Rapport de

Projet du problème de Plateau ---Surface minimale

Responsable Frédéric Hecht

Jingang QU

30/01/2019

Table de matières

1.Introduction

le problème de Plateau consiste à montrer, l'existence d'une surface minimale s'appuyant sur un bord donnée, En mathématique, soit Γ une courbe fermée de, le but est de trouver la surface S d'aire minimale de bord la courbe Γ.

Dans le cas discret,

Notre problème est de trouver $arg \min_{S_h \in \sum_h} \mathcal{J} = \sum_{K \in S_h} |K|$

D'où $\sum_h$ est un l'ensemble des surfaces discrètes admissibles, approchant Γ donnée. En facilitant notre problème, nous prenons l'ensembles des maillages 2D triangulaire du domaine Ω de bord Γ_h. Et où nous ajoutons une troisième coordonnée z_i à chaque sommet du maillage T_h.

La fonctionnelle $\mathcal{J}$ est convexe et donc le problème de minimisation est équivalent à résoudre l'équation de Euler-Lagrange non linéaire suivante :

Trouver les $z_i, \forall i \in I_0$ tel que.

$\forall i \in I_0,\ \sum_{K \in T_h} \partial z_i |K|$

2.Reformulation du problème

Paramètres :

Noterons u_i la troisième composante des sommets $A^i = (x_i, y_i, u_i)$

$\|N^K\| = \left(A^j - A^i\right) \wedge \left(A^k - A^i\right)$ La normal au triangle de A^i, A^j, A^k ;

On a $|K| = \|N^K\|/2$

$DN^K(v): v \longmapsto N^K(v) - N^K(0)$

Nous étudions la dépendance en u de N^K , écrivons $N^K(u)$ le vecteur normal sur le triangle K avec , $u = u_i = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} \in I_0$.

Notre problème de minimisation est donc

$arg \min_u \mathcal{J} = \sum_{K \in S_h} \|N^K(u)\|$

Développons,

$$\|N^K(u)\| = \begin{pmatrix} (y_j - y_i) * (u_k - u_i) - (u_j - u_i) * (y_k - y_i) \\ (x_k - x_i) * (u_j - u_i) - (x_j - x_i) * (u_k - u_i) \\ (x_j - x_i) * (y_k - y_i) - (y_j - y_i) * (x_k - x_i) \end{pmatrix}$$

Quand nous regardons la troisième composante, nous trouvons qu'elle ne dépend pas du tout u. Nous en conduisons que cette dépendance N^K(u) est affine en u. Nous travaillons donc en dimension 2 sur le plan.

Nous obtenons donc la différentielle de $DN^K(v)$: $v \longmapsto N^K(v) - N^K(0)$ (*), alors nous avons

Si K= (P^0, P^1, P^2) dans le plan, $DN^K(u) = \sum_{i=0}^{2} u^{i_g}\left(P^{i+2} - P^{i+1}\right)^{\perp}$.

D' où les opérations dans 0,1,2 sont modulo 3, et ig est le numéro du sommet global du sommet i de K

Développons $DN^K(u)$

$DN^K(u) = u^{0_g}(P^2 - P^1)^{\perp} + u^{1_g}(P^0 - P^2)^{\perp} + u^{2_g}(P^1 - P^0)^{\perp}$

$P^0 = (x_i, y_i)$ $P^1 = (x_j, y_j)$ $P^2 = (x_k, y_k)$

$P^2 - P^1 = \begin{bmatrix} x_k - x_j \\ y_k - y_j \end{bmatrix}$ $P^0 - P^2 = \begin{bmatrix} x_i - x_k \\ y_i - y_k \end{bmatrix}$ $P^1 - P^0 = \begin{bmatrix} x_j - x_i \\ y_j - y_i \end{bmatrix}$

Nous avons donc

$DN^K(u) = u_0 \begin{bmatrix} x_k - x_j \\ y_k - y_j \end{bmatrix} + u_1 \begin{bmatrix} x_i - x_k \\ y_i - y_k \end{bmatrix} + u_2 \begin{bmatrix} x_j - x_i \\ y_j - y_i \end{bmatrix}$

