Calcul Scientifique

ZHANG Minghe, ZHENG Yijie

0 Introduction

Ce projet consiste à trouver la solution approcher de l'équation des dérivées partielles 1D en utilisant la méthode des éléments finis.

1 Préliminaires

1.1 Problème 1D

Soit $\Omega =]-a, a[\times]-b, b[$ un ouvert, on considère l'équation

$$\begin{cases} -div(\eta\nabla u)(x,y) = f(x,y); & \forall(x,y) \in \Omega \\ u(x,y) = 0; & \forall(x,y) \in \partial\Omega \end{cases}$$

où η et f désignent 2 fonctions continues de $\bar{\Omega}$
Si u est la solution du système, alors

$$\forall v \in V, -\int_\Omega div(\eta\nabla u)v \; dxdy = \int_\Omega fv \; dxdy$$

V est l'espace de fonction v de classe C^1 sur $\bar{\Omega}$ vérifiant $v(x,y) = 0 \;\; \forall(x,y) \in \partial\Omega$
D'après la formule de Green, on a

$$\int_\Omega div(\eta\nabla u)v \; d\Omega = -\int_\Omega \eta\nabla u\nabla v \; d\Omega + \int_{\partial\Omega} \eta\nabla uv \; d\Gamma$$

on sait que $u(x,y) = 0$ au bord, donc

$$\int_\Omega div(\eta\nabla u)v \; d\Omega = -\int_\Omega \eta\nabla u\nabla v \; d\Omega$$

$$\Longrightarrow \int_\Omega \eta(x,y)\nabla u(x,y)\nabla v(x,y) \; dxdy = \int_\Omega f(x,y)v(x,y) \; dxdy$$

1.2 Théorème de changement de variable pour des intégrales mul- tiples

Théorème[1] Si U est borné et si f et $(f \circ \Phi) \times |\det J\phi|$ sont Riemann-intégrables, alors

$$\iint_{\Phi(U)} f(x,y)dxdy = \iint_U f(\Phi(u,v))|\det J_\Phi(u,v)|dudv$$

où $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ de classe C^1, ici n = 2.

2 Le maillage

Pour avoir le maillage, on a fait des classes en C++ pour ce faire.

2.1 Les noeuds

Pour les noeuds, on a fait une classe `Point` pour les décrire.

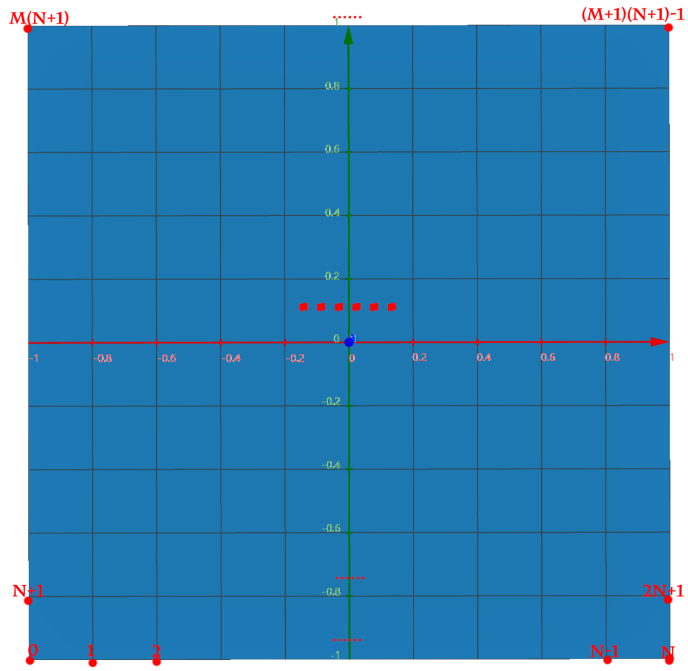
```
class Point
{
private:

public:
    double x;
    double y;
    Point(double x0=0,double y0 =0):x(x0),y(y0){}
    Point invnumgb(int N, int s);
    Point invnumint(int N, int k);
};
```

pour avoir les coordonnée des noeuds, il faut subdiviser le domaine Ω,voici les code:

```
vector<double> Subdiv(double a,int N)    //subdivision de l'intervalle
{
    vector<double> xi = vector<double>(N+1);
    for (int i=0;i<N+1;i++){
        xi[i] = -a +i*2*a/N;
    }
    return xi;
}
```

2.1.1 Numérotation de tous les noeuds

1 la règle de numérotation dans l'exemple ci-dessous



2 Fonction `numgb` qui renvoit le numéro global $s_{i,j}$ en fonction de couple (i, j) et entier N.

```
int numgb(int N, const Point &P)
{
    return (N+1)*P.y+P.x;
}
```

le numéro global $s_{i,j} = (N+1)j + i$, $\forall 0 \leq i \leq N; 0 \leq j \leq M$
Ici, i et j sont stockées dans le Point P.

3 D'après le théorème de division euclidienne, pour un couple d'entiers (a, b), il existe un unique couple d'entiers (q, r) tels que $a = bq + r$. Donc pour $s = s_{i,j} = (N+1)j + i$, on considère comme la division euclidenne de s par $(N+1)$, donc il existe un unique couple (i, j) vérifiant, d'où i est le quotient, j est le reste. $(i, j) = (s/(N+1), s\%(N+1))$.

4 Fonction `invnumgb` qui renvoit le couple (i, j) en fonction de numéro global de $s_{i,j}$ et entier N.

```
Point Point::invnumgb(int N, int s){
    x = s%(N+1);// i: le reste de la division s par (N+1)
    y = s/(N+1);// j: le quotient de la division s par (N+1)
    return *this;
}
```

2.1.2 Numérotation des noeuds intérieur

Comme la partie précédente, dans cette partie, on ne s'intéresse pas les noeuds sur le bord.

1 La fonction **numint** qui renvoit le numéro global intérieur $k_{i,j}$ en fonction de couple (i,j) et entier N.

```
int numint(int N, const Point &P)
{
    return (P.y-1)*(N-1)+P.x-1;
}
```

2 Même preuve que 2.1.1.3. On considère comme la division euclidienne de $(j-1)(N-1)+i-1$ par $(N-1)$. Le quotient est $j-1$, le reste est $i-1$.

Donc $(i,j) = (1 + k\%(N+1), 1 + k/(N+1))$

3 Fonction **invnumgb** qui renvoit le couple (i,j) en fonction de numéro global de $k_{i,j}$ et entier N.

```
Point Point::invnumint(int N, int k){
    x = k%(N+1)+1; // i-1 est le reste de la division k par (N+1)
    y = k/(N+1)+1; // j-1 est le quotient de la division k par (N+1)
    return *this;
}
```

4 La fonction **num_int_gb** comme paramètre de numéro intérieur k et entier N, renvoyant le numéro global s de ce noeud.

```
int num_int_gb(int N, int k){
    Point P;
    P = P.invnumint(N,k);  //couple (i,j)
    int s=numgb(N,P);   //numero global
    return s;
}
```

5 La fonction `num_gb_int` comme paramètre de numéro global s et entier N, renvoyant le numéro intérieur k de ce noeud.

```
int num_gb_int(int N, int s){
    Point P;
    P = P.invnumgb(N,s);  //couple (i,j)
    int k= numint(N,P);   //numero interieur
    return k;
}
```

Dans cette partie, on a une façon de représenter les noeuds, mais les coordonnées des noeuds sont des entiers. Plus précisement, le couple (i,j) représente i-ème noeud de l'abscisse et j-ème noeud d'ordonnée. Par conséquent, on a rajouté une fonction `Point_exact` qui prend l'entrée de a(domaine de l'abscisse), b(domaine de l'ordonnée), N, M, s(numéro global de noeud), en renvoyant le coordonnée exact de ce noeud.

```
Point point_exact(double a,double b,int N,int M,int s){
    vector<double> xi = Subdiv(a,N);     //subdivision de l'abscisse
    vector<double> yi = Subdiv(b,M);     //subdivision de l'ordonnee
    Point R;
    Point P;
    R.invnumgb(N, s);    //couple (i,j)
    P.x = xi[R.x];      //i-eme noeud de l'abscisse
    P.y = yi[R.y];      //j-eme noeud d'ordonnee
    return P;
}
```

