Sup-Galilée

discrétisation de l'équation de Poisson par différences finies et éléments finis

Encadré par : Emanuel Audusse
Réalisé par : Adem ZOUAOUI
Omar BOUZEKRI

Table des matières

Introduction

Les équations aux dérivés partielles (EDP) sont omniprésentes dans toutes les sciences, les systèmes et phénomènes physiques les plus intéressants sont aussi les plus complexes à étudier. Ils sont souvent régis par un grand nombre de paramètres non-linéaires intéragissant entre eux. Les solutions analytiques étant exeptionnelles, des solutions approchées sont obtenues par des méthodes numériques qui reposent sur un modèle mathématiques qui est une représentation ou une interprétation abstraite de la réalité physique qui est accessible à l'analyse et au calcul.

L'équation de Poisson ainsi nommée en l'honneur du mathématicien et physicien français Siméon Denis Poisson est une équation aux dérivées partielles parabolique du second ordre avec un opérateur Laplacien et une fonction généralement donnée. Ce projet approche cette équation de deux méthode différentes : par différences finise et éléments finis. Deux différentes méthodes car la première est utilisée pour la résolution numérique des équations différentielles, tout particulièrement pour les problèmes de conditions aux limites ; et la deuxième surexploitent des approximations d'intégrales. Toutefois, la méthode des éléments finis se base sur les problèmes sous leur forme variationnelle.

Pour la résolution numérique en différences finies, nous allons utilisé le logiciel Matlab car nous allons faire une analyse numerique matricielle pour la résolution des systèmes linéaires. En éléments finis, le logiciel Freefem qui permet la résolution de la formulation variationnelle.

Chapitre 1

Discrétisation par différences finies

La méthode consiste à remplacer les dérivées partielles par des différences divisées ou conbinaisons de valeurs ponctuelles de la fonction en un nombre de points discrets ou noeuds du maillage.
Avantages : grande simplicité d'écriture et faible coût de calcul.
Inconvénients : limitation à des géométries simples, difficultés de prise en compte des conditions aux limites de type Neumann.

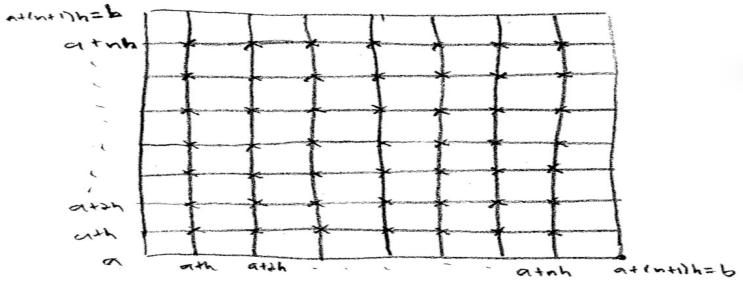
1.1 Discrétisation de l'équation de Poisson

Considérons le problème bidimensionnel stationnaire. Le domaine de calcul est discrétisé en (N+1)*(P+1) noeuds (x_i,y_j) (i variant de 0 à N et j variant de 0 à P). On supposera que les pas d'espace dans chaque direction $\triangle x$ et $\triangle y$ sont constants. L'équation de Poisson en 2D est :

$$\boxed{\triangle u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y)}$$

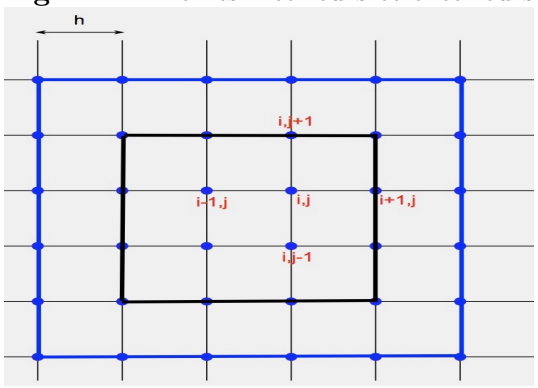
On commence par discrétiser le domaine uniformément sur (a,b) :

Fig. 1.1 –



Puis on divise le domaine en deux parties : Points intérieurs en noir et extrieurs en bleu