$N^K(0) = (0,0, \det(P^1 - P^0, P^2 - P^0))$

D'où $\det(P^1 - P^0, P^2 - P^0) = \begin{bmatrix} x_j - x_i & x_k - x_j \\ y_j - y_i & y_k - y_j \end{bmatrix}$

Maintenant, nous pouvons calculer la différentielle de $\|N^K(u)\|$,

$D\|N^K(u)\|$: $v \longmapsto \quad (DN^K(v).N^K(u))/\|N^K(u)\|$

L'équation d'Euler Lagrange peut se réécrire comme :

Trouver u ∈ $\mathbb{R}^I$, tel que $u_i = g_i \ \forall i \in I_0$

$\forall v \in R^{I_0}$, $\sum_K (DN^K(v).N^K(u))/\|N^K(u)\| = 0$ (1)

D'après (*), (1) devient

$\forall v \in R^{I_0}$

$\sum_K (DN^K(v).DN^K(u))/\|N^K(u)\| = -\sum_K (DN^K(v).N^K(0))/\|N^K(u)\|$

Après la reformulation, le problème est toujours non-linéaire

Finalement, nous allons baser sur le calcul du problème linéaire (P_a) :

$\forall v \in R^{I_0}$

$\sum_K (DN^K(v).DN^K(u))/\|N^K(w)\| = -\sum_K (DN^K(v).N^K(0))/\|N^K(w)\|$

D'après le calcul, nous savons que le membre est égale zéro. Le membre droit peut simplifier comme une forme linéaire A*U=0.

$DN^K(u)$ peut s'écrire sous la forme matricielle :

$DN^K(u) = \begin{pmatrix} y_j - y_k & y_k - y_i & y_i - y_j \\ x_k - x_j & x_i - x_k & x_j - x_i \\ 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} u^0 \\ u^1 \\ u^2 \end{pmatrix}$=A*U

De même pour $DN^K(v)$ = A*V

Nous avons donc A*U=0 ;

Si v=$\begin{pmatrix} v^0 \\ v^1 \\ v^2 \end{pmatrix}$=$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$, le produit scalaire de $DN^K(v)$ et $DN^K(u)$ est égale 0. Tous les coefficients de u sont zéros.

Si v=$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$, les coefficients avec u ne sont pas égaux à 0.

Comme il y a une partie des données qui sont connues $\begin{pmatrix} g_0 \\ g_1 \\ \vdots \end{pmatrix}$ et sur le bord parmi les u. Nous les détectons et les mettons dans b comme une seconde membre. En enlevant les éléments correspondants dans la matrice A. Nous obtenons donc un nouvel système linéaire DecompMatA*UC=b. nous pouvons obtenir u en résolvant ce système linéaire avec l'algorithme du gradient conjugué et puis nous pouvons donc trouver u^{i+1} en utilisant l'algorithme du point fixe.

3.Conception détaillé du C++

Dans cette partie, nous présentons la partie cruciale de notre code de manière détaillée : les classes, les fonctions et les algorithmes.

3.1 Classe R3 :

Idée : cette classe est inspirée par la classe R2, avec laquelle nous initialisons un point en dimension 3 au lieu 2. Par exemple R3 A ; ->A (x, y, z).

Les fonctions membres :

-4 constructeurs :

R3 () constructeur par défaut qui initialise chaque paramètre par 0.

R3(R a, R b, R c) en affectant les valeurs a,b,c.

R3(Vertex a, R b) qui initialise un couple de point de type Vertex en indiquant s'il est sur le bord

R3(const R3 & a, const R3 & b)

Initialise les trois paramètres en faisant la différence de deux points de type R3.

-3 fonctions :

R norme() const : pour calculer la norme de N^K et DN^K

Inline R3 det_(const R3 & A, const R3 & B, const R3 & C) pour calculer le produit vectoriel de deux objets R3

R3 operator^(const R3 & P) const : operator du produit vectoriel

3.2 Les fonctions :

Void build_g(double *g, const Mesh2d &Th, const vector<int> & BordeIdx, int Ng):

Initialise une matrice identité pour le vecteur $v \in R^{I_0}$.

Double borde(double x, double y, int lab) :

Initialise le bord en dépendant de la variation de fonction.