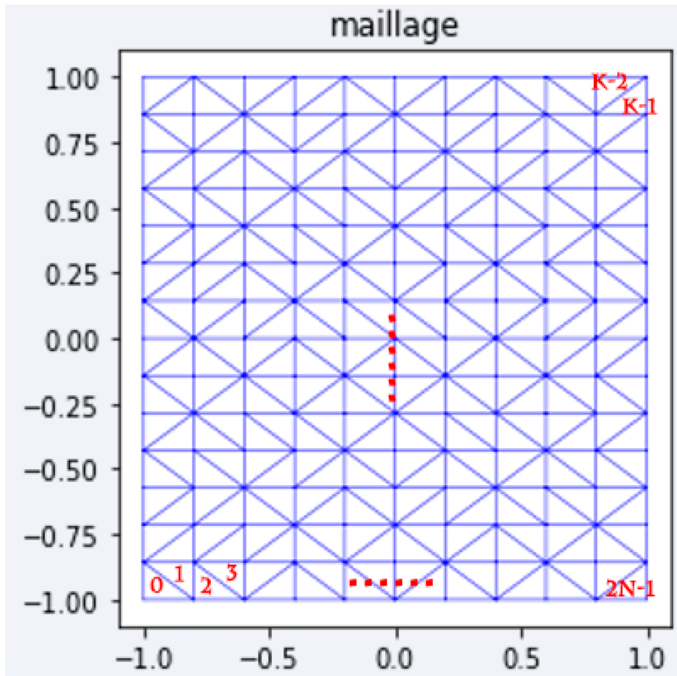
2.2 La triangulation

Dans cette partie, on a créé une classe `Triangle` pour décrire les triangles dans le maillage.

```
class Triangle
{
    private:

    public:
    double a;
    double b;
    double c;
    Triangle(double a0 = 0,double b0 = 0,double c0 = 0):a(a0),b(b0),c(c0){}
};
```

2.2.1 Une partition en triangles

1 Illustration de la numérotation des triangles:



Ici $K = 2NM$

2 La fonction `maillageTR` prend entrée des entier N et M, renvoit le tableau de triangles.

```
vector<Triangle>  maillageTR(int N,int M)
{
    vector<Triangle> TRG = vector<Triangle>(2*N*M);
    Point P;
    int i=0;   //compteur
    for(int s=0;s<(N+1)*(M)-1;s++){
        P.invnumgb(N,s);  //coordonnee de noeud
        if (P.x == N){   //touche le bord
            continue;
        }
        else{            //stocker 2 triangles a la fois
            TRG[2*i].a = s;      //triangle+
            TRG[2*i].b = s+N+1;
            TRG[2*i].c = s+N+2;
            TRG[2*i+1].a = s;    //triangle-
            TRG[2*i+1].b = s+1;
```

```
            TRG[2*i+1].c = s+N+2;
            i++;
        }
    }
    return TRG;
}
```

Ici on n'a qu'un type de triangle dans le maillage, afin de simplifier le code.

2.2.2 Coordonnées barycentriques dans un triangle. Triangle de référence.

1

Démonstration: Posons $\phi : \mathbb{R}^2 \to \mathbb{R}$, $\vec{x} \mapsto f(\vec{x}) - f(0)$.

Suppons f affine, montrer que ϕ linéaire:

$$\begin{aligned}
\phi(\alpha\vec{x} + (1-\alpha)\vec{y}) &= f(\alpha\vec{x} + (1-\alpha)\vec{y}) - f(0) \\
&= \alpha f(\vec{x}) + (1-\alpha) f(\vec{y}) - f(0) \\
&= \alpha(f(\vec{x}) - f(\vec{y})) + f(\vec{y}) - f(0) \\
&= \alpha(f(\vec{x}) - f(0) - f(\vec{y}) + f(0)) + \phi(\vec{y}) \\
&= \alpha(\phi(\vec{x} - \phi(\vec{y}) + \phi(\vec{y}) \\
&= \alpha\phi(\vec{x}) + (1-\alpha)\phi(\vec{y})
\end{aligned}$$

Donc on a bien montré que ϕ est linéaire.

Inversement, supposons que ϕ linéaire, montrer que f affine:

$$\begin{aligned}
\phi(\alpha\vec{x} + (1-\alpha)\vec{y}) &= f(\alpha\vec{x} + (1-\alpha)\vec{y}) - f(0) \\
&= \alpha\phi(\vec{x}) + (1-\alpha)\phi(\vec{y}) \\
&= \alpha(f(\vec{x}) - f(0)) + (1-\alpha)(f(\vec{y}) - f(0)) \\
&= \alpha f(\vec{x}) + (1-\alpha) f(\vec{y}) - f(0)
\end{aligned}$$

On a donc $f(\alpha\vec{x} + (1-\alpha)\vec{y}) = \alpha f(\vec{x}) + (1-\alpha) f(\vec{y})$, donc f est bien affine.

□

2

Démonstration: Il suffit de montrer que l'application $g : \mathbb{R}^2 \to \mathbb{R}, (x, y) \mapsto f(x, y) - f(0, 0)$ est linéaire.

- $f(0, 0) = \gamma$, donc $g(x, y) = \alpha x + \beta y + \gamma - f(0, 0) = \alpha x + \beta y$.

- Soient $(x,y),(s,t) \in \mathbb{R}^2$, $\lambda, \mu \in \mathbb{R}$,

$$\begin{aligned} g(\lambda(x,y)+\mu(s,t)) &= g((\lambda x, \lambda y)+(\mu s, \mu t)) \\ &= g(\lambda x+\mu s, \lambda y+\mu t) \\ &= f(\lambda x+\mu s, \lambda y+\mu t) - f(0,0) \\ &= \alpha(\lambda x+\mu s)+\beta(\lambda y+\mu t) \\ &= \lambda(\alpha x+\beta y)+\mu(\alpha s+\beta t) \\ &= \lambda g(x,y)+\mu g(s,t) \end{aligned}$$

Ce qui montre la linéairité de g. D'après 1, f est donc affine. $\square$

3

Démonstration: On a

$$\begin{cases} \lambda_0 x_0+\lambda_1 x_1+\lambda_2 x_2 &= x \\ \lambda_0 y_0+\lambda_1 y_1+\lambda_2 y_2 &= y \\ \lambda_0+\lambda_1+\lambda_2 &= 1 \end{cases} \tag{1}$$

Le système (1) peut s'écrire sous la forme matricielle.

$$\begin{pmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \lambda_0 \\ \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Pour montrer que ce système a une solution, il suffit de montrer que $\begin{pmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{pmatrix}$ est inversible.

Pour cela, on calcule le déterminant:

$$\begin{aligned} \begin{vmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} x_0 & x_1-x_0 & x_2-x_0 \\ y_0 & y_1-y_0 & y_2-y_0 \\ 1 & 0 & 0 \end{vmatrix} &= \begin{vmatrix} x_1-x_0 & x_2-x_0 \\ y_1-y_0 & y_2-y_0 \end{vmatrix} \\ &= (x_1-x_0)(y_2-y_0)-(x_2-x_0)(y_1-y_0) \end{aligned}$$

Le triangle T est non aplati, c'est-à-dire $(x_0,y_0),(x_1,y_1)$ et (x_2,y_2) ne sont pas alignés, donc le déterminant ne peut pas être nul. Alors, la matrice est inversible, donc l'existence de la solution a été prouvé. Comme le système a 3 équations et 3 inconnues, donc il existe une unique solution, cqfd. $\square$

4 Pour trouver λ_0, λ_1 et λ_2, on calcule l'inverse de la matrice:

$$\begin{pmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} y_1 - y_2 & x_2 - x_1 & x_1y_2 - x_2y_1 \\ y_2 - y_0 & x_0 - x_2 & x_2y_0 - x_0y_2 \\ y_0 - y_1 & x_1 - x_0 & x_0y_1 - x_1y_0 \end{pmatrix} \times \frac{1}{(x_1 - x_0)(y_2 - y_0) - (x_0 - x_2)(y_0 - y_1)}$$

Donc

$$\lambda_0 = \frac{(y_1 - y_2)x + (x_2 - x_1)y + x_1y_2 - x_2y_1}{(x_1 - x_0)(y_2 - y_0) - (x_0 - x_2)(y_0 - y_1)}$$

λ_0 peut donc s'écrire de la forme $\alpha x + \beta y + \gamma$ avec
$\alpha = \frac{y_1-y_2}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)}, \beta = \frac{x_2-x_1}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)}$, et $\gamma = \frac{x_1y_2-x_2y_1}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)}$.