Fig. 1.2 – Points intérieurs et extérieurs



Pour les points intérieurs nous utilisons un schéma centré d'ordre 2 pour approximer les dérivées secondes en espace :

$$(\frac{\partial^2 u}{\partial x^2})_{(i,j)} = \frac{u_{(i+1,j)} - 2 * u_{(i,j)} + u_{(i-1,j)}}{\triangle x^2}$$

$$(\frac{\partial^2 u}{\partial y^2})_{(i,j)} = \frac{u_{(i,j+1)} - 2 * u_{(i,j)} + u_{(i,j-1)}}{\triangle y^2}$$

La formulation discrétisée est alors, pour i variant de 1 à N-1 et j variant de 1 à P-1 :

$$\boxed{\frac{u_{(i+1,j)} - 2 * u_{(i,j)} + u_{(i-1,j)}}{\triangle x^2} + \frac{u_{(i,j+1)} - 2 * u_{(i,j)} + u_{(i,j-1)}}{\triangle y^2} = F_{(i,j)}}$$

Soit sous forme matricielle. Pour N=P=4 et c= $\triangle x^2 + \triangle y^2$, on a le système suivant :

$$A * U = b$$

avec

$$A = \begin{pmatrix} -2c & \triangle y^2 & 0 & \triangle x^2 & 0 & 0 & 0 & 0 & 0 \\ \triangle y^2 & \text{-2c} & \triangle y^2 & 0 & \triangle x^2 & 0 & 0 & 0 & 0 \\ 0 & \triangle y^2 & \text{-2c} & 0 & 0 & \triangle x^2 & 0 & 0 & 0 \\ \triangle x^2 & 0 & 0 & \text{-2c} & \triangle y^2 & 0 & \triangle x^2 & 0 & 0 \\ 0 & \triangle x^2 & 0 & \triangle y^2 & \text{-2c} & \triangle y^2 & 0 & \triangle x^2 & 0 \\ 0 & 0 & \triangle x^2 & 0 & \triangle y^2 & \text{-2c} & 0 & 0 & \triangle x^2 \\ 0 & 0 & 0 & \triangle x^2 & 0 & 0 & -2c & \triangle y^2 & 0 \\ 0 & 0 & 0 & 0 & \triangle x^2 & 0 & \triangle y^2 & -2c & \triangle y^2 \\ 0 & 0 & 0 & 0 & 0 & \triangle x^2 & 0 & \triangle y^2 & \text{-2c} \end{pmatrix}$$

et

$$\text{U}= \begin{pmatrix} u_{1,1} \\ u_{1,2} \\ u_{1,3} \\ u_{2,1} \\ u_{2,2} \\ u_{2,3} \\ u_{3,1} \\ u_{3,2} \\ u_{3,3} \end{pmatrix} \text{ et } \boxed{b = f - Ub}$$

Dans nôtre cas les pas d'espace sont identiques $\triangle x^2 = \triangle y^2$, la formulation devient, pour i variant de 1 à N-1 et j variant de 1 à P-1 :

$$\boxed{u_{(i+1,j)} - 4 * u_{(i,j)} + u_{(i-1,j)} + u_{(i,j+1)} + u_{(i,j-1)} = f_{(i,j)}}$$

Soit sous forme matricielle, pour N=P=4 :

$$\text{A} = \begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix}$$

et

$$\text{U}= \begin{pmatrix} u_{1,1} \\ u_{1,2} \\ u_{1,3} \\ u_{2,1} \\ u_{2,2} \\ u_{2,3} \\ u_{3,1} \\ u_{3,2} \\ u_{3,3} \end{pmatrix} \text{ et } \boxed{b = f - Ub} \text{ avec f}= \begin{pmatrix} f_{1,1} \\ f_{1,2} \\ f_{1,3} \\ f_{2,1} \\ f_{2,2} \\ f_{2,3} \\ f_{3,1} \\ f_{3,2} \\ f_{3,3} \end{pmatrix} \text{ et Ub}= \begin{pmatrix} u_{1,0} + u_{0,1} \\ u_{0,2} \\ u_{0,3} + u_{1,4} \\ u_{2,0} \\ 0 \\ u_{2,4} \\ u_{3,0} + u_{4,1} \\ u_{4,2} \\ u_{3,4} + u_{4,3} \end{pmatrix}$$