Remarque : Notre programme est désigné au cas le plus normal possible, c'est-à-dire que l'utilisateur n'a pas besoin de changer beaucoup de codes pour avoir un résultat. Sauf cette fonction, l'utilisateur peut saisir leurs bords différents en fonction de cas différent.

Int BordeGlobIdx(vector<int> &BordIdx, Mesh2d &Th) : rend les indices des points qui se trouvent sur le bord.

Void build_g(double* g, const vector<int>& BordeIdx, const Mesh2d &Th, int Ng);

Void build_W0(double* W, const vector<int>& BordeIdx, const double* g, int N)

Les deux fonctions initialisent respectivement deux vecteurs g_i et w. le vecteur w peut généralement stocker les valeurs différentes des points qui sont sur la frontière(Ici 0).

Void MatrixA(double *MatA, const Mesh2d &Th, const double *V, const double *W, const vector<int> & BordeIdx) :rendre la matrice MatA du système linéaire MatA*U=0.

Void DecompMatA(double* A, double* b, const double* MatA, const vector<int> & LocalIdx, const double * g, int N) : reconstruit la matrice MatA pour avoir le vrai système linéaire A*U=b en enlevant les points sur le bord.

Remarque : Nouvelle matrice A est bien symétrique définit positif.

4.Test avec FreeFem++

4.1 Maillage

4.1.1 Maillage triangulaire régulier dans un domaine rectangulaire

On considère le domaine] x0,x1[×]y0,y1[. Pour générer un maillage régulier n×m:

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

4.1.2 Maillage triangulaire non structuré e d défini à partir de ses frontières

Pour définir les frontières, on utilise la commande border :
border name(t=deb,fin){x=x(t);y=y(t);label=num_label};

Pour d définir un maillage `à partir de ses frontières,
on utilise buildmesh :
buildmesh nom_maillage=buildmesh(a1(n1)+a2(n2)+...+ak(nk));

4.2 Définition de l'espace d'approximation

fespace nom_espace(nom_maillage,type_elements_finis)

4.3 Entrées / sorties fichiers

savemesh(nom_maillage,nom_fichier); // sauver le maillage (.msh)
readmesh(nom_fichier); // lire un maillage `a partir d'un fichier
Lire les donné ees d'un maillage
Th.nt (nbre de triangle), Th.nv (nbre de noeuds), Th[i][j] (sommet j du triangle i), ...

4.4 Compilation de FreeFem++

Éditer un fichier "nomfichier.edp" avec
Vim nomfichier.edp ; // créer et entrer dans nomfichier.edp
I pour editer,
:wq sortir avec un enregistrement ;
FreeFem++ nomfichier.edp // lancer nomfichier.edp

5.Application : La catenoide

On génère le premier maillage du domaine $\Omega =]-\pi, \pi[^2$

```
mesh Th1 = square(20,20) ;
plot(Th1,wait=1,dim=3,fill=5,ps="nosplitmesh.eps") ;
savemesh(Th1, "mesh.msh");
```