Idem pour λ_1 et λ_2:

$$\lambda_1 = \frac{(y_2 - y_0)x + (x_0 - x_2)y + x_2y_0 - x_0y_2}{(x_1 - x_0)(y_2 - y_0) - (x_0 - x_2)(y_0 - y_1)}$$

λ_1 peut s'écrire sous la forme $\alpha x + \beta y + \gamma$ avec
$\alpha = \frac{y_2-y_0}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)}, \beta = \frac{x_0-x_2}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)}$, et $\gamma = \frac{x_2y_0-x_0y_2}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)}$.

$$\lambda_2 = \frac{(y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0}{(x_1 - x_0)(y_2 - y_0) - (x_0 - x_2)(y_0 - y_1)}$$

λ_2 peut s'écrire sous la forme $\alpha x + \beta y + \gamma$ avec
$\alpha = \frac{y_0-y_1}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)}, \beta = \frac{x_1-x_0}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)}$, et $\gamma = \frac{x_0y_1-x_1y_0}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)}$.

Donc, les fontions $\lambda_i, 0 \leq i, j \leq 2$ sont toutes affines.

5 En appliquant (x_i, y_i) dans l'expression de $\lambda_j(x_i, y_i)$ trouvée dans la question 4 , on a $\lambda_j(x_i, y_i) = 0$ pour $0 \leq i, j \leq 2, i \neq j$.

6

Démonstration: Comme f est une fonction affine, d'après 2.2, il existe α, β, γ tels que $f(x, y) = \alpha x + \beta y + \gamma$. Comme on a $x = \lambda_0 x_0 + \lambda_1 x_1 + \lambda_2 x_2$, $y = \lambda_0 y_0 + \lambda_1 y_1 + \lambda_2 y_2$, on a

$$\begin{aligned} f(x, y) &= \alpha x + \beta y + \gamma \\ &= \alpha(\lambda_0 x_0 + \lambda_1 x_1 + \lambda_2 x_2) + \beta(\lambda_0 y_0 + \lambda_1 y_1 + \lambda_2 y_2) + \gamma(\lambda_0 + \lambda_1 + \lambda_2) \\ &= \lambda_0(\alpha x_0 + \beta y_0 + \gamma) + \lambda_1(\alpha x_1 + \beta y_1 + \gamma) + \lambda_2(\alpha x_2 + \beta y_2 + \gamma) \\ &= \lambda_0 f(x_0, y_0) + \lambda_1 f(x_1, y_1) + \lambda_2 f(x_2, y_2), \text{cqfd.} \end{aligned}$$

□

Si $f(x_i, y_i) = 0$ pour $0 \leq i, j \leq 2$, par précédent, $f = 0$, donc f est identiquement nulle.

7 On a $\hat{x_1} = 1$ et $\hat{y_2} = 1$. Les autres $\hat{x_i}, \hat{y_i}$ sont tous nuls, donc

$$\begin{cases} \hat{\lambda_0} = 1 - x - y \\ \hat{\lambda_1} = x \\ \hat{\lambda_2} = y \end{cases}$$

8 On a

$$\begin{cases} F_T(0,0) = (x_0, y_0) \\ F_T(1,0) = (x_1, y_1) \\ F_T(0,1) = (x_2, y_2) \end{cases} \tag{2}$$

F_T s'écrit sous la forme $F_T(x_i, y_i) = A(x_i, y_i) + B$, d'où A est une matrice 2×2, B est un vecteur 2×1. Par le système (2), $B = (x_0, y_0)^T$. On en déduit donc $A = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix}$. Pour vérifier (2), cette écriture est unique.

Donc

$$F_T \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix} \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

9 Supposons $F_T \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$, alors

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix} \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

Ce qui implique

$$\begin{aligned} \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} &= \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix}^{-1} \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix} \\ &= \frac{1}{(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)} \begin{pmatrix} y_2 - y_0 & x_0 - x_2 \\ y_0 - y_1 & x_1 - x_0 \end{pmatrix} \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix} \end{aligned}$$

C'est-à-dire:

$$F_T^{-1} : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \frac{1}{(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)} \begin{pmatrix} y_2 - y_0 & x_0 - x_2 \\ y_0 - y_1 & x_1 - x_0 \end{pmatrix} \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix}$$

Par précédent, F_T^{-1} est bien définie, donc F_T est inversible.

10
$$B_T = DF_T = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix}$$

11 Pour calculer la matrice B_T, il nous faut le noeud exact du sommet du triangle, pour ce faire, on a programmé une fonction `sommetTR`, en utilisant la fonction `Point_exact` précédente, afin de calculer le noeud exact du sommet.

```
vector<Point> sommetTR(double a,double b,int N,int M,const Triangle &T){
    vector<Point> St = vector<Point>(3);
    St[0] = point_exact(a,b,N,M,T.a);   //sommet 1
    St[1] = point_exact(a,b,N,M,T.b);   //sommet 2
    St[2] = point_exact(a,b,N,M,T.c);   //sommet 3
    return St;
}
```

Ensuite, le calcul de la matrice B_T devient facile.

```
vector<vector<double>> CalcMatBT(double a,double b,int M,const Triangle& t)
{
    vector<vector<double>> BT(2,vector<double>(2));
    int N = t.c - t.a -2;      //N
    vector<Point> St = sommetTR(a,b,N,M,t);   //sommet du triangle
    Point p0 = St[0];
    Point p1 = St[1];
    Point p2 = St[2];
    BT[0][0] = p1.x-p0.x;
    BT[0][1] = p2.x-p0.x;
    BT[1][0] = p1.y-p0.y;
    BT[1][1] = p2.y-p0.y;
    return BT;
}
```

On peut aussi ajouter une fonction `DetBT` qui calcule le déterminant de la matrice B_T, pour faciliter les étapes suivantes.

```
double DetBT(double a,double b,int M,const Triangle&t)    // Derminant de BT
{
    vector<vector<double>> BT = CalcMatBT(a,b,M,t);
    return (BT[0][0]*BT[1][1]-BT[0][1]*BT[1][0]);
}
```

12

$$\begin{aligned}\text{air}T &= \int_T 1\mathrm{d}x\\ &= \int_{\hat{T}} 1\mathrm{d}\hat{x}\\ &= \frac{1}{2}|\text{dét}B_T|\end{aligned}$$

13

$$\lambda_i = \hat{\lambda}_i \circ F_T^{-1}$$

14

$$\nabla\lambda_i = (B_T^{-1})^T \circ \nabla\hat{\lambda}_i(F_T^{-1})$$

15

$$\nabla\lambda_i.\nabla\lambda_j = (B_T^{-1})^T.\nabla\hat{\lambda}_i.(B_T^{-1})^T.\nabla\hat{\lambda}_j$$

16

(a) Par le théorème de changement de variable pour les intégrales multiples,

$$\int_T H(x,y)dxdy = \int_{\hat{T}} |\det(B_T)|h(\hat{x},\hat{y})d\hat{x}d\hat{y} = |\det(B_T)|\int_{\hat{T}} h(\hat{x},\hat{y})d\hat{x}d\hat{y}.$$

(b)

1. $d=0$: $h:\mathbb{R}^2 \to \mathbb{R}, (x,y)\mapsto 1$.

   $$\int_0^1\int_0^{1-\hat{x}} h(\hat{x},\hat{y})d\hat{x}d\hat{y} = \int_0^1\int_0^{1-\hat{x}} 1d\hat{x}d\hat{y} = \int_0^1 (1-\hat{y})d\hat{y} = \frac{1}{2} = \frac{1}{6}+\frac{1}{6}+\frac{1}{6}$$

   Donc vrai pour $d=0$.

2. $d=1$: $h:\mathbb{R}^2 \to \mathbb{R}, (x,y)\mapsto y$

   $$\int_0^1\int_0^{1-\hat{y}} h(\hat{x},\hat{y})d\hat{x}d\hat{y} = \int_0^1\int_0^{1-\hat{x}} yd\hat{y}d\hat{x} = \int_0^1 \frac{1}{2}(1-\hat{x})^2d\hat{x} = \frac{1}{6} = \frac{1}{12}+\frac{1}{12}$$

   idem pour $(x,y)\mapsto x$, donc vrai pour $d=1$.

