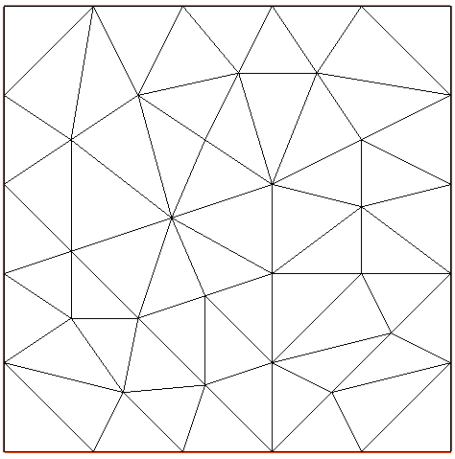


FIGURE 2.2 – Maillage border n=5

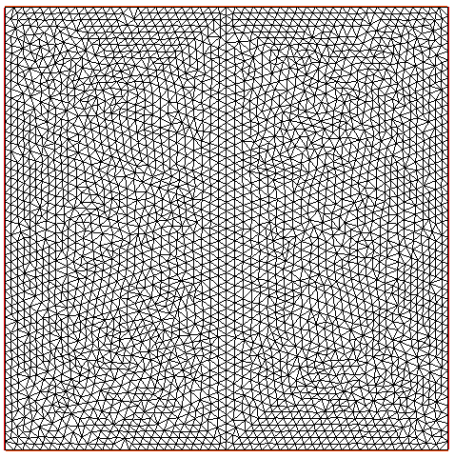


FIGURE 2.3 – Maillage border n=50

## 2.2 Méthodes de résolution de systèmes linéaires

### 2.2.1 Cadre

Considérons l'équation que l'on souhaite résoudre :

$$Ax = b$$

avec $A$ une matrice inversible de taille $N \times N$, $b$ un vecteur donné et $x$ le vecteur inconnu de taille $N$.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

### 2.2.2 Méthodes directes

#### 2.2.2.1 Méthode de Gauss

La méthode de Gauss est une méthode pour transformer une matrice en une autre matrice, équivalente, qui est triangulaire supérieure et que l'on sait résoudre par la méthode de la remontée.

En théorie, on effectue une succession d'opérations élémentaires au système pour transformer $A = A^{(1)}$ dont la $1^{re}$ colonne n'est pas nulle, en une matrice $A^{(2)}$, dont la $1^{re}$ colonne a tous ses coefficients nuls sauf le premier. Puis par une succession d'opérations élémentaires, la matrice peut être transformée en une matrice échelonnée.

Les opérations élémentaires autorisées sur le système, sont de deux types :

$$L_i \longleftrightarrow L_j (i \neq j)$$

$$L_i \underset{\text{remplacée par}}{\longleftarrow} \alpha L_i + \beta L_j (i \neq j, \alpha \neq 0)$$

Pour cela, on part de notre matrice $A = A^{(1)}$et $b = b^{(1)}$ puis on introduit un multiplicateur $c_i^1 = \frac{a_{i,1}}{a_{1,1}}$. On peut ainsi éliminer l'inconnue $x_1$ des lignes $i = 2...N$ en leur soustrayant $c_i^1$ fois la première ligne. Et bien évidemment, le vecteur $b$ suit les mêmes transformations. On obtient alors une matrice équivalente $A^{(2)}$et un nouveau $b^{(2)}$.

$$\begin{cases} a_{i,j}^{(2)} &= a_{i,j} - c_i^1 a_{1,j}^{(1)} \text{ , pour } i,j = 1...N \\ b_{i,j}^{(2)} &= b_{i,j} - c_i^1 b_1^{(1)} \end{cases}$$

Ce que l'on souhaite résoudre maintenant, c'est $A^{(2)}x = b^{(2)}$.

$$\begin{pmatrix} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,n}^{(1)} \\ 0 & a_{2,2}^{(2)} & \cdots & a_{2,n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n,2}^{(2)} & \cdots & a_{nn}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{pmatrix}$$

Suivant le même principe, on peut éliminer $x_2$ des lignes $i = 3...N$, en les soustrayant $c_i^2$ fois la deuxième ligne, avec $c_i^2 = \frac{a_{i,2}}{a_{2,2}}$, puis $x_3$ des lignes $i = 4...N$, en les soustrayant $c_i^3$ fois la troisième ligne, avec $c_i^3 = \frac{a_{i,3}}{a_{3,3}}$ et ainsi de suite en posant successivement $c_i^k = \frac{a_{i,k}}{a_{k,k}}$ pour $k = 1...N-1$ puis en soustrayant $c_i^k$ fois la $k^{ime}$ ligne pour supprimer $x_k$ des lignes $i = k+1...N$, jusqu'à obtenir un système triangulaire supérieur équivalent, i.e. jusqu'à avoir $A^{(n)}x = b^{(n)}$.

$$\begin{pmatrix} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,n-1}^{(1)} & a_{1,n}^{(1)} \\ 0 & a_{2,2}^{(2)} & \cdots & a_{2,n-1}^{(2)} & a_{2,n}^{(2)} \\ 0 & 0 & a_{3,3}^{(3)} & \cdots & a_{3,n}^{(3)} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & a_{n_n}^{(n)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \\ \vdots \\ b_n^{(n)} \end{pmatrix}$$

Et lorsqu'on arrive à ce sytème, il suffit simplement d'utiliser la méthode de la remontée, introduite précedemment, pour le résoudre.

Cependant, pour que cette méthode soit applicable, il est nécessaire que les pivots $a_{k,k}^{(k)}$pour $k = 1...n-1$ soient non nuls. Si ce n'est pas le cas, il suffit de s'y ramener en effectuant des permutations de lignes ou de colonnes.

#### 2.2.2.2 Factorisation de LU

La factorisation de $LU$ (Lower Upper) de la matrice A inversible, consiste à décomposer $A$ en deux matrices triangulaires tel que

$$A = L \times U$$

avec $L$, une matrice triangulaire inférieure, dont les coefficients diagonaux sont égaux à 1 et $U$, une matrice triangulaire supérieure.

Par ailleurs, en s'intéressant de plus près à cette méthode, il s'avère que cette décomposition n'existe pas toujours. Voici la proposition qui permet de savoir si une matrice possède ou non une décomposition LU.

**Définition.** *Etant données une matrice $A = (a_{i,j})_{\substack{1\leq i\leq n \\ 1\leq j\leq n}} \in M_n(\mathbb{K})$ ainsi que des entiers $i$ et $j$ de $[\![1,n]\!]$, on appelle mineur de $a_{i,j}$ le déterminant de la matrice extraite de $A$ obtenue en supprimant la $i^{ime}$ ligne et la $j^{ime}$ colonne de $A$. Et le mineur est dit principal si $i = j$.*

**Proposition.** *Une matrice $A$ de $GLn(\mathbb{K})$ possède une décomposition $LU$ si et seulement si ses mineurs principaux sont non nuls. Cette décomposition est alors unique.*

La matrice $U$ s'obtient par la méthode de Gauss. Et la matrice $L$ reprend les pivots successifs de l'algorithme de Gauss $c_i^k$ :

$$U = \begin{pmatrix} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,n-1}^{(1)} & a_{1,n}^{(1)} \\ 0 & a_{2,2}^{(2)} & \cdots & a_{2,n-1}^{(2)} & a_{2,n}^{(2)} \\ 0 & 0 & a_{3,3}^{(3)} & \cdots & a_{3,n}^{(3)} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & a_{n,n}^{(n)} \end{pmatrix} \text{ et } L = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ c_2^1 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ c_{n-1}^1 & c_{n-1}^2 & \ddots & 1 & 0 \\ c_n^1 & c_n^2 & \cdots & c_n^{n-1} & 1 \end{pmatrix}$$

A présent, nous avons $A = L \times U$. Donc, pour résoudre le système

$$Ax = b \Leftrightarrow LUx = b$$

on pose $Ux = Y$, puis on résout successivement par la méthode de la remontée (pour U) et de la descente (pour L) :

$$\begin{cases} LY &= b \\ Ux &= Y \end{cases}$$

### 2.2.3 Méthodes itératives

#### 2.2.3.1 Méthode du gradient à pas fixe

Cette méthode de résolution concerne les matrices symétriques définies positives.
Elle consiste à minimiser la fonctionnelle $J : \mathbf{R}^n \to \mathbf{R}$, tel que,

$$J(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$$

avec $\langle \bullet, \bullet \rangle$, le produit scalaire euclidien usuel.