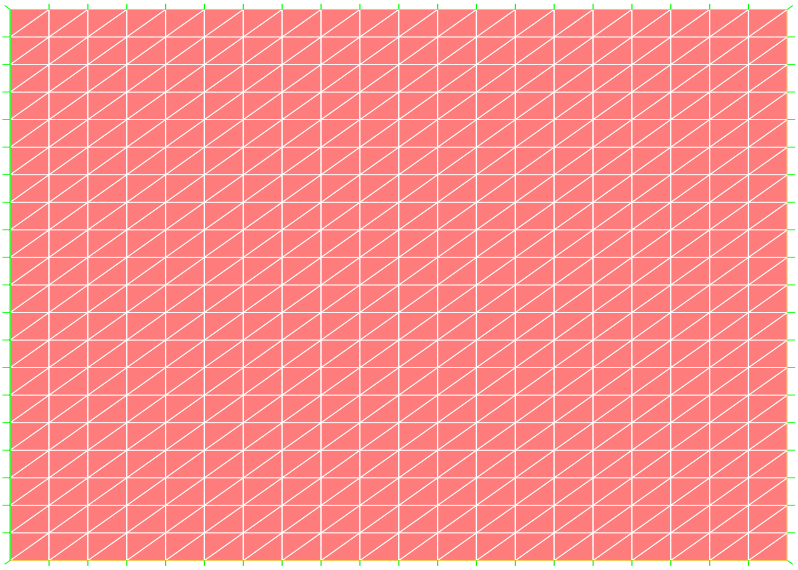

Figure 1 : maillage carré sur $]-\pi,\pi[^2$

5.1 Sur un maillage carré sur $]-\pi,\pi[^2$

```
mesh Th=square(20,20,[-pi+2*pi*x,-pi+2*pi*y]);
plot(Th,wait=1,ps="nosplitmesh.eps");
savemesh(Th,"catenoide.msh");
fespace Vh(Th,P1);
Vh u, du;
u=u(x,y) = cos(x) * cos(y);
plot(u,wait=1,dim=3,fill=1);
```

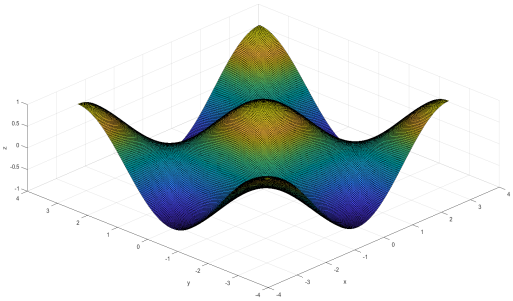

Figure 2 : Avant minimisation

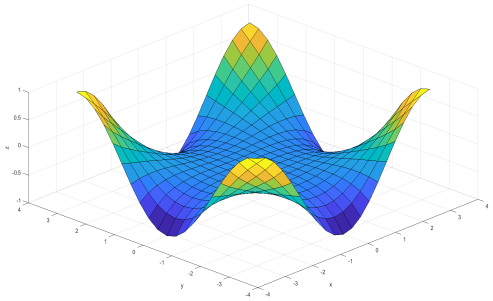

Figure 3 : Après minimisation

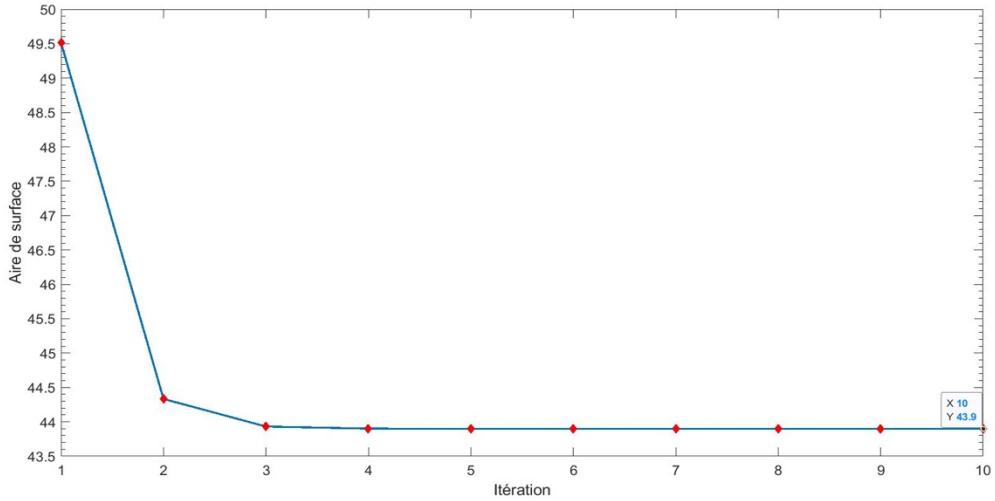

Figure : changement de l'aire

5.2 sur un maillage circulaire

```
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));
plot(Th,wait=1,ps="nosplitmesh.eps");
savemesh(Th,"catenoide.msh");
fespace Vh(Th,P1);
Vh u, du;
u=x*x+y*y;
plot(u,wait=1,dim=3,fill=1);
```