Ub la solution exacte dans les points extérieurs (figure 1.2)

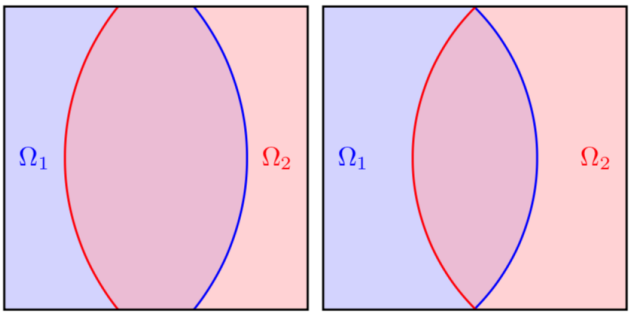
1.2 Décomposition de domaine de type Schwarz

les méthodes de décomposition de domaine de type Schwarz consiste à remplacer la résolution d'une EDP posée sur un domaine Ω potentiellement gros et compliqué par une succession de résolutions de la même EDP sur des sous-domaines de Ω, notés Ω_i, supposément plus simples, plus petits, etc ... Evidemment, ceci ne peut se faire tout à fait aisément et il s'agit de procéder à des résolutions itératives sur les sous-domaines, choisies de telle sorte que la solution exacte du problème complet soit obtenue à convergence.

1.2.1 Méthode avec recouvrement

On commence par étudier le cas où les sous-domaines de Ω se recouvrent, par exemple de l'une des deux façons indiquées dans la figure suivante :

Fig. 1.3 – Décomposition du carré unité en deux sous-domaines qui se recouvrent



(a) Les interfaces sont disjointes (b) Les interfaces se croisent

1.2.2 Principe général

Etant donnée une parition de Ω de la forme donnée dans la figure 1.3, l'algorithme de Schwarz le plus élémentaire consiste à construire des suites $(u_n^1)_n \subset H1(\Omega 1) et (u_n^2)_n \subset H1(\Omega 2)$ de la facon suivante :

— Choisir des initialisations $u_0^1 et u_0^2$. ;

— Pour tout n$\geq$ 0, effectuer de façon successive :

□Calcul de $u_{n+1}^1 \in H1(\Omega 1)$ solution du problème suivant :

$$\begin{cases} \triangle u_{n+1}^1 = f, dans \Omega 1 \\ u_{n+1}^1 = g, sur \partial\Omega \cap \partial\Omega 1, \\ u_{n+1}^1 = u_n^2, sur \omega_{12}. \end{cases}$$

□Calcul de $u_{n+1}^2 \in H1(\Omega 1)$ solution du problème suivant :

$$\begin{cases} \triangle u_{n+1}^2 = f, dans \Omega 2 \\ u_{n+1}^2 = g, sur \partial\Omega \cap \partial\Omega 2, \\ u_{n+1}^2 = u_{n+1}^1, sur \omega_{21}. \end{cases}$$

On note ω_{12} (resp. ω_{21}) la partie du bord de $\partial\Omega 1$ (resp. $\partial\Omega 2$) qui est contenue dans $\bar{\Omega}2$ (resp. $\bar{\Omega}1$).

— S'arrêter à convergence (si celle-ci se produit !), i.e. quand u_n^1 et u_n^2 sont suffisamment proches sur $\Omega 1 \cap \Omega 2$ par exemple.