3. $d = 2$: $h : \mathbb{R}^2 \to \mathbb{R}, (x, y) \mapsto x^2$

$$\begin{aligned}\int_0^1 \int_0^{1-\hat{y}} h(\hat{x}, \hat{y}) d\hat{x} d\hat{y} = \int_0^1 \int_0^{1-\hat{x}} x^2 d\hat{y} d\hat{x} &= \frac{1}{3} \int_0^1 (1-y)^3 d\hat{y} \\ &= \frac{1}{12} [(1-y)^4]_0^1 \\ &= \frac{1}{12} = \frac{1}{6} \times \frac{1}{4} + \frac{1}{6} \times \frac{1}{4}\end{aligned}$$

idem pour $(x, y) \mapsto y^2$.

pour $h : \mathbb{R}^2 \to \mathbb{R}, (x, y) \mapsto xy$:

$$\begin{aligned}\int_0^1 \int_0^{1-\hat{y}} h(\hat{x}, \hat{y}) d\hat{x} d\hat{y} = \int_0^1 \int_0^{1-\hat{x}} xy d\hat{y} d\hat{x} = \int_0^1 x \int_0^{1-\hat{x}} y d\hat{y} d\hat{x} &= \frac{1}{2} \int_0^1 x(1-x)^2 d\hat{x} \\ &= \frac{1}{2} [\frac{1}{4} y^4 - \frac{2}{3} y^3 + \frac{1}{2} y^2]_0^1 \\ &= \frac{1}{24} = \frac{1}{6} \times \frac{1}{2} \times \frac{1}{2}\end{aligned}$$

4. $d = 3$: $h : \mathbb{R}^2 \to \mathbb{R}, (x, y) \mapsto x^3$

$$\begin{aligned}\int_0^1 \int_0^{1-\hat{y}} h(\hat{x}, \hat{y}) d\hat{x} d\hat{y} = \int_0^1 \int_0^{1-\hat{y}} x^3 d\hat{x} d\hat{y} &= \frac{1}{4} \int_0^1 (y-1)^4 d\hat{y} \\ &= \frac{1}{20} [(y-1)^5]_0^1 \\ &= \frac{1}{20}\end{aligned}$$

Mais, $\frac{1}{6} \times (\frac{1}{2})^3 = \frac{1}{48} \neq \frac{1}{20}$

En conclusion, $d \leq 2$.

(c) La fonction `GradGrad` renvoit la matrice GrdGrd de taille 3x3 et dont les coefficients sont

$$\int_T \eta \nabla \lambda_k \nabla \lambda_j \, dxdy, \ 0 \leq k, j \leq 2$$

en utilisant la fonction **eta** et les coordonnée exacts du triangle t.

```
vector<vector<double>> GradGrad(double a,double b,int M,const Triangle &t,
double (*eta)(double x,double y)){
    int N = t.c - t.a -2;   //N
    vector<Point> St = sommetTR(a,b,N,M,t);   //les sommets
    double D = abs(DetBT(a,b,M,t));  //la valeur absolue du determinant de BT
    vector<vector<double>> GrdGrd(3,vector<double>(3));
```

```
    double val = (eta(St[0].x,St[0].y)+
    eta(St[1].x,St[1].y)+eta(St[2].x,St[2].y)); //valeur de l'integrale d'eta
    GrdGrd[0][0] = 1./6/D*(pow(St[1].y-St[2].y,2)+pow(St[2].x-St[1].x,2))*val;
    GrdGrd[0][1] = 1./6/D*((St[1].y-St[2].y)*(St[2].y-St[0].y)+
    (St[2].x-St[1].x)*(St[0].x-St[2].x))*val;
    GrdGrd[0][2] = 1./6/D*((St[1].y-St[2].y)*(St[0].y-St[1].y)+
    (St[2].x-St[1].x)*(St[1].x-St[0].x))*val;
    GrdGrd[1][0] = 1*GrdGrd[0][1];
    GrdGrd[1][1] = 1./6/D*(pow(St[2].y-St[0].y,2)+pow(St[0].x-St[2].x,2))*val;
    GrdGrd[1][2] = 1./6/D*((St[2].y-St[0].y)*(St[0].y-St[1].y)+
    (St[0].x-St[2].x)*(St[1].x-St[0].x))*val;
    GrdGrd[2][0] = 1*GrdGrd[0][2];
    GrdGrd[2][1] = 1*GrdGrd[1][2];
    GrdGrd[2][2] = 1./6/D*(pow(St[0].y-St[1].y,2)+pow(St[1].x-St[0].x,2))*val;
    return GrdGrd;
}
```

17

Démonstration:

$$M \in T \Leftrightarrow x_0 \le x \le x_1, y_0 \le y \le y_2$$

$\lambda_0 = \dfrac{(y_1-y_2)x+(x_2-x_1)y+x_1y_2-x_2y_1}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)}$, On remplace x et y dans l'expression de λ_0:

$$M \in T \Leftrightarrow \frac{(y_1-y_2)x_0+(x_2-x_1)y_0+x_1y_2-x_2y_1}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)} \le \lambda_0$$

On simplifie cette expression et on a:$\lambda_0 \ge 1$.
Idem pour λ_1 et λ_2:
$\lambda_1 = \frac{(y_2-y_0)x+(x_0-x_2)y+x_2y_0-x_0y_2}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)} \Rightarrow \frac{(y_2-y_0)x_0+(x_0-x_2)y_0+x_2y_0-x_0y_2}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)} \le \lambda_1 \Rightarrow \lambda_1 \ge 0.$
$\lambda_2 = \frac{(y_0-y_1)x+(x_1-x_0)y+x_0y_1-x_1y_0}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)} \Rightarrow \frac{(y_0-y_1)x_0+(x_1-x_0)y_0+x_0y_1-x_1y_0}{(x_1-x_0)(y_2-y_0)-(x_0-x_2)(y_0-y_1)} \le \lambda_2 \Rightarrow \lambda_2 \ge 0.$
On a donc $M \in T \Leftrightarrow \lambda_i(x,y) \ge 0, 0 \le i \le 2$. □

18 La fonction `DansTrg` renvoit l'info $= 1$ si le point (x, y) est dans le triangle et l'info $= 0$ sinon.

```
int DansTrg(double a,double b,int N,int M,const Triangle &t,double x,double y)
{
    int info = 0;
    vector<Point> St = sommetTR(a,b,N,M,t);  //les sommets
    if ((x<=St[2].x)and(x>=St[0].x)and(y<=St[2].y)and(y>=St[0].y)){
        // si x et y sont entre les valeurs des sommets
```

```
        if (St[0].x==St[1].x){  //si c'est triangle-
            if((y-St[0].y)/(x-St[0].x) >= 1.){
                //la pente entre (x,y) et le somme
                info = 1;
            }
        }
        else{  //si c'est triangle+
            if((y-St[0].y)/(x-St[0].x) <= 1.){
                //la pente entre (x,y) et le sommet
                info = 1;
            }
        }
    }
    return info;
}
```

3 Le problème discret

1

Démonstration. Pour montrer que I_h est un isomorphisme, il suffit de montrer que pour un i fixé, compris entre 0 et 2, tous les noeuds intérieurs M_k^i associe à une unique application $v \in \mathcal{C}_0^0(\bar{\Omega})$ telle que $v(M_k^i) = 0$, $0 \leq k \leq I-1$.

La restriction de v à un triangle T_l est une fonction affine, donc il existe trois constantes α, β et γ tels que $v(x,y) = \alpha x + \beta y + \gamma$. Comme chaque M_k^i appartient à un triangle T_l, les α, β et γ sont déterminés d'une manière unique par ses valeurs au sommets de T_l, donc v est unique. On a donc I_h est un isomorphisme.

Dans ce cas-là, la dimension de V_h est égale au nombre de noeuds du maillage, $\dim V_h = \dim(\mathbb{R}^I) = I$. □

2 Comme I_h est un isomorphisme de $V_h \to \mathbb{R}^I$ et $\dim V_h = I$, chaque élément $w_k(M_k^i)$ de la famille $(w_k)_{0\leq k \leq I-1}$ associé à un élément de $\mathbb{R}^I$, qui vaut 1 si $k = l$ et 0 sinon. On a donc $(w_k(M_k^i))_{0\leq k\leq I-1} = (e_k)_k$, où e_k est un vecteur de la base canonique de $\mathbb{R}^I$. Donc, l'image réciproque de M_k^i forme une base de V_h, c'est-à-dire $(w_k)_{0\leq k\leq I-1}$ forme une base de V_h.