Cependant, on peut montrer le théorème suivant :

**Théorème.** *Chercher le minimum de la fonction quadratique J équivaut à résoudre le système* $Ax = b$

En effet,

$$\begin{aligned} \nabla J &= \frac{1}{2}(\langle A, x\rangle + \langle Ax, 1\rangle) - \langle b, 1\rangle \\ &= \frac{1}{2}(Ax + Ax) - b \\ &= Ax - b \end{aligned}$$

Le minimum est obtenu en annulant le gradient de J. D'où le nom de méthode de gradient.
Voici l'algorithme.

- $x^0 \in \mathbf{R}^n$ donné,
- pour k=0,1,2, ... faire

$$x^{k+1} = x^k - \alpha J'(x^k)$$

  avec $\alpha$ un paramètre à fixer.
- jusqu'à convergence.

Pour déterminer $\alpha$, rappelons tout d'abord le théorème spectral.

**Théorème** (spectral)**.** *Une matrice symétrique définie positive est diagonalisable sur une base de vecteurs propres associés à leurs valeurs propres réels* $(u_n, \lambda_n) \in \mathbf{R}^n \times \mathbf{R}_+^*$, *avec*

$$0 < \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$$

On peut alors trouver :

**Proposition.** *La méthode du gradient à pas fixe converge si et seulement si* $0 < \alpha < \frac{2}{\lambda_n}$. *Sa valeur optimale est* $\alpha_{opt} = \frac{2}{\lambda_1 + \lambda_n}$

#### 2.2.3.2 Méthode du gradient conjugué préconditionné

Le principe de cette méthode consiste à résoudre, non pas, $Ax = b$, mais $C^{-1}Ax = C^{-1}b$ avec $C^{-1}$, une matrice de préconditionnemenet bien choisie. Cette méthode est très efficace pour les matrices symétriques définies positives et creuses.

Voici l'algorithme.
- On initialise. On choisit un $x^{(0)}$ arbitraire, s'il n'est pas donné, de résidu $R^{(0)} = b - Ax^{(0)}$. Le vecteur $P^{(0)}$ est obtenu en résolvant $CP^{(0)} = R^{(0)}$. Puis on introduit le vecteur $Z^{(0)} = R^{(0)}$.
- On itère pour k variant de 0 à n-1

$$\begin{aligned}
\alpha_k &= \frac{\langle R^{(k)}, Z^{(k)} \rangle}{\langle AP^{(k)}, P^{(k)}} && \leftarrow \text{pas de descente optimal} \\
X^{(k+1)} &= X^{(k)} + \alpha_k P^{(k)} && \leftarrow \text{calcul itératif du gradient} \\
R^{(k+1)} &= R^{(k)} - \alpha_k AP^{(k)} \\
CZ^{(k+1)} &= R^{(k+1)} \\
\beta_{k+1} &= \frac{\langle R^{(k+1)}, Z^{(k+1)} \rangle}{\langle R^{(k)}, Z^{(k)} \rangle} \\
P^{(k+1)} &= Z^{(k+1)} + \beta_{k+1} P^{(k)}
\end{aligned}$$

- A chaque itération, on résout le système linéaire $CZ = R$.

## 2.3 Application FreemFem++

FreeFem++ est un logiciel libre écrit en C++, développé au Laboratoire de l'Université Pierre et Marie Curie. La syntaxe a été expliquée à l'annexe A.

### 2.3.1 Maillage

#### 2.3.1.1 Maillage triangulaire régulier dans un domaine rectangulaire

Soit le domaine $]x_0, x_1[\times]y_0, y_1[$. La génération d'un maillage régulier de taille $n \times m$ se fait par :

```
mesh nom_maillage = square(n,m,[x0+(x1-x0)*x,y0+(y1-y0)*y]);
```

#### 2.3.1.2 Maillage triangulaire non structué défini à partir de ses frontières

Pour définir les frontières, il existe une commande "'border"', utilisée comme suit :

```
border name(t=deb,fin){x=x(t);y=y(t);label=num_label};
```

Pour définir un maillage à partir de ses frontières, nous utilisons "'buildmesh"' :

```
buildmesh nom_maillage= buildmesh(a1(n1)+a2(n2)+...+ak(nk));
```

#### 2.3.1.3 Exemples

(1) Un domaine carré : $]0,1[^2$ (donné par M.HECHT)

```
mesh Th= square(10,10,[x,y]);// Carre ]0,1[^2
plot(Th,wait=1);
```

(2) Un domaine sous forme L : $]0,1[^2 \times [0.5,1[^2$

```
border a(t=0,1.0){x=t ; y=0 ; label=1 ;} ;
border b(t=0,0.5){x=1 ; y=t ; label=2 ;} ;
border c(t=0,0.5){x=1-t ; y=0.5 ;label=3 ;} ;
border d(t=0.5,1){x=0.5 ; y=t ; label=4 ;} ;
border e(t=0.5,1){x=1-t ; y=1 ; label=5 ;} ;
border f(t=0.0,1){x=0 ; y=1-t ;label=6 ;} ;
plot(a(6) + b(4) + c(4) +d(4) + e(4) + f(6),wait=1) ;
// pour voir les 6 borders
mesh Th2 = buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) + f(6)) ;
```

(3) Lire un maillage externe

```
mesh Th("Th.msh ") ;
```

(4) Entrées / sorties fichiers

```
savemesh(nom_maillage,nom_fichier) ; // sauver le maillage (.msh)
readmesh(nom_fichier) ; // lire un maillage à partir d'un fichier
```

(5) Autres fonctions sur les maillage

```
mesh mail2 = movemesh(mail1,[f1(x,y),f2(x,y)]) ;
// déformer le maillage
mesh mail2 = adaptmesh(mail1,var) ;
// raffiner le maillage dans les zones de forte variations
```

(6) Lire les données d'un maillage

```
Th.nt (nbre de triangle), Th.nv (nbre de noeuds),
Th[i][j] (sommet j du triangle i), ...
```

#### 2.3.1.4 Visualisation des résultats

Avec Freefem++, l'affichage des maillages, des courbes d'isovaleurs et des champs de vecteurs peut se faire de la manière suivante :

```
plot(var1,[var2,var3],...[liste d'options]) ;
wait=true/false, value=true/false, fill=num, ps="nom_fichier",...
```