1.3 Application

Prenons le cas de la solution exacte :

$$uex(x,y) = exp(k1 * x + k2 * y). * cos(a1 * 2 * \pi * x). * sin(a2 * 2 * \pi * y);$$

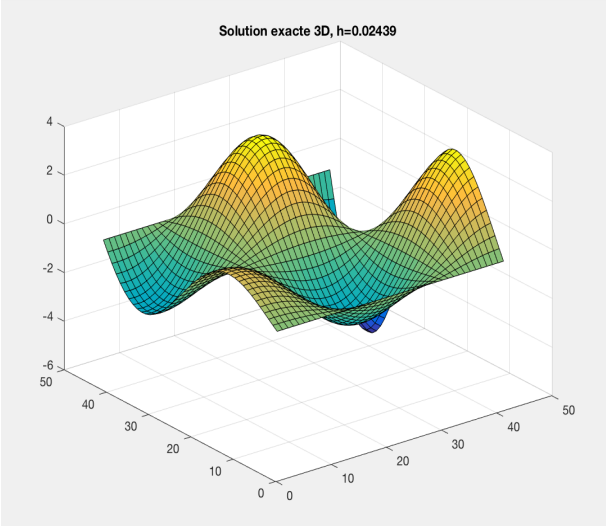
avec : k1=k2=a1=a2=1

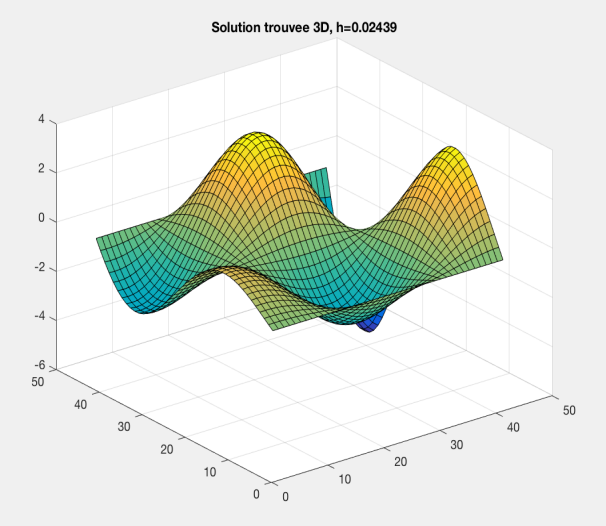
En dérivant deux fois par rapport à x et à y, on obtient :

$\frac{\partial^2 uex}{\partial x^2} + \frac{\partial^2 uex}{\partial y^2} = f(x,y) = 2 * exp(x + y). * cos(2 * \pi * x). * sin(2 * \pi * y) + 4 * \pi * exp(x + y). * cos(2 * \pi * x). * cos(2 * \pi * y) - 4 * pi * exp(x + y). * sin(2 * \pi * x). * sin(2 * \pi * y) - 8 * \pi^2 * exp(x + y). * cos(2 * \pi * x). * sin(2 * \pi * y)$

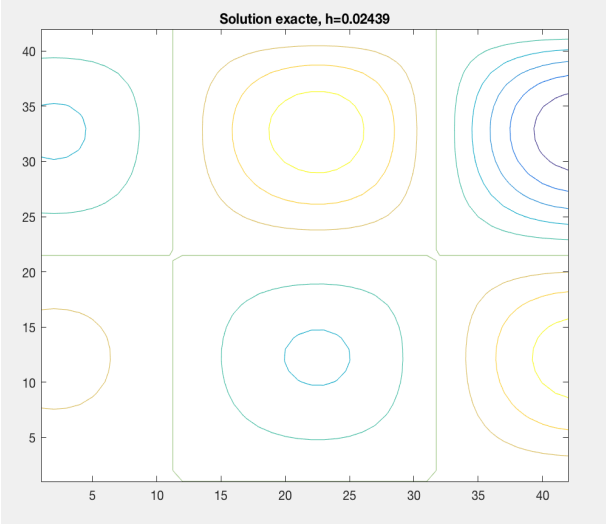
En prenant le nombre de points 40 et le maillage sur (0 à 1), on obtient sur un carre :

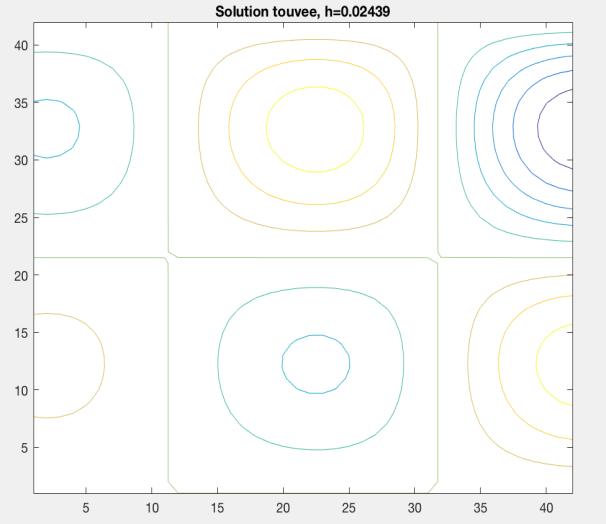
Solutions 3D :