3

a Si M_k^i n'est pas un sommet de T, alors la restriction de w_k à T est nulle.

b Si M_k^i est un sommet de T, alors la restriction de w_k à T est égale à λ_i.

c

$$\nabla w_k.\nabla w_j = \nabla \lambda_l.\nabla \lambda_s = (B_T^{-1})^T.\nabla \hat{\lambda_l}.(B_T^{-1})^T.\nabla \hat{\lambda_s}$$

4 Comme $u_h \in V_h$, la solution u_h peut s'écrit sur la base $(w_k)_{0\leq k\leq I-1}$, c'est-à-dire

$$u_h = \sum_{k=0}^{i-1} \alpha_k w_k(x,y)$$

Soit $S_i \in \Omega$,$S_i = (x,y)$. D'après la question 3, $u_h(S_i) = \alpha_k \lambda_{i_s}(x,y)$, d'où

$$\forall j \leq I-1, \alpha(\sum_{j=0}^{I-1} u_h(x,y)w_k(x,y)w_j(x,y)) = \int_\Omega f(x,y)w_k(x,y)dxdy$$

Posons $\alpha = \int_\Omega \eta(x,y)dxdy$, on peut reformuler cette équation:

$$\begin{aligned}\alpha(\sum_{j=0}^{I-1} u_h(x,y)w_k(x,y)w_j(x,y)) &= \sum_{j=0}^{I-1}\int_\Omega \eta(x,y)dxdy.w_k(x,y)w_j(x,y).u_h(x,y)\\ &= \sum_{j=0}^{I-1}\int_\Omega \eta(x,y)\nabla w_k(x,y)\nabla w_j(x,y)dxdy.u_h(x,y)\\ &= \int_\Omega f(x,y)w_k(x,y)dxdy\end{aligned}$$

Cette équation peut donc s'écrire sous la forme matricielle $AU = B$ avec $A = ((a_{k,j}))_{0\leq k,j\leq I-1}$ et B=$(b_0,\dots,b_{I-1})^t$ définis par

$$a_{k,j} = \int_\Omega \eta(x,y)\nabla w_k(x,y)\nabla w_j(x,y)dxdy.u_h(x,y)$$

$$b_k = \int_\Omega f(x,y)w_k(x,y)dxdy$$

5 Il suffit de montrer que $\forall v \in \mathbb{R}^I$, $v^T A v \geq 0$.

$$\begin{aligned}
v^T A v &= \sum_k \sum_j v_j a_{kj} v_k \\
&= \sum_k \sum_j v_j v_k \int_\Omega \eta(x,y) \nabla w_k(x,y) . \nabla w_j(x,y) dx dy \\
&= \sum_k \sum_j v_j v_k \int_\Omega \eta(x,y) \nabla \lambda_k(x,y) . \nabla \lambda_j(x,y) dx dy \\
&= \int_\Omega \eta(x,y) \sum_k v_k \nabla \lambda_k(x,y) \sum_j v_j \nabla \lambda_j(x,y) dx dy \\
&= \int_\Omega \eta(x,y) u^2(x,y) dx dy
\end{aligned}$$

En faisant un changement de variable $u = \sum_i v_i \nabla \lambda_i(x,y)$.

D'après la définition de la partie 1 de la fonction η, on a toujours $\eta(x,y) \geq \eta_0$, d'où $\eta_0 > 0$, donc la fonction η est toujours positive. Comme u^2 est toujours positive comme une carrée, on a $\int_\Omega \eta(x,y) u^2(x,y) dx dy \geq 0$, donc A est strictement positive.

6

(a) Ici, la fonction `extendVec` qui transforme le vecteur V à $\tilde{V}$, où $V \in \mathbb{R}^I$ dont chacune des composantes peut être associée à un noeud intérieur, $\tilde{V} \in \mathbb{R}^G$ dont chacune des composantes peut être associée à un noeud intérieur ou sur le bord.

```
vector<double> extendVec(vector<double> V,int N,int M){
    vector<double> VV = vector<double>((N+1)*(M+1));
    int c = 0;  //compteur
    for (int j=0;j<VV.size();j++){
        if ((j<N+1) or (j>M*(N+1)-1) or (j%(N+1)==0) or ((j+1)%(N+1)==0)){
            // sur le bord en bas ou en haut ou a gauche ou a droite
            VV[j] = 0;
        }
        else{
            VV[j] = V[c];
            c += 1;
        }
    }
    return VV;
}
```

(b) La fonction `IntVec` est l'inverse de la fonction précédente, c'est-à-dire, cette fonction transforme le vecteur $\tilde{V} \in \mathbb{R}^G$ à vecteur $V \in \mathbb{R}^I$

```
vector<double> IntVec(vector<double>VV,int N,int M){
    vector<double> V = vector<double>((N-1)*(M-1));
    int c = 0;   //compteur
    for (int j=0;j<VV.size();j++){
        if ((j<N+1) or (j>M*(N+1)-1) or (j%(N+1)==0) or ((j+1)%(N+1)==0)){
            // sur le bord en bas ou en haut ou a gauche ou a droite
            continue;
        }
        else{
            V[c] = VV[j];
            c += 1;
        }
    }
    return V;
}
```

7 La fonction `matvec` correspond un produit entre matrice et vecteur, cette fonction renvoit un vecteur $W = AV$ qui est le résultat du produit de matrice A et vecteur V quelconque. Ici, A est la matrice d'assemblage de la matrice GrdGrd obtenu avec la fonction `GradGrad` précédente.

```
vector<double> matvec(vector<double> V, double a,double b,int N,int M){
    vector<double> VV = extendVec(V,N,M);
    vector<double> WW = vector<double>((N+1)*(M+1));
    vector<Triangle> TRG = maillageTR(N,M);  //triangle
    int s;
    int r;
    for (int t=0;t<2*N*M;t++){
        vector<vector<double>> GrdGrd = GradGrad(a,b,M,TRG[t],&eta);
        for (int i=0;i<3;i++){
            if (i==0){
                s = TRG[t].a;
            }
            if (i==1){
                s = TRG[t].b;
            }
            if (i==2){
                s = TRG[t].c;
            }
            double res = 0;  //resultat
```

```
            for (int j=0;j<3;j++){
                if (j==0){
                    r = TRG[t].a;
                }
                if (j==1){
                    r = TRG[t].b;
                }
                if (j==2){
                    r = TRG[t].c;
                }
                double PROD2 = GrdGrd[i][j];
                res += VV[r]*PROD2;  //produit par element
            }
            WW[s] += res;
        }
    }
    vector<double> W = IntVec(WW,N,M);
    return W;
}
```

8 On s'intéresse maintenant au calcul du second membre B. Dans cette question, on a utilisé une méthode un peu différente que l'énoncé.
On sait que

$$(B)_i = \int_\Omega f w_i \, d\Omega$$

et pour la fonction f, on peut l'approcher par

$$\tilde{f} = \sum_{j=1}^{G} f(S_j) w_j$$

où S_j est le sommet du triangle.
Comme ça on peut avoir

$$(B)_i \approx \int_\Omega \tilde{f} w_i \, d\Omega = \sum_{j=1}^{G} f(S_j) w_j \int_\Omega w_i \, d\Omega = \sum_{j=1}^{G} f(S_j) \int_\Omega w_j w_i \, d\Omega$$

où on peut représenter $\int_\Omega w_j w_i \, d\Omega$ sous forme matricielle. Avant avoir la matrice d'assemblage, on doit calculer la matrice élémentaire M_{elem} d'abord.

$$\int_\Omega w_j w_i \, d\Omega = \int_T \lambda_j \lambda_i \, dxdy = aire(\hat{T}) \int_{\hat{T}} \hat{\lambda}_i \hat{\lambda}_j \, d\hat{x} d\hat{y}$$

Si $i = j$ (par exemple $i = j = 2$)

$$\int_{\hat{T}} \hat{\lambda}_2^{\,2} \; d\hat{x}d\hat{y} = \int_{\hat{T}} x^2 \; dxdy = \int_0^1 \int_0^{1-x} x^2 \; dxdy = \int_0^1 (1-x)x^2 dx = [\frac{x^3}{3} - \frac{x^4}{4}]_0^1 = \frac{1}{12}$$

Si $i \neq j$ (par exemple $i = 3, j = 2$)

$$\int_{\hat{T}} \hat{\lambda}_2^{\,2} \; d\hat{x}d\hat{y} = \int_{\hat{T}} xy \; dxdy = \int_0^1 (\int_0^{1-x} ydy)xdx = \frac{1}{2}\int_0^1 (1-x)^2 xdx = \frac{1}{24}$$

Les calculs sont similaires pour les autres combinaisons. Donc

$$M_{elem} = \frac{|DetB_T|}{24} \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

Donc le code est suivant:

```
vector<vector<double>> mat_elem(double a,double b,int N,int M, const Triangle &t)
    vector<Point> St = sommetTR(a,b,N,M,t);
    double D = abs(DetBT(a,b,M,t));  //valeur absolu du determinant de BT
    vector<vector<double>> mat(3,vector<double>(3));
    for (int i=0;i<3;i++){
        for (int j=0;j<3;j++){
            if (i==j){
                mat[i][j] = 1./12*D;
            }
            else{
                mat[i][j] = 1./24*D;
            }
        }
    }
    return mat;
}
```