Des commandes existent pour exporter ces données vers d'autres logiciels Gnuplot, Medit,...

```
{ ofstream file("exemple.bb") ;
// file pour medit
file <<"2 1"<<uh[].n<<" 2"<<endl ;
for (int i=0 ;i<uh[].n ;i++){
file << uh[][j] << endl ;
```

```
}}
// d'appeler la commande gnuplot et attendre 5 secondes
// (grâce à unix command)
// et plot postscipt
exec("echo 'plot \"plot.gp\" using 1:2 w l pause 5 set term
postscript set output \"gnuplot.eps\" replot quit' | gnuplot") ;
```

### 2.3.2 Application : la caténoïde

#### 2.3.2.1 A partir d'un maillage carré

Tout d'abord, il faut créer le maillage sur lequel notre caténoïde va reposer.

```
mesh Th= square(10,10,[x,y]); // Carre ]0,1[2
plot(Th,wait=1);
savemesh(Th,"Th.msh");
fespace Vh(Th,P1);
Vh u,du;
```

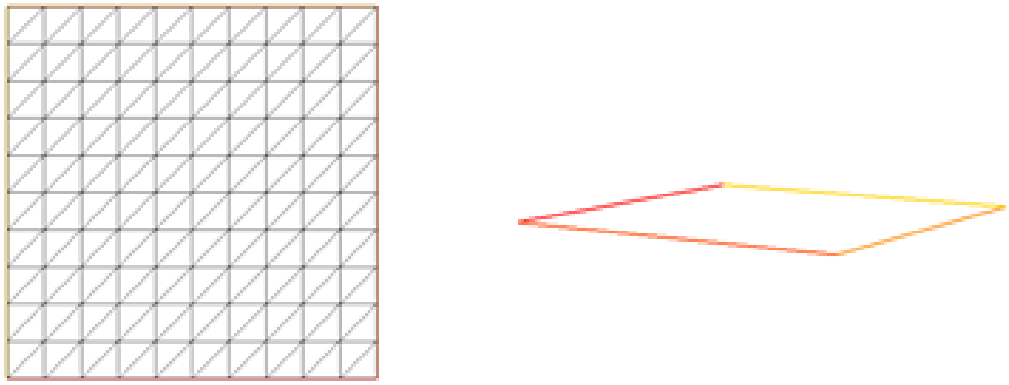


FIGURE 2.4 – Aperçu du maillage carré sur FreeFrem++

La caténoïde se dessine à partir d'un demi-cercle dans le plan vertical du maillage, dont son équation s'obtient à partir de celle d'un cercle de centre $(0.5, 0)$ et de rayon $0.5$ ("0.5" pour le placer au centre du maillage) :

$$(x - 0.5)^2 + y^2 = 0.5^2 \iff y = \sqrt{0.5^2 - (x - 0.5)^2}$$

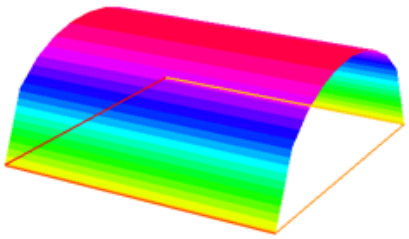


FIGURE 2.5 – Avant minimisation

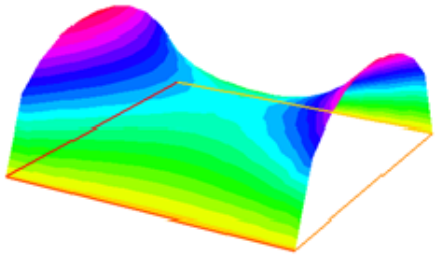


FIGURE 2.6 – Après minimisation

Comme nous pouvons le voir, après minimisation d'un demi-cercle sur un maillage à bords fixes, nous obtenons une demi-caténoïde.

#### 2.3.2.2 A partir d'un maillage circulaire

Il est également possible de travailler avec un maillage circulaire. Il suffit pour cela de paramétriser l'équation du cercle précédent :

$$\text{Pour } t \in \mathbb{R}, x = cos(t)$$
$$y = sin(t)$$

```
border a(t=0,2*pi){ x=cos(t); y=sin(t);};
mesh Th = buildmesh(a(50));
plot (Th,wait=1,ps="nosplitmesh.eps");
```

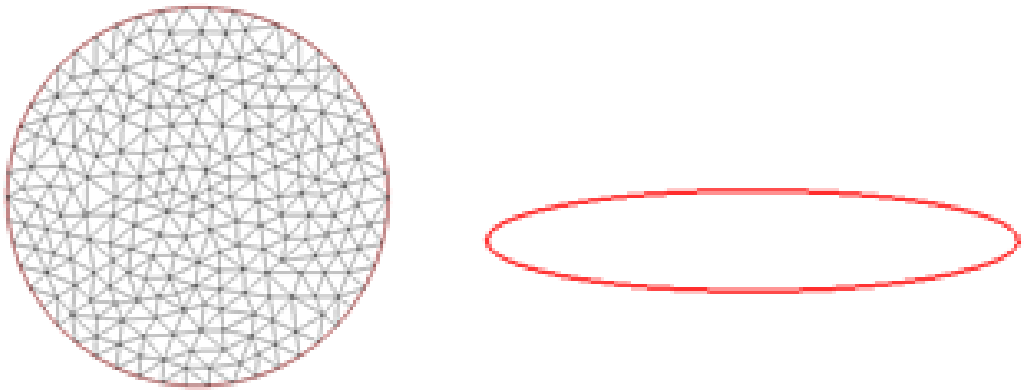


FIGURE 2.7 – Aperçu du maillage circulaire sur FreeFrem++

Puis nous pouvons créer un maillage en forme anneau avec 2 équations paramétriques de cercles de rayons différents. Par exemple, nous pouvons prendre un cercle de rayon $1$ et un autre de rayon $0.5$.

```
border a(t=0,2*pi){ x=cos(t); y=sin(t);};
border b(t=0,2*pi){ x=cos(t)/2; y=sin(t)/2;};
mesh Th = buildmesh(a(50)+b(-50));
//attention: on ne fait pas a(50)-b(50) ici
plot (Th,wait=1,ps="nosplitmesh.eps");
savemesh(Th,"Th.msh");
fespace Vh(Th,P1);
Vh u,du;
```

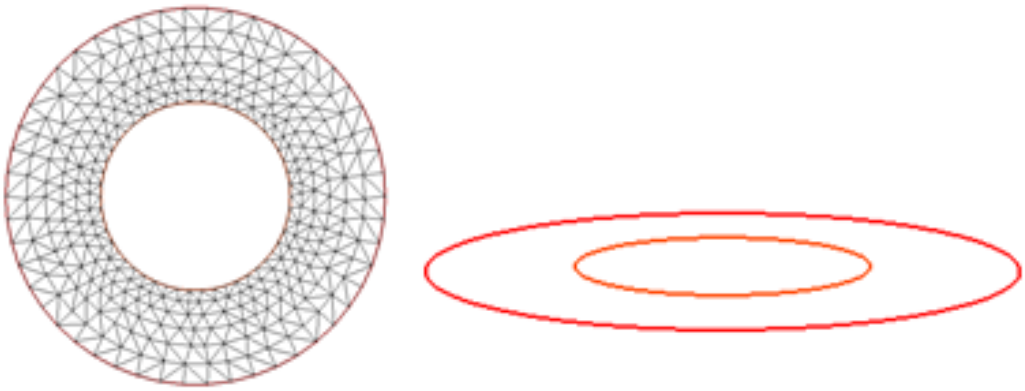


FIGURE 2.8 – Aperçu du maillage en forme anneau sur FreeFrem++

Et pour obtenir la caténoïde, nous utilisons l'équation : $u = x^2 + y^2$

```
u = x*x+y*y;
plot(u,wait=1,dim=3,fill=1);
```

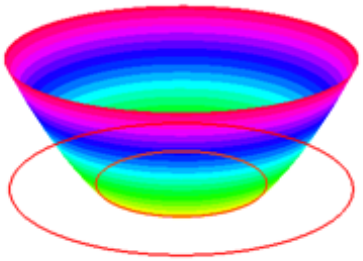


FIGURE 2.9 – Avant minimisation

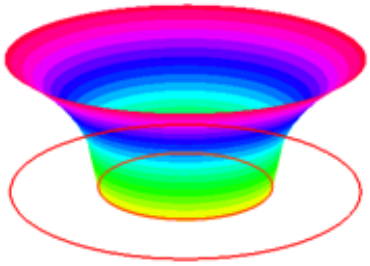


FIGURE 2.10 – Après minimisation

Ainsi, si nous minimisons un cône tronqué en fixant les bords du haut et du bas, nous obtenons une semi-caténoïde.