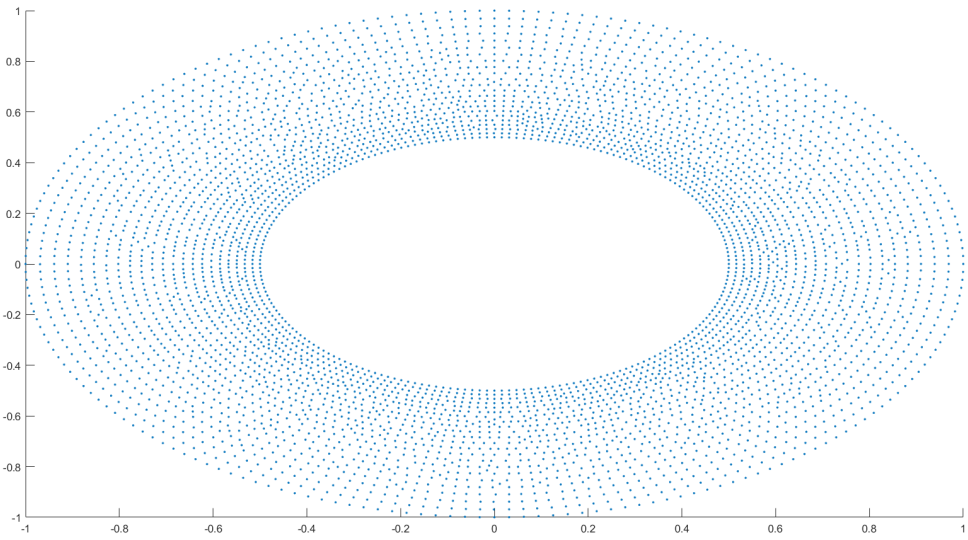

Figure 4: le maillage circulaire

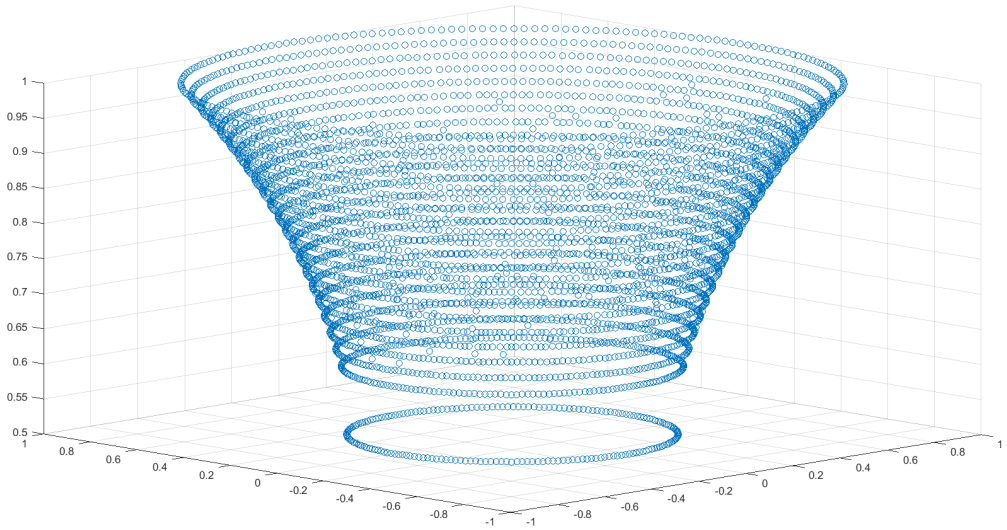

Figure 5 : la catenoide obtenue

6. Visualisation des autres cas personnels

6.1 u(x,y) = sin(2*pi*x)