Ensuite, la matrice d'assemblage M peut être obtenue facilement.

```
vector<vector<double>> mat_assem(double a,double b,int N,int M){
    vector<Triangle> TRG = maillageTR(N,M);  //triangle
    vector<vector<double>> MM((N+1)*(M+1),vector<double>((N+1)*(M+1)));
    for(int t=0;t<TRG.size();t++){
        double k = TRG[t].a;   //numero global du 1e sommet
        double l = TRG[t].b;   //numero global du 2e sommet
        double m = TRG[t].c;   //numero global du 3e sommet
        vector<vector<double>> mat = mat_elem(a,b,N,M,TRG[t]);
```

```
        MM[k][k] += mat[0][0];
        MM[l][l] += mat[1][1];
        MM[m][m] += mat[2][2];
        MM[k][l] += mat[0][1];
        MM[k][m] += mat[0][2];
        MM[l][m] += mat[1][2];
        MM[l][k] += mat[1][0];
        MM[m][k] += mat[2][0];
        MM[m][l] += mat[2][1];
    }
    return MM;
}
```

Après, on peut calculer facilement que le seconde membre

$$(B)_i = (MF)_i$$

où M est la matriced'assemblage et $(F)_i = f(Si)$.

```
vector<double> scdmembre(double a,double b,int N,int M,
double (*rhsf)(double x,double y)){
    vector<vector<double>> MM = mat_assem(a,b,N,M);  //matrice d'assemblage
    vector<double> F = vector<double>((N+1)*(M+1));
    for (int i=0;i<F.size();i++){
        Point P = point_exact(a,b,N,M,i);  //sommet exact
        double x = P.x;
        double y = P.y;
        F[i] = rhsf(x,y);
    }
    vector<double> B = IntVec(pmv(MM,F),N,M);//pmv est produit matrice vecteur
    return B;
}
```

9 Cette question consiste à calculer la normr L^2 du vectuer $V \in \mathbb{R}^I$ ($\|\nabla v\|_{L^2(\Omega)}$) par la fonction `normL2Grad`. Pour ce faire, on peut prendre la sortie d'un vecteur $W = (V^t A_0 V)^{\frac{1}{2}}$, où A_0 est la matrice A quand $\eta = 1$, et on peut appliquer la fonction `matvec` précédente avec $\eta = 1$ pour calculer le produit $A_0 V$.

```
double normL2Grad(vector<double> V,double a,double b,int N,int M){
    vector<double> W = matvec(V,a,b,N,M);
    return sqrt(ps(V,W));  //ps est produit scalaire du V et W
}
```

10

Démonstration.

$$\begin{aligned}
\int_\Omega v(x,y)^2 dxdy &= \sum_{T\in\mathcal{T}} \int_T v(x,y)^2 dxdy \\
&= \sum_{T\in\mathcal{T}} \int_T \sum_{s=0}^{2}\sum_{j=0}^{2} v_{m(s)} w_s(x,y) v_{m(j)} w_j(x,y) dxdy \\
&= \sum_{T\in\mathcal{T}} \sum_{s=0}^{2}\sum_{j=0}^{2} \int_T v_{m(s)} w_s(x,y) v_{m(j)} w_j(x,y) dxdy
\end{aligned}$$

Par précédent, si (x,y) est un sommet de T, alors la restriction de w_j à T est égal à λ_j et $v_{k(j)} = \hat{v}_{k(j)}$. Sinon, la restriction de w_j à T est nulle et $v_{k(j)} = 0$. On a donc

$$\sum_{T\in\mathcal{T}} \sum_{s=0}^{2}\sum_{j=0}^{2} \int_T v_{m(s)} w_s(x,y) v_{m(j)} w_j(x,y) dxdy = \sum_{T\in\mathcal{T}} \sum_{s=0}^{2}\sum_{j=0}^{2} \hat{v}_{m(s)} \hat{v}_{m(j)} \int_T \lambda_s(x,y)\lambda_j(x,y) dxdy$$

□

11 Pour calculerla norme $\|v\|_{L^2(\Omega)}$, il suffit d'utuliser la formule de la question précédente, et la partie $\int_T \lambda_s(x,y)\lambda_j(x,y)\ dxdy$, il suffit d'utiliser la valeur de la question (8), comme ça on peut facilement coder la fonction `normL2`.

```
double normL2(vector<double> V,double a,double b,int N,int M){
    vector<Triangle> TRG = maillageTR(N,M);  //triangles
    vector<double> VV = extendVec(V,N,M);
    int s;
    int r;
    double res = 0;
    for(int t=0;t<2*N*M;t++){
        double D = abs(DetBT(a,b,M,TRG[t]));
        for (int i=0;i<3;i++){
            if (i==0){
                s = TRG[t].a;
            }
            if (i==1){
                s = TRG[t].b;
            }
            if (i==2){
                s = TRG[t].c;
            }
            for (int j=0;j<3;j++){
```

```
                    if (j==0){
                        r = TRG[t].a;
                    }
                    if (j==1){
                        r = TRG[t].b;
                    }
                    if (j==2){
                        r = TRG[t].c;
                    }
                    if (i==j){
                        res += VV[s]*VV[r]*1./12*D; //formule
                    }
                    else{
                        res += VV[s]*VV[r]*1./24*D; //formule
                    }
                }
            }
        }
        return res;
}
```

4 Inversion du système

1

Démonstration: Supposons $AX = B$, montrer que pour tout $Y \in \mathbb{R}^I$, $G(Y) - G(X) \geq 0$.

$$\begin{aligned}\forall Y \in \mathbb{R}^I, G(Y) - G(X) &= \frac{1}{2}Y^TAY + B^TY - \frac{1}{2}X^TAX - B^TX \\ &= \frac{1}{2}Y^TAY - \frac{1}{2}X^TAX + B^T(Y - X)\end{aligned}$$

Comme $AX = B$,

$$\begin{aligned}\frac{1}{2}Y^TAY - \frac{1}{2}X^TAX + B^T(Y - X) &= \frac{1}{2}Y^TAY - \frac{1}{2}X^TAX + (AX)^T(Y - X) \\ &= \frac{1}{2}Y^TAY + \frac{1}{2}X^TAX - (AX)^TY\end{aligned}$$

Comme A est symétrique, $A^T = A$, donc

$$
\begin{aligned}
\frac{1}{2}Y^TAY + \frac{1}{2}X^TAX - (AX)^TY &= \frac{1}{2}Y^TAY + \frac{1}{2}X^TAX - (AX)^TY \\
&= \frac{1}{2}Y^TAY + \frac{1}{2}X^TAX - X^TAY \\
&= \frac{1}{2}Y^TAY + \frac{1}{2}X^TAX - \frac{1}{2}X^TAY - \frac{1}{2}X^TAY \\
&= \frac{1}{2}(Y^T - X^T)AY + \frac{1}{2}X^TA(X - Y) \\
&= \frac{1}{2}(Y - X)^TAx + \frac{1}{2}X^TA(Y - X) \\
&= \frac{1}{2}(Y - X)^TA(Y - X)
\end{aligned}
$$

Comme A est définie positive, on a donc $\frac{1}{2}(Y-X)^TA(Y-X) \geq 0$, donc $G(Y) - G(X) \geq 0, \forall Y \in \mathbb{R}^I$.