D'autres visualisations sont données en annexe.

## 2.4 Analyse de la performance des algorithmes

### 2.4.1 Comparaison entre les différents algorithmes de résolution

Au cours de ce projet, nous avons étudié deux programmes permettant de minimiser des surfaces : le logiciel FreeFem++ et un programme plus modulaire codé en C++. Afin de déterminer quel algorithme utiliser en fonction des situations, nous avions pour idée de déterminer la même surface minimale avec chacun des algorithmes, et ensuite comparer leur temps d'exécution, ainsi que leur précision respective. Pour cela, nous avions implémenté une fonction TimeSaveFile.cpp qui stockait dans des fichiers .txt les temps d'exécution pour différentes surfaces pour un algorithme particulier. Cependant, un problème s'est vite imposé : en effet, de par la manière dont chaque algorithme était codé, il était impossible d'induire une comparaison quelconque. En effet, au sein du programme codé en C++ *glplot.cp*, l'affichage graphique de la simulation est difficilement dissociable du calcul de minimisation lui-même. Les affichages graphiques nécessitant des ressources assez importantes, les résultats d'une comparaison auraient été biaisés.

Ainsi, au lieu de comparer nos deux méthodes de simulation, nous avons entrepris d'effectuer un test de performance situationnel et non comparatif : en fonction des simulations données et de paramètres variés, nous avons pu observer le comportement des méthodes informatiques.

### 2.4.2 Test de performance de FreeFem++

Dans un premier temps, nous avons étudié le temps d'exécution de FreeFem++ pour différents fonctions à deux variables (polynomiales et trigonométriques).

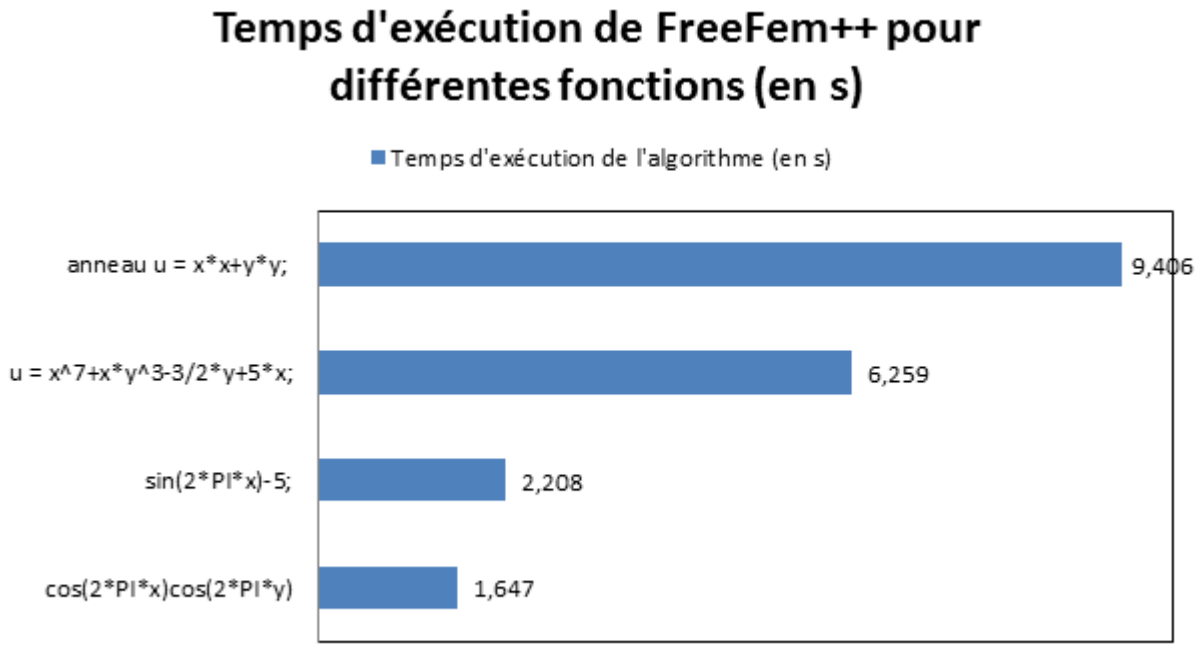



FIGURE 2.11 – Temps d'exécution de FreeFem++ pour différentes fonctions (en seconde)

Nous pouvons constater un temps d'exécution environ $5.7$ fois plus important pour la minimisation d'un anneau $u = x^2 + y^2$ que pour la fonction trigonométrique $cos(2\pi x)cos(2\pi y)$. Cependant, cela ne nous permet pas d'analyser un comportement quelconque du programme : le calcul des fonctions polynomiales est à l'origine beaucoup plus long et nécessite plus de ressources que les fonctions trigonométriques, de par la conception des processeurs d'ordinateurs. Nous pouvons juste constater les conséquences de ce phénomène dans la minimisation.

Dans un deuxième temps, nous nous sommes interrogés sur l'impact du nombre de bords fixés. Nous pouvons en effet intuiter que plus le nombre de bords fixés (et donc de contraintes sur le calcul) est important, et plus le temps d'exécution sera long. La deuxième question était de savoir si des contraintes sur deux côtés adjacents avaient ou non, plus d'impact sur le temps d'exécution que sur deux côtés parallèles (qui induiraient une symétrie). Les résultats obtenus sont les suivants :

Nous pouvons constater que la première assertion était vraie : plus le nombre de bords fixés est élevé et plus le calcul nécessite de temps (environ $1.76$ fois plus de temps nécessaire entre une simulation avec un côté du maillage fixé, et une simulation avec les quatre bords du maillage fixés). Cependant, nous constatons la faible différence de temps entre côtés du maillage adjacents fixés et parallèles.

Puis, dans un dernier temps, nous voulions observer le comportement de FreeFem++ dans diverses plages de précision, toujours en matière de temps d'exécution.