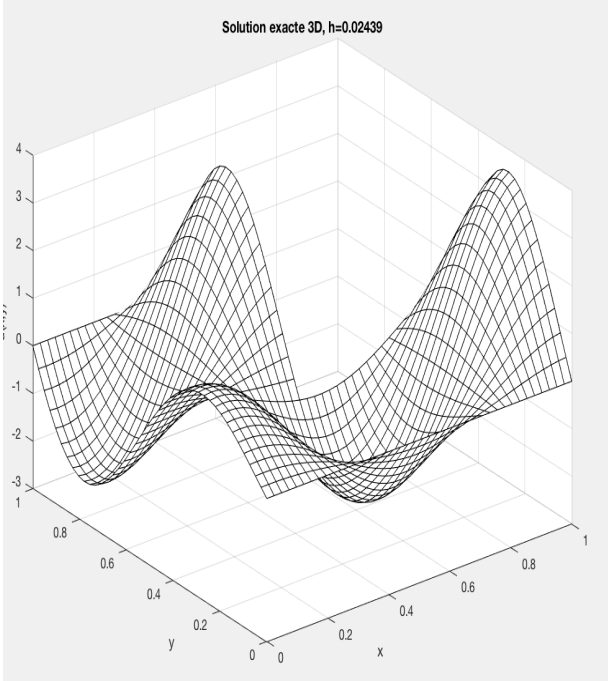
Solutions 2D :

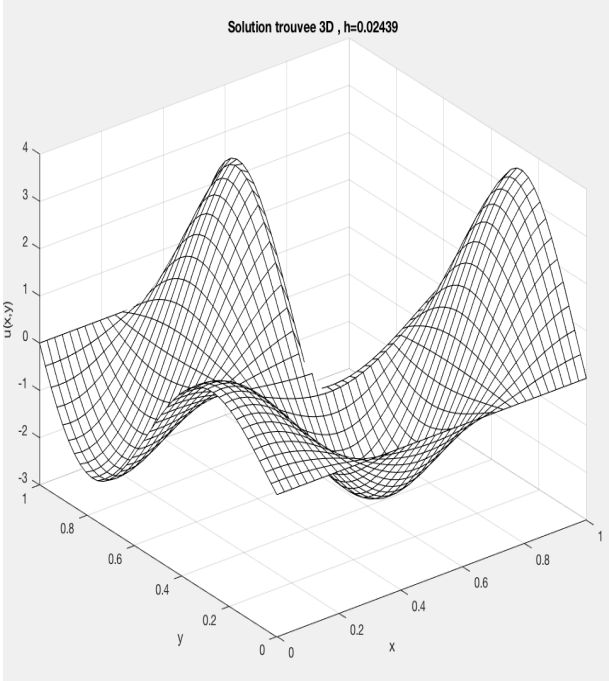




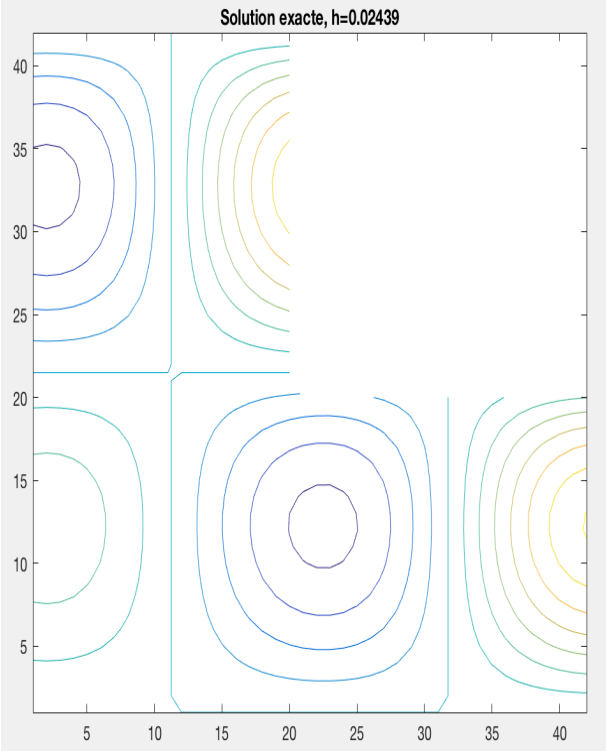
En changeant la géométrie, avec les mêmes hypothèses ; on obtient sur un L :