Inversement, si $G(Y) - G(X) \geq 0$, c'est-à-dire X minimisant G. Posons

$$g(t) = G(X + tZ)$$

$$\forall t \in R, g(0) \leq g(t), g'(0) = 0$$

Par définition,

$$
\begin{aligned}
g'(0) &= \lim_{t\to 0} \frac{g(t) - g(0)}{t} \\
&= \lim_{t\to 0} \frac{G(X + tZ) - G(X)}{t} \\
&= \lim_{t\to 0} \frac{\frac{1}{2}(X + tZ)^TA(X + tZ) - B^T(X + tZ) - \frac{1}{2}X^TAX + B^TX}{t} \\
&= \lim_{t\to 0} \frac{\frac{1}{2}(X + tZ)^TAX + \frac{t}{2}(X + tZ)^TAZ - \frac{1}{2}X^TAX - tB^TZ}{t} \\
&= \lim_{t\to 0} \frac{\frac{1}{2}X^TAX + \frac{t}{2}Z^TAX + \frac{t}{2}X^TAZ + \frac{t^2}{2}Z^TAZ - \frac{1}{2}X^TAX - tB^TZ}{t} \\
&= \lim_{t\to 0} \frac{tX^TAZ + \frac{t^2}{2}Z^TAZ - tB^TZ}{t} \\
&= \lim_{t\to 0} (X^TAZ + \frac{t}{2}Z^TAZ - B^TZ) = X^TAZ - B^TZ = 0
\end{aligned}
$$

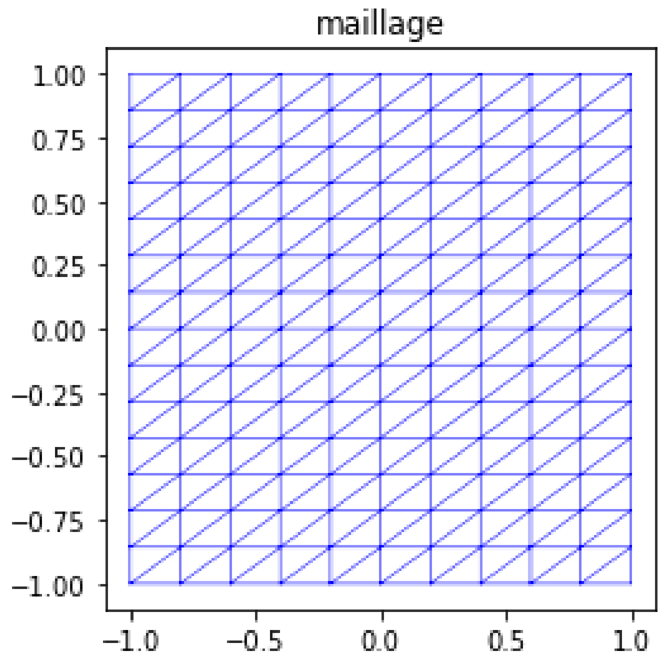
On a donc $AX = B$, d'où $X = A^{-1}B$. $\square$

2 codage:

```
vector<double> Grad_conj(vector<double> B, double a,double b,int N,int M){
    double eps = 1e-8;  //erreur
    vector<double> x = vector<double>(B.size());  //vecteur initial
    for (int i=0;i<B.size();i++){          // vectuer qui ne contient que des 1
        x[i] = 1;
    }
    vector<double> y = matvec(x,a,b,N,M);  //vecteur initial
    for (int i=0;i<10000;i++){
        double lamb = ps(B-matvec(x,a,b,N,M),y)/ps(y,matvec(y,a,b,N,M));//lambda
        x = x+lamb*y;
        double beta =
        ps(matvec(x,a,b,N,M)-B,matvec(y,a,b,N,M))/ps(y,matvec(y,a,b,N,M));//beta
        y = B-matvec(x,a,b,N,M)+beta*y;
        if (norme(matvec(x,a,b,N,M)-B) < eps){
            return x;
        }
        if (i==9999){
            // ne converge pas
            return 0*x;
        }
    }
}
```

5 Testes Numériques

1 Visualisation du maillage:



2 $h_1 = \frac{1}{5}, h_2 = \frac{1}{7}$. $i = E[(x+a)/h_1] = E[5(x+1)]$, $j = E[(y+b)/h_2] = E[7(y+1)]$. $(x,y) \in \Omega =]-1,1[\times]-1,1[$. Donc $x+1 \in]0,2[$, $5(x+1) \in]0,10[$, $7(y+1) \in]0,14[$, donc $5(x+1) < 10$ et $7(y+1) < 14$. On a donc $E(5(x+1)) \leq 9$ et $E(7(y+1)) \leq 13$, c'est-à-dire $i \leq 9$ et $j \leq 13$.

Donc, les x_i et les y_j désigne une numérotation de l'interval $]-1,1[\times]-1,1[$, qui est 10×14 rectangles.

Soit $(x,y) \in \Omega$, on fait la division euclidienne de $x+a$ par $\frac{2a}{N}$, le quotient vaut i, donc on a $x_i \leq x < x_{i+1}$. On fait la division euclidienne de $y+b$ par $\frac{2b}{M}$, le quotient vaut j, donc on a $y_j \leq y < y_{j+1}$.

Posons $l = (1-(-1))(jN+i) = 2jN+2i$. D'après la définition, l est le numéro global de (i,j), le triangle T_l et T_{l+1} sont des triangles, donc $T_l \cup T_{l+1}$ est un rectangle. $T_l \cup T_{l+1}$=$[x_i, x_{i+1}[\times[y_j, y_{j+1}[$. Donc $(x,y) \in [y_j, y_{j+1}[$ implique $(x,y) \in T_l$ ou $(x,y) \in T_{l+1}$.

3

(a) On a $\eta = 2 - \sin(x+2y)$ et $u(x,y) = \sin(k_0\pi x)\sin(k_0\pi y)$, donc on peut avoir

$$\nabla\eta = \begin{pmatrix} -\cos(x+2y) \\ -2\cos(x+2y) \end{pmatrix} \qquad \nabla u = \begin{pmatrix} k_0\pi\cos(k_0\pi x)\sin(k_0\pi y) \\ k_0\pi\sin(k_0\pi x)\cos(k_0\pi y) \end{pmatrix}$$

et donc

$$\Delta u = -2k_0^2\pi^2\sin(k_0\pi x)\sin(k_0\pi y)$$

Ensuite, par ces formule, on peut avoir

$$\begin{aligned} f &= -div(\eta\nabla u) \\ &= -\nabla\eta\nabla u + \eta\Delta u \\ &= (2-\sin(x+2y))(2k_0^2\pi^2\sin(k_0\pi x)\sin(k_0\pi y)) \\ &- k_0\pi\cos(x+2y)(cos(k_0\pi x)\sin(k_0\pi y) + 2\sin(k_0\pi x)\cos(k_0\pi y)) \end{aligned}$$

(b) Comme on sait déjà $u(x,y)$, on peut facilement coder la fonction `solExa` en utilisant la fonction `u`

```
double u(double x,double y){
    double k0 = 2;
    return sin(k0*PI*x)*sin(k0*PI*y);
}

vector<double> solExa(double a,double b,int N,int M){
    vector<double> sol = vector<double>((N+1)*(M+1));
    Point P;
    double x;
    double y;
    for(int i=0;i<(N+1)*(M+1);i++){
        P = point_exact(a,b,N,M,i);  //coordonnee exacte des noeuds
        x = P.x;
        y = P.y;
        sol[i] = u(x,y);
    }
    return sol;
}
```

Ensuite, on est capable de calculer les erreurs en écrivant la fonction `erreurs`.

```
Triangle erreurs(double a,double b,int N,int M){
    Triangle err;   //triple pour stocker les erreurs
    vector<double> B = scdmembre(a,b,N,M,&rhsf); //second mebre
    vector<double> sol_app = Grad_conj(B,a,b,N,M);  //solution approchee
    vector<double> sol_exa = solExa(a,b,N,M);   //solution exacte
    vector<double> err_sol = sol_app-sol_exa;  //difference entre les solutions
    err.a = normL2(err_sol,a,b,N,M)/normL2(sol_app,a,b,N,M);  //e0
    err.b = normL2Grad(err_sol,a,b,N,M)/normL2Grad(sol_app,a,b,N,M); //e1
    err.c = norm_inf(err_sol)/norm_inf(sol_app);  //e2
    return err;
}
```

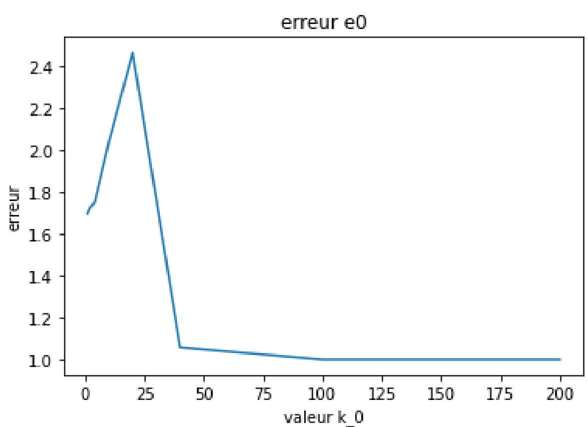
pour $k_0 = 2$, on a `err = [1.37314,1.95899,1.96264]`.