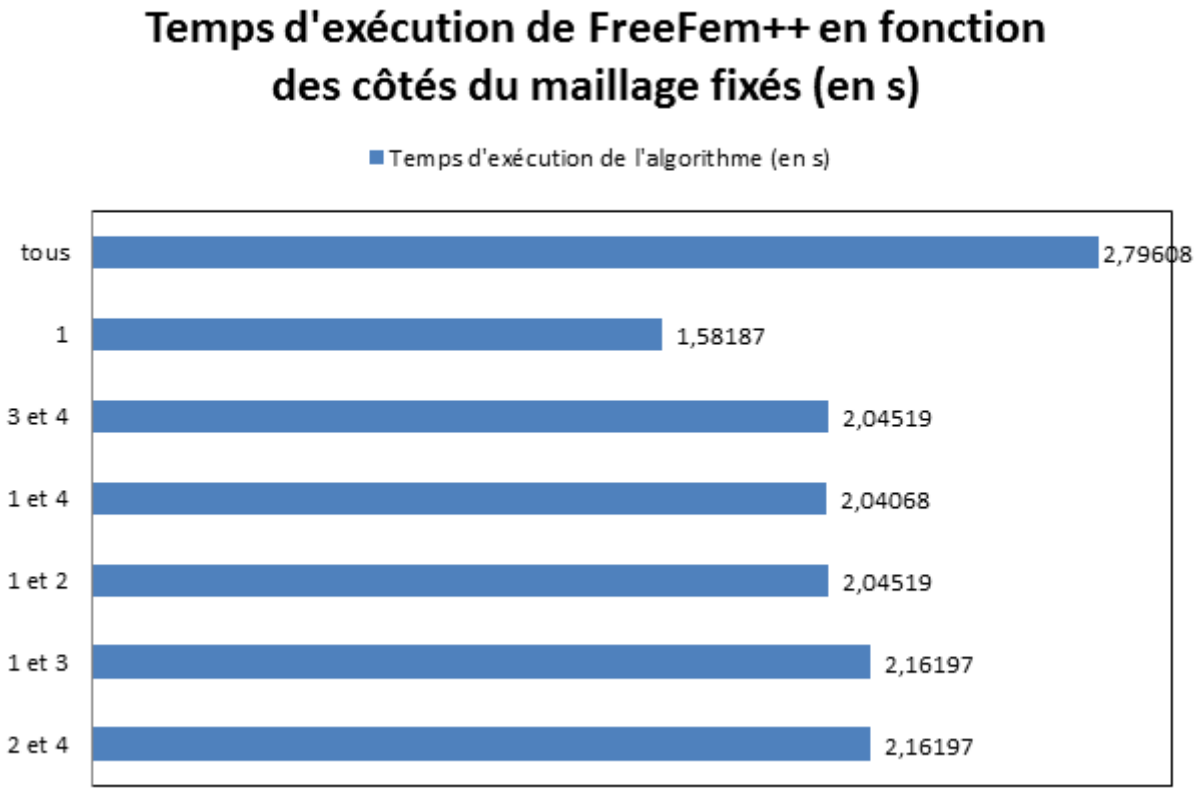



FIGURE 2.12 – Temps d'exécution de FreeFem++ en fonctions des côtés du maillage fixés (en seconde)

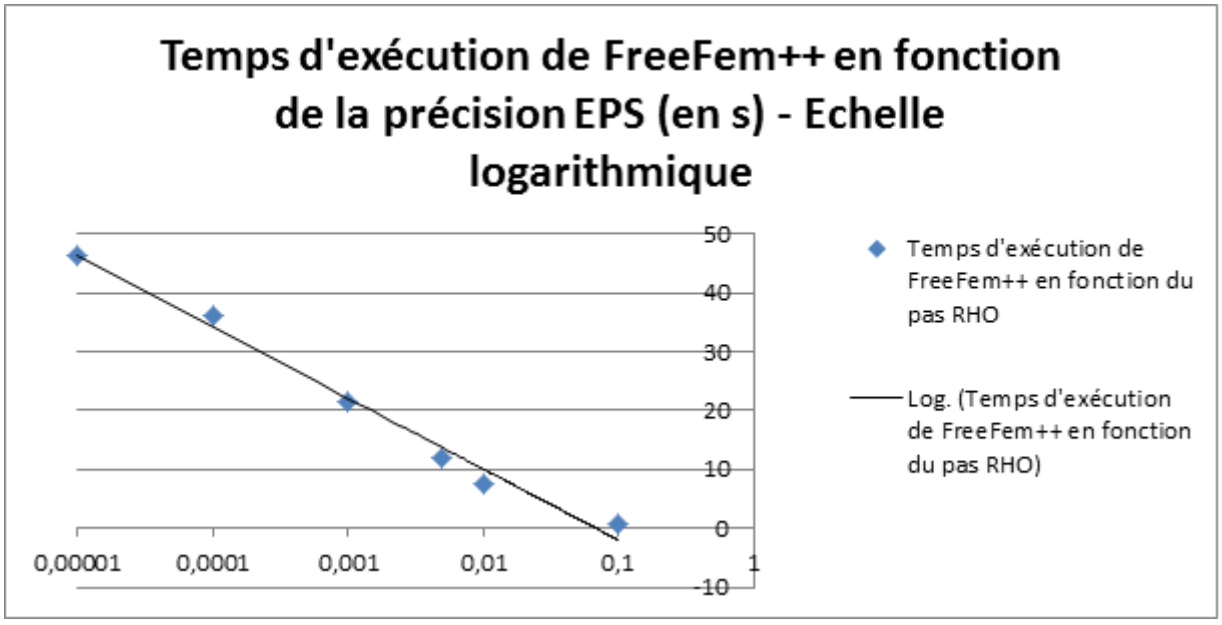



FIGURE 2.13 – Temps d'exécution de FreeFem++ en fonctions de la précision EPS (en seconde)- Echelle logarithmique

Pour une précision de l'ordre de $0.1$ (précision faible), la minimisation ainsi que l'affichage de la surface est quasi-instantanée. Il faut augmenter la précision vers un ordre de $0.005$ afin de commencer à observer la minimisation « en direct ». Ensuite, le temps d'exécution observe une dépendance exponentielle par rapport à la précision, mais l'on peut constater un

temps toujours inférieur à $50$ secondes pour une précision élevée de l'ordre de $10^{-5}$. FreeFem++ constitue donc une solution fiable et dont les temps d'exécution restent parfaitement adaptés aux problèmes posés dans ce projet.

### 2.4.3 Test de performance de *glplot.cpp*

* Pour l'ensemble des tests effectués sur cette méthode, nous avons choisi d'utiliser quatre fonctions numérotées de 1 à 4 dans les graphiques suivants :

- Fonction n° 1 : $f(x,y) = sin(M\pi x) \times cos(0.5 \cdot M\pi y) + \frac{y^3}{100}$ ;
- Fonction n° 2 : $f(x,y) = sin(M\pi x) + cos(0.5 \cdot M\pi y)$ ;
- Fonction n° 3 : $f(x,y) = (0.3x)x + (x-y)y$ ;
- Fonction n° 4 : $f(x,y) = \sqrt{0.5^2 - x^2}$ ;

* FreeFem++ utilisant la méthode de gradient à pas fixe, les tests effectués sur glplot.cpp seront basés sur la même méthode de résolution.