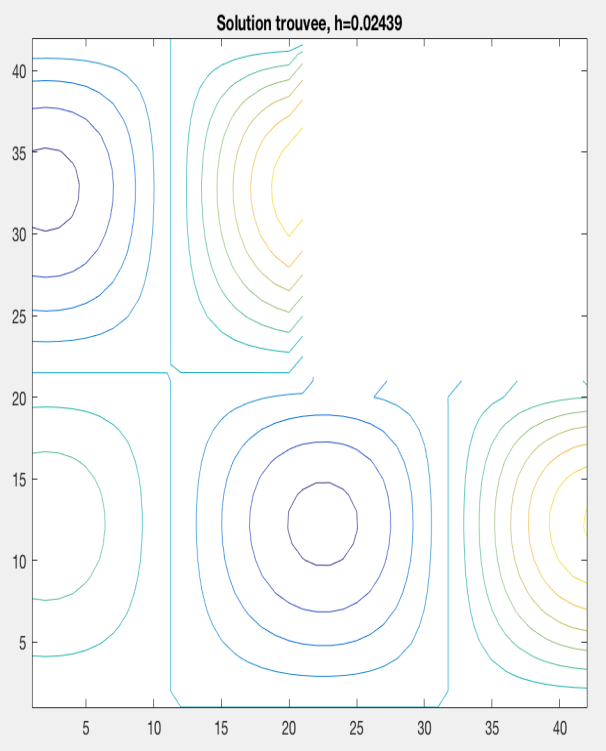
Solutions 3D :



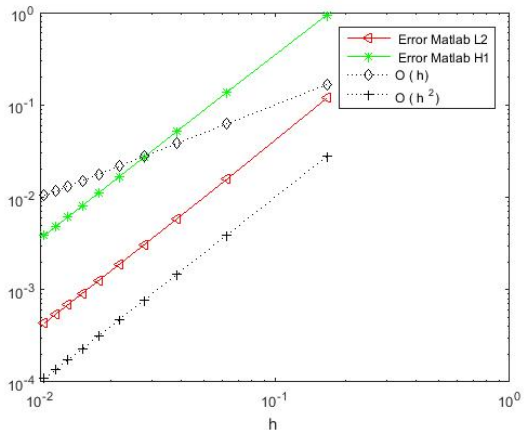


Solutions 2D :