(c)

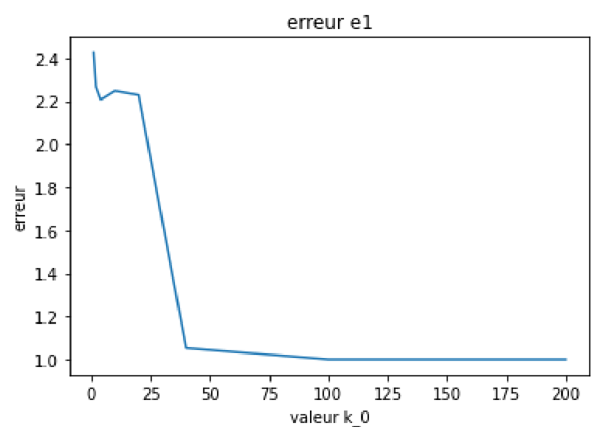
(d) dans cette question on s'intéresse au comportement des erreurs quand la valeur de k_0 varie. On va tester la valeur de k_0 dans le vecteur $E = [1, 2, 4, 10, 20, 40, 100, 200]$ et ici $N = 50$, $M = 50$. On obtient

```
e1 = [1.69537,2.42665,1.87941]
e2 = [1.72362,2.26947,1.94408]
e4 = [1.74977,2.20859,1.96729]
e10 = [2.03893,2.24953,2.34528]
e20 = [2.46284,2.23012,2.64782]
e40 = [1.05696,1.05331,1.08465]
e100 = [1,1,1]
e200 = [1,1,1]
```

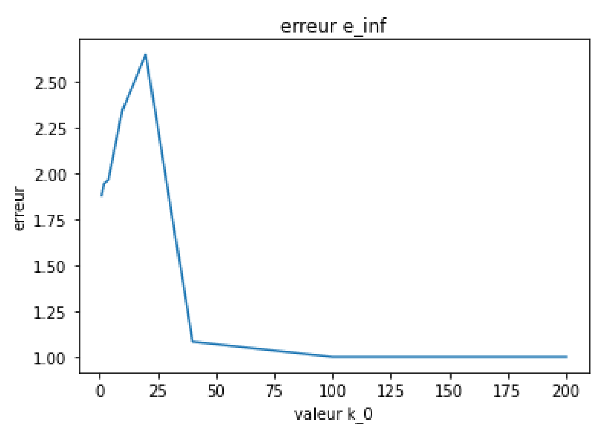
Si on les représente graphiquement, on obtient



(a) e0



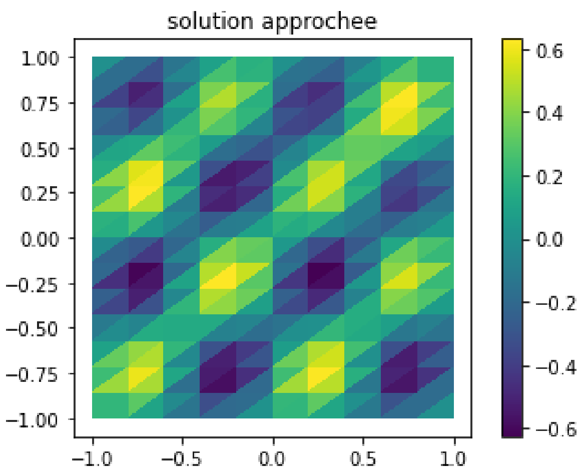
(b) e1



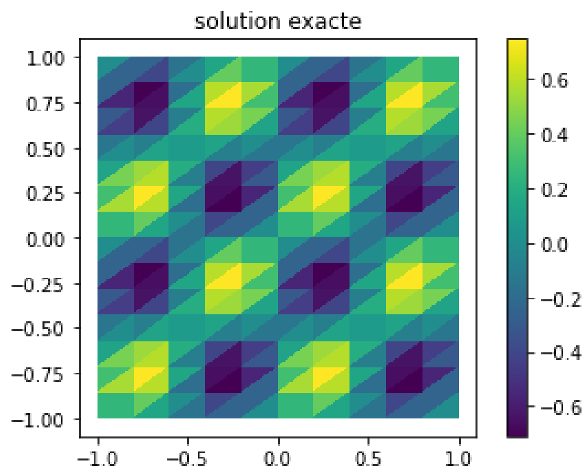
(c) e_inf

6 Résultat

Dans cette partie, on va montrer notre résultat graphiquement.
Avec les donnée dans la partie précédente et $k_0 = 2$ dans notre cas, la solution appréchée et la solution exacte nous montrent

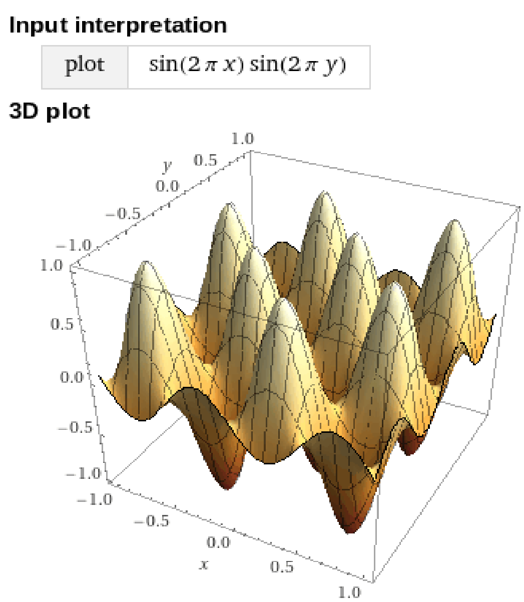


(d) solution approchée



(e) solution exacte

on peut voir que il n'y a pas trop de différence entre les 2 figures. De plus, on peut voir graphiquement la solution exacte en 3D:



ça montre notre résultat n'est pas faux.

7 Annexe

Dans cette partie, on va mettre les fonctions qu'on a utilisé mais pas encore présentée.

```
vector<double> operator + (vector<double> u ,vector<double> v)
//addition vectorielle
{
    vector<double> r =  vector<double>(u.size());
    for (int i=0;i<r.size();i++){
        r[i] = u[i]+v[i];
    }
    return r;
}

vector<double> operator - ( vector<double> u , vector<double> v)
//soustraction vectorielle
{
    vector<double> r =  vector<double>(u.size());
    for (int i=0;i<r.size();i++){
        r[i] = u[i]-v[i];
    }
    return r;
}

vector<double> operator * (double a , vector<double> u)
//multiplication reel vecteur
{
    vector<double> r =  vector<double>(u.size());
    for (int i=0;i<r.size();i++){
        r[i] = u[i]*a;
    }
    return r ;
}


ostream & operator <<(ostream & os , const Point &P)
//operation pour print le Point
{
os<<"("<<P.x<<","<<P.y<<")" ;
return os ;
}

```

```
void print_v(vector<double> A)    //affichage de vecteur
{
    cout<<"[";
    for( int i =0; i<A.size()-1 ; i++){
        cout<<A[i]<<"," ;
    }
    cout<<A[A.size()-1]<<"]";
}

void print_m(vector<vector<double>> A)    //affichage de matrice
{
    for( int i =0; i<A.size() ; i++){
        cout<<"ligne"<<i<<": " ;
        for ( int j =0; j<A.size() ; j++){
            cout<<A[i][j]<<" " ;
        }
    cout<<endl ;
    }
}


double ps(vector<double> u,vector<double> v)     //produit scalaire
{
    double r=0;
    for(int i=0;i<v.size();i++){
        r+=u[i]*v[i];
    }
    return r;
}

vector<double> pmv(vector<vector<double>> A, vector<double> v)
// produit matrice vecteur
{
    vector<double> u = vector<double>(v.size());
    for (int i=0;i<v.size();i++){
        u[i] = ps(A[i],v);
    }
    return u;
}

double norme(vector<double> u)   // norme 2
```

```
{
    double r = 0;
    for(int i=0;i<u.size();i++){
        r += u[i]*u[i];
    }
    r = sqrt(r);
    return r;
}

double norm_inf(vector<double> u) //norme infinie
{
    double max = -1.;
    for (int i=0;i<u.size();i++){
        if (abs(u[i])>max){
            max = abs(u[i]);
        }
    }
    return max;
}

ostream & operator <<(ostream &os, const Triangle &T){
    //operation pour print les triangles
    os<<"["<<T.a<<","<<T.b<<","<<T.c<<"]" ;
    return os ;
}

void print_t(vector<Triangle> A)  //affichage du triangle
{
    cout<<"[";
    for( int i =0; i<A.size()-1 ; i++){
        cout<<A[i]<<"," ;
    }
    cout<<A[A.size()-1]<<"]";
}
```

References

[1] Wikipédia, Intégrale multiple.