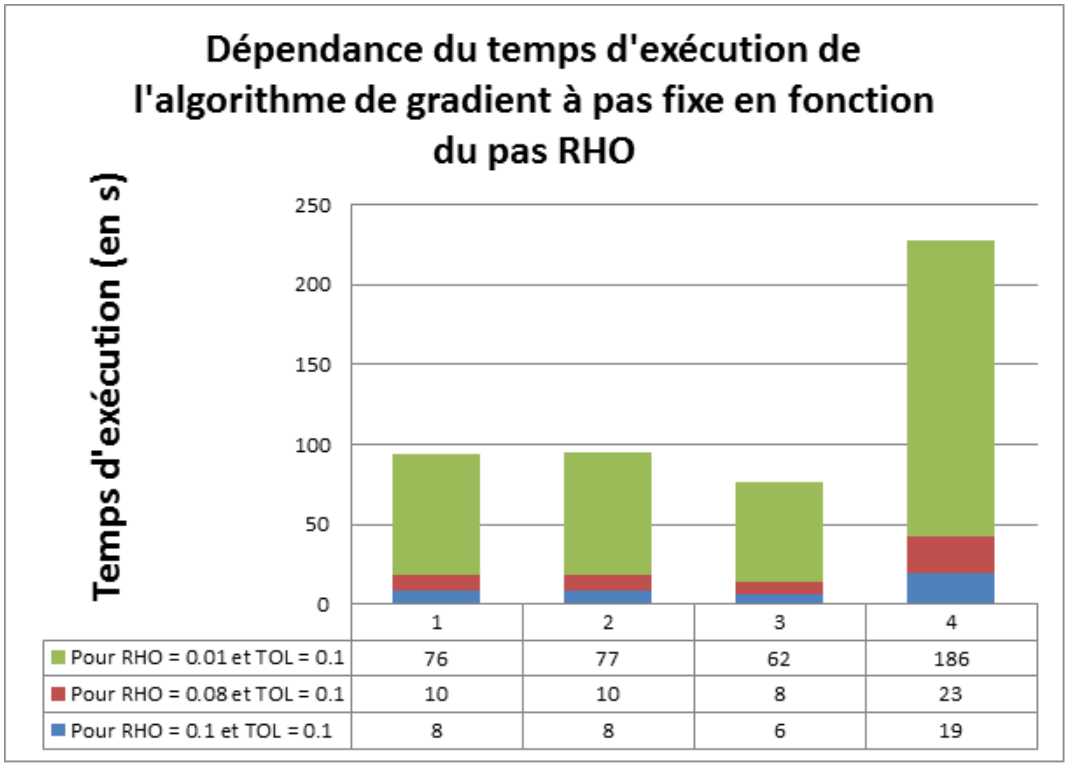



FIGURE 2.14 – Dépendance du temps d'exécution (en seconde) de l'algorithme de gradient à pas fixe en fonction du pas RHO

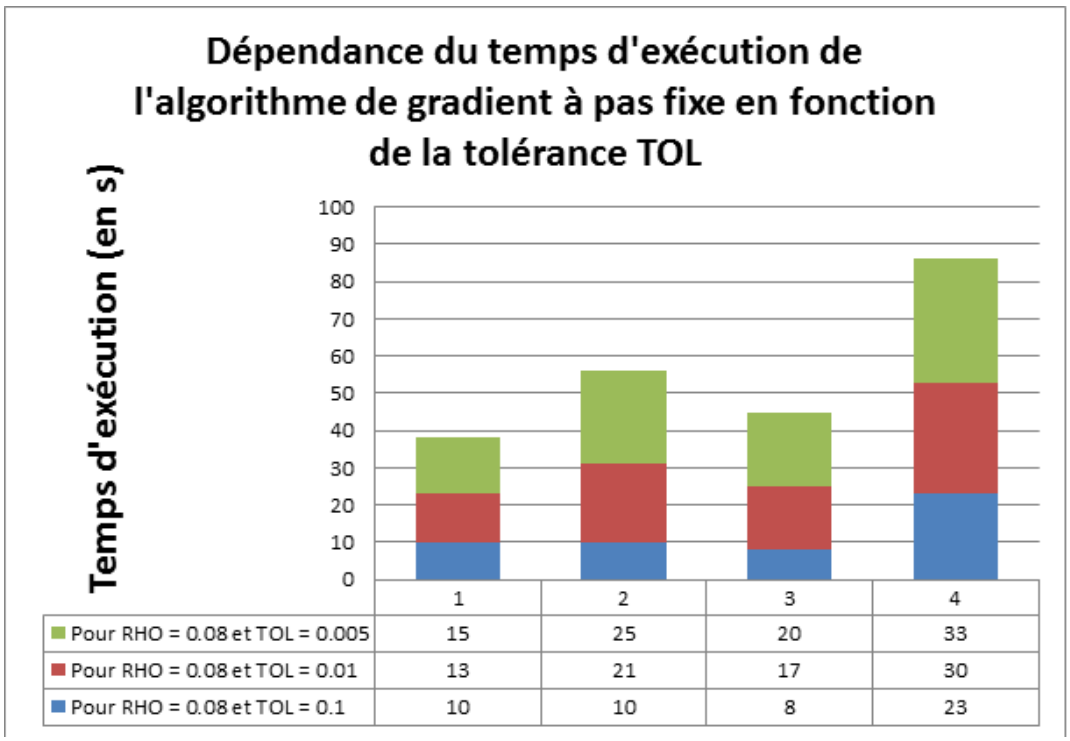



FIGURE 2.15 – Dépendance du temps d'exécution (en seconde) de l'algorithme de gradient à pas fixe en fonction de la tolérance TOL

Quel que soit le cas, glplot.cpp s'avère être un programme à l'exécution plus lente : cela est normal, étant donné que le logiciel FreeFem++ est un logiciel spécialement optimisé pour ce genre de calcul, tandis qu'à l'inverse le programme glplot.cpp s'axe sur la modularité, avec un nombre de méthodes de résolution plus élevé. On observe très rapidement une augmentation importante du temps d'exécution lorsque le pas RHO dépasse du $10^{-2}$. Les mesures s'avèrent extrêmement difficiles dès lors que RHO est inférieur à $10^{-3}$, avec des minimisations prenant plus de $5$ minutes chacune (là où FreeFem++ nécessite moins de trente secondes). L'augmentation de la précision TOL augmente progressivement le temps d'exécution, mais avec cependant moins d'impact que celui du pas RHO. Ces tests montrent bien que FreeFem++ constitue la méthode la plus optimisée.

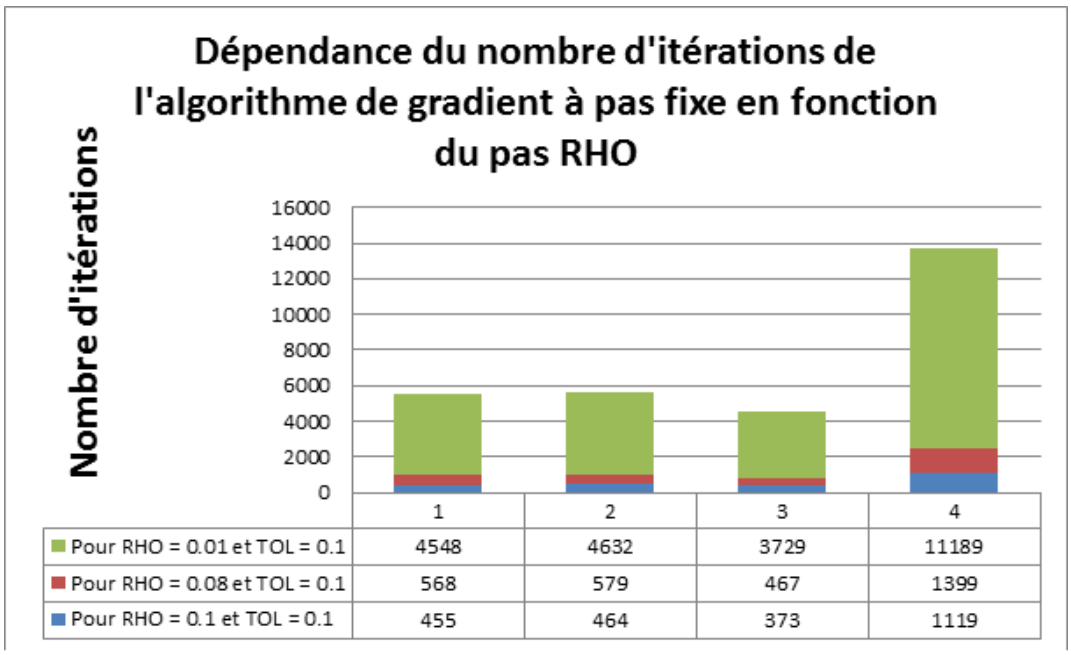



FIGURE 2.16 – Dépendance du nombre d'itérations de l'algorithme de gradient à pas fixe en fonction du pas RHO