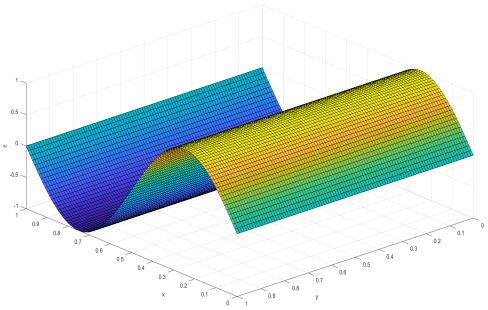

Figure 6 : Avant minimisation

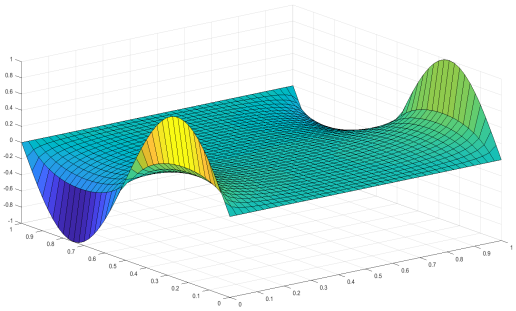

Figure 7 : Après minimisation

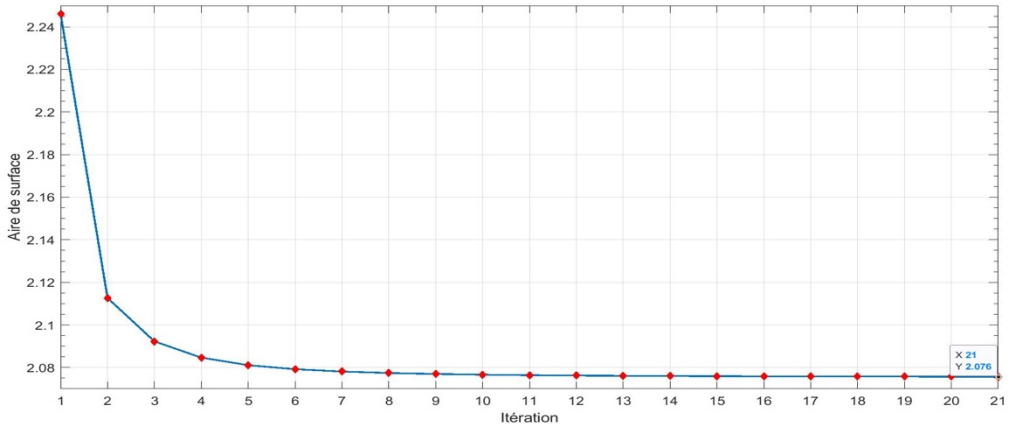

Figure 8 : changement de l'aire

6.2 U(x,y) = exp(x) + y^3 + cos(x*10*pi) + sin(y*10*pi);

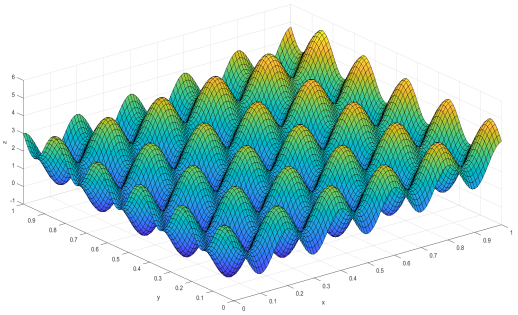

Figure 9 : Avant minimisation

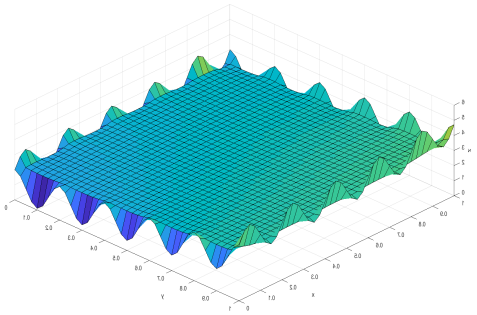

Figure 10 : Après minimisation

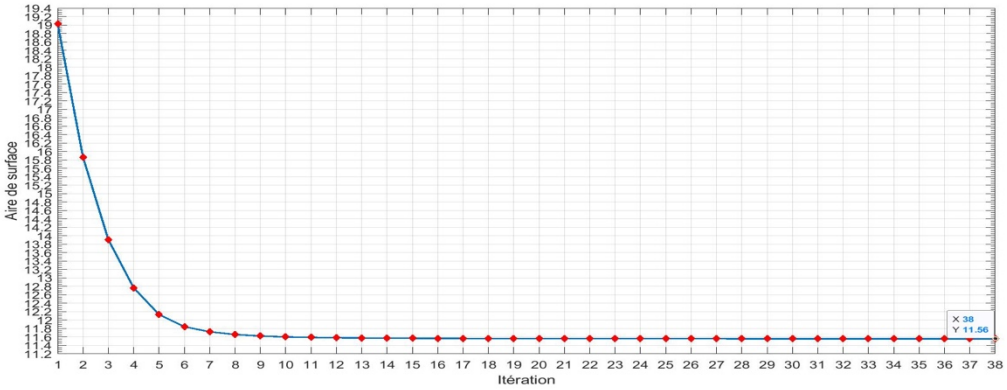

Figure 11 : changement de l'aire

6.3 u(x,y) =x^2 + y^2 + 5*cos(x*2*pi) + sin(y*3*pi)