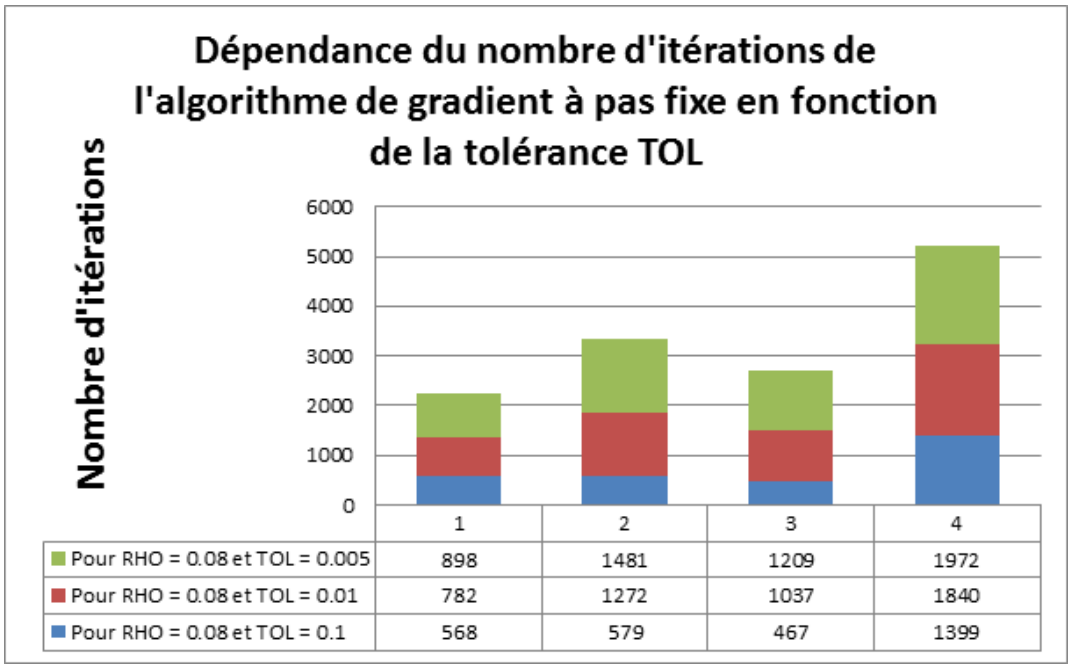



FIGURE 2.17 – Dépendance du nombre d'itérations de l'algorithme de gradient à pas fixe en fonction de la tolérance TOL

# Conclusion et perspectives

## 2.5 Conclusion générale

En définitive, ce projet nous a permis d'appréhender un domaine de la physique qui présente son utilité au quotidien, tout particulièrement dans l'architecture et le génie civil. Ce dernier, faisant appel à des notions mathématiques qui nous étaient auparavant inconnues, a nécessité une recherche bibliographique conséquente s'étalant sur la première moitié du semestre. Dans un deuxième temps, nous avons entrepris de nous séparer le travail, du fait de l'étendue du sujet, en fonction des compétences de chacun. Cette organisation basée sur la subdivision des tâches au sein d'une équipe constitue la base du travail de l'ingénieur, et constitue en soi une première expérience intéressante.

## 2.6 Conclusion personnelle

### 2.6.1 Laurent

Pour ma part, ce projet fut l'opportunité de travailler sur un nouveau langage de programmation : le C++, l'ensemble des programmes liés à la détermination de surfaces minimales étant basé sur ce dernier. De plus, une des principales difficultés rencontrées au cours de ce projet fut la complexité des notions mathématiques, mais également le temps alloué à cet EC. En effet, un créneau hebdomadaire d'une heure et demie a nécessité un travail personnel conséquent afin d'assimiler les notions. Cependant, cette contrainte m'a permis d'optimiser mon temps de travail, ainsi que mon autonomie. Un approfondissement possible du sujet à l'avenir serait l'exploitation d'autres logiciels éventuels de résolution d'équations aux dérivées partielles, afin de résoudre des problèmes de minimisation, ou encore, d'entreprendre un apprentissage complet du C++ afin d'essayer de créer notre propre générateur de maillage dans ce langage.

### 2.6.2 Hélène

Ce projet fut très enrichissant, car il était intéressant de pouvoir allier la physique avec les mathématiques. En effet, au cours du cursus scolaire, les deux domaines sont souvent dissocier, alors qu'il est évident que l'un ne va sans l'autre, bien que j'ai toujours un peu de mal à relier les deux parties. Durant une bonne partie du semestre, j'ai dû effectuer beaucoup de recherches pour comprendre les différents types de méthodes de résolution d'équations linéaires. Ce travail a nécessité énormément de temps personnel, mais cela m'a permis d'acquérir de l'autonomie dans la recherche. Et je suis convaincue que la résolution de systèmes linéaires est un sujet d'étude que je retrouverai en département.

Les perspectives de pousuites de ce projet sont nombreuses, puisqu'il existe de nombreuses méthodes de résolution. Il est clair que ce qui a été développé dans ce rapport, peut

être améliorer et être plus précis. Et comme l'a précisé Laurent, il aurait fallut avoir une maîtrise du C++ pour approndir le sujet.

### 2.6.3 Xin

Dans ce projet, on a étudié les méthodes pour trouver les surfaces minimales. De plus, on a fait les applications avec FreeFem++ et visualisé les résultats . J'ai étudié comment créer les maillages avec FreeFem++ pour minimaliser une surface. Par ailleurs, j'ai amélioré ma connaissance de FreeFem++. La structure de surface minimale est stable. Donc les surfaces minimales peuvent être utilisées dans de nombreux domaines comme l'architecture, l'aviation, la fabrication des navires, etc.

### 2.6.4 Chen

Dans ce projet, j'ai étudié des algorithmes de freefem++ pour minimiser une surface. Pour bien comprendre les codes de freefem++, j'ai cherché et étudié les commandes de freefem++ et aussi le langage C++ étant donné que freefem++ est écrit en C++.

### 2.6.5 Fatime-Zahra

Souhaitant intégrer le département Génie Mathématique de l'INSA l'année prochaine, j'étais ravie d'avoir obtenu ce sujet de P6, qui est selon moi en parfaite relation avec ce département. En effet, l'intitulé du sujet étant "etude et détermination d'une surface minimale", j'en ai conclu que le travail sera de la modélisation basée sur de l'informatique et des mathématiques. Je ne savais cependant pas que ce cours nécessiterait la compréhension du language C++ que je ne maîtrise absolument pas. Même si je n'ai pas pu approfondir mes connaissances dans ce domaine, cela m'a permis de comprendre le fonctionnement du logiciel free fem++. Il permet de calculer et de tracer des surfaces basées sur un maillage puis de minimaliser la surface. Pour mieux appréhender le coté mathématique du sujet, il fallait se familiariser avec des outils mathématiques que nous n'avions que peu explorés pendant le cycle STPI. Enfin pour appliquer ces outils, j'ai essayé de comprendre la minimisation de surface qui permet d'obtenir la catenoïde.