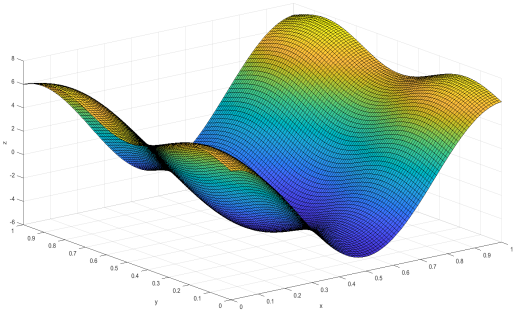

Figure 12 : Avant minimisation

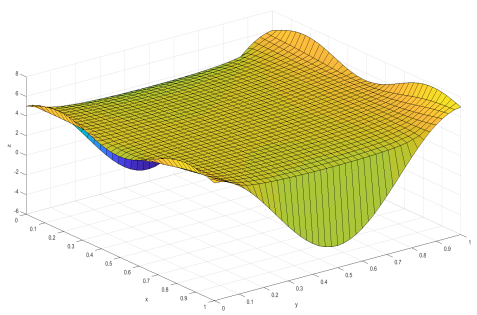

Figure 13 : Après minimisation

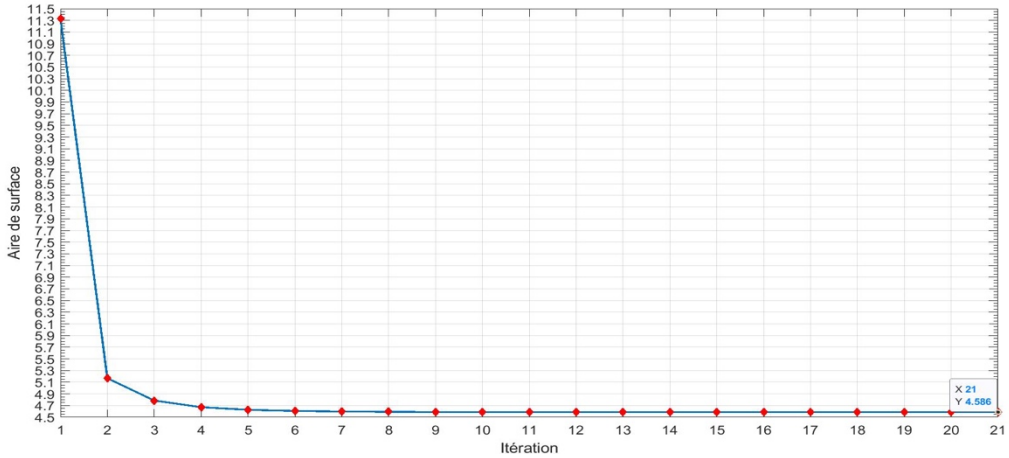

Figure 14 : changement de l'aire

7.Conclusion : Le C++, l'ensemble des programmes liés à la détermination de surfaces mi- nimales étant basé sur ce dernier. De plus, une des principales difficultés rencontrées au cours de ce projet fut la complexité des notions mathématiques et la manipulation des indices de la matrice A. Notamment, quand nous détectons les points sur le bord et essayons de renuméroter dans un nouvel tableau. En attendant le calcul de l'ordinateur, nous avons l'impression que plus le nombre de bords fixés est élevé et plus le calcul nécessite de temps.

8.Bibilographie

8.1 https://doc.freefem.org/documentation/

8.2 La Caténoide http://mathworld.wolfram.com/Catenoid.html

8.3 Introduction à FreeFem++.
http://www.cmap.polytechnique.fr/IMG/pdf/Setif-FreeFem_I.pdf

8.4 Problème de plateau https://fr.wikipedia.org/wiki/Problème_de_Plateau

8.5 Poly de l'approximation d'EDP M.Chrictinel Mardare
https://www.ljll.math.upmc.fr/mardare/enseignement/2018-2019/ApproxEDP/ApproxEDP-Chapitre3.pdf

8.6 Rapport d'Etude et determination d'une surface minimale