# Bibliographie


[1] JEAN FREY Pascal et George Paul-Louis, *Le maillage facile*, 8 mai 2003.

[2] George Paul-Louis, *Maillage et adaptation*, octobre 2001.

[3] GONCALVÈS Eric, *Méthodes, analyse et calculs numériques*, septembre 2005.

[4] I.DANAILA, *Algorithmes et techniques avancées de programmation*, extrait du livre "Simulation numérique en C++, Dunod,2003".

[5] http://www.freefem.org/ff++ (Valide à la date du 23/04/14)

[6] http://www.freefem.org/ff++/ftp/freefem++doc.pdf (Valide à la date du 21/05/2014)

[7] http://perso.ensta-paristech.fr/~kielbasi/docs/freefem.pdf (Valide à la date du 21/05/2014)

[8] http://www.cmap.polytechnique.fr/IMG/pdf/Setif-FreeFem_II.pdf (Valide à la date du 21/05/2014)

[9] http://www.lamacs.fr/index.php?option=com_content&view=article&id=62&Itemid=73 (Valide à la date du 14/05/2014)

[10] http://www.math.rutgers.edu/~falk/math575/freefeminfo.html (Valide à la date du 14/05/2014)

[11] http://web.mit.edu/freefempp_v3.20/freefem++doc.pdf (Valide à la date du 06/06/14)

[12] http://www.sciences.ch/htmlfr/mecanique/mecananalytique01.php (Valide à la date du 06/06/14)

# Annexe A

# Utilisation de FreeFem++

1. On génère un maillage triangulaire régulier(Th) de taille $10 \times 10$ carré dans un domaine rectangulaire (carré $]0,1[^2$)

   ```
   mesh Th= square(10,10,[x,y]);
   plot(Th,wait=1);
   savemesh(Th,"Th.msh");
   ```

   plot :affiche des maillages. 'wait' détermine si la fenêtre graphique se ferme immédiatement ou non. savemesh : permet de sauver le maillage au format .msh
2. On définit l'espace d'approximation Vh

   ```
   fespace Vh(Th,P1);
   Vh u,du;
   ```
3. On définit la surface u que l'on veut minimiser et afficher.

   ```
   u = sqrt(0.5*0.5-(x-0.5)*(x-0.5));
   plot(u,wait=1,dim=3,fill=1);
   func surface = int2d(Th)( sqrt( 1+ dx(u)*dx(u) + dy(u)*dy(u) ));
   real surf = surface;
   cout << " surf = " << surf << endl;
   ```

   'cout' permet d'afficher les valeurs ou les phrases
4. On écrit la formule différenciée associée de la surface.

   ```
   real rho = 0.1;
   varf dsurf(uu,du) = int2d(Th)(( dx(du)*dx(u) + dy(du)*dy(u))
   /sqrt( 1+ dx(u)*dx(u) + dy(u)*dy(u) ))
   + on(1,2,3,4,uu=0) ;
   ```

   int2d : Calcul d'une intégrale 2D on : Définition des bords fixés dans un maillage carré, qui sont numérotés de 1 à 4.
5. On calcule le gradient en chaque point du maillage

   ```
   real eps = 1e-2;
   for(int iter = 0; iter < 1000; ++iter)
   {
   du[] = dsurf(0,Vh);
   u[] -= du[]* rho;
   cout << iter << " surf = " << surface << " " << du[].linfty << endl;
   if( du[].linfty < eps ) break;
   plot(u,wait=0,dim=3,fill=1);
   }
   ```

6. On définit un fonctionnel à minimiser.

```
func real J(real[int] & U)
{
Vh uu; uu[]=U;
return int2d(Th)( sqrt( 1+ dx(uu)*dx(uu) + dy(uu)*dy(uu) ));
}
```

uu est la fonction élément fini correspondand au tableau U.

7. On utilise IPOPT(Interior Point OPTimizer), un logiciel pour minimiser le gradient conjugué non linéaire.

```
load "ff-Ipopt"

func real[int] DJ(real[int]& U)

{

Vh u; u[]=U;

varf dsurf(uu,du) = int2d(Th)(( dx(du)*dx(u) + dy(du)*dy(u))
/sqrt( 1+ dx(u)*dx(u) + dy(u)*dy(u) ))
+ on(1,2,3,4,uu=0) ;

real[int] G= dsurf(0,Vh);

return G;}

int ret = IPOPT(J,DJ,u[]);

cout << " ipopt = " << surface << endl;

plot(u,wait=0,dim=3,fill=1,cmm="ipopt");
```

# Annexe B

# Autres surfaces minimisées

(a) $u = sin(2\pi x)$

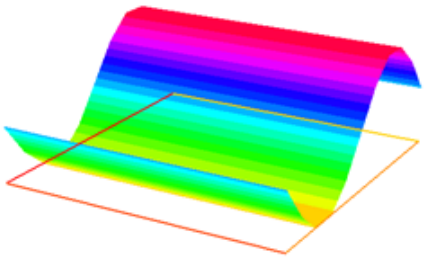


FIGURE B.1 – Avant minimisation

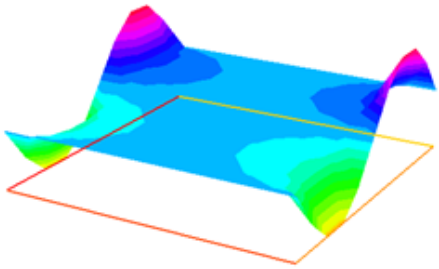


FIGURE B.2 – Après minimisation

(b) $u = sin(\pi x)cos(\pi(y - 0.5))$

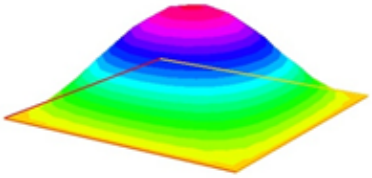


FIGURE B.3 – Avant minimisation

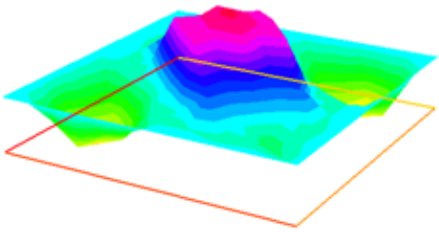


FIGURE B.4 – Après minimisation

(c) $u = \sqrt{(x + 0.5)^2 + (y - 0.5)^2} \times \frac{cos((x+0.5)\pi)}{sin((y+0.5)\pi}$

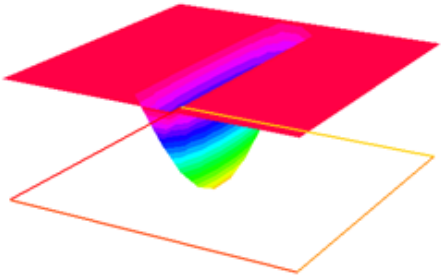


FIGURE B.5 – Avant minimisation

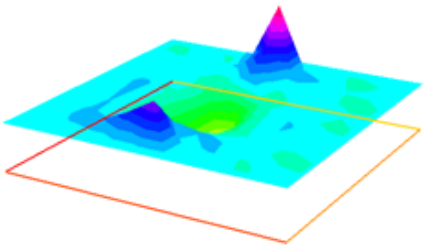


FIGURE B.6 – Après minimisation