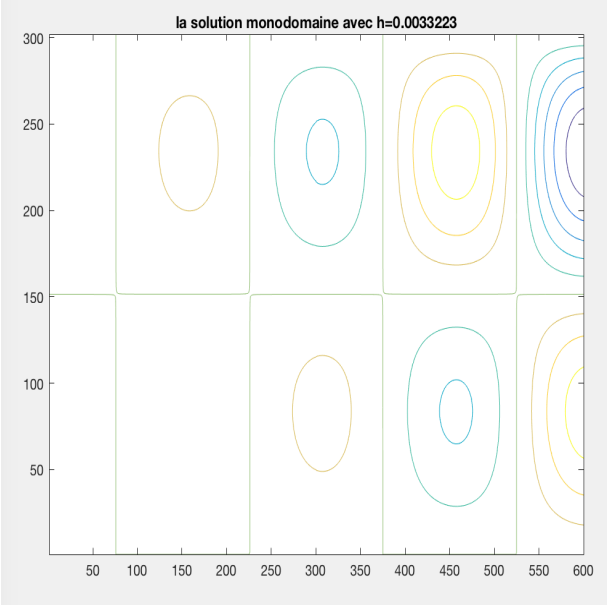


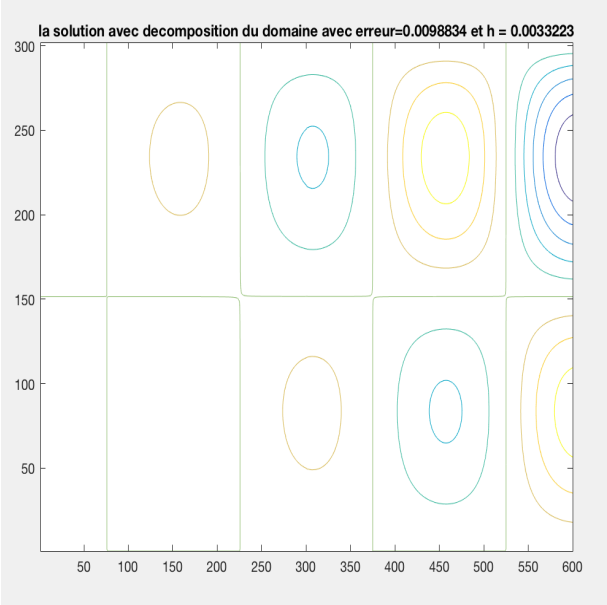


On obtient une erreur d'ordre 10^-3 :

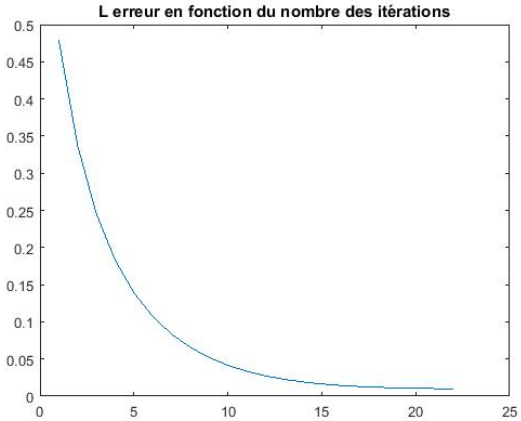


En ceux qui concerne la partie de décomposition de domaine, on obtient les solutions suivantes :





On obtient l'erreur suivante :



Chapitre 2

Freefem

2.1 Introduction

FreeFem++ est un logiciel Open Source permettant de résoudre numériquement des équations différentielles par éléments finis1. Il possède son propre langage de script, inspiré du C++, pour décrire le type de problème différentiel, les équations aux dérivées partielles et les conditions initiales et aux limites. Il peut ainsi résoudre des problèmes dits multi-physiques, présentant des non-linéarités, en bi- comme en tri-dimensionnel, sur des maillages pouvant aller au million de nœuds (ordinateur de calcul standard) jusqu'à quelques milliards de nœuds (gros système multi-processeurs dédié au calcul).
Développé au Laboratoire Jacques-Louis Lions de l'Université Pierre et Marie Curie [F. Hecht, O. Pironneau], porté sous Windows, Unix (Linux) et MacOs.

2.2 Caractéristiques de Freefem++

Freefem contient plusieurs caractéristiques comme :
– Plusieurs types d'éléments finis : élément linéaire et quadratique, P1 discontinu et élément de Raviart-Thomas, élément vectoriel,...
– Interpolation automatique des données à partir d'un maillage sur un autre.
– Définition du problème aux limites (complexe ou réel) directement avec la forme variationnelle.
– Formulation Galerkin Discontinu.
– Discription analytique de la frontière.
– Créer des maillages, basée sur l'algorithme de Delaunay-Voronoi.
– Lire et sauver des maillages, solutions
– Adaptation des maillages, isotrope ou

Chapitre 3

Discrétisation par éléments finis

La méthode consiste à approcher, dans un sous-espace de dimension finie, un problème écrit sous forme variationnelle dans un espace de dimension infinie. La solution approchée est dans ce cas une fonction déterminée par un nombre de paramètres comme, par exemple, ses valeurs en certains points ou noeuds du maillage.
Avantages : traitement possible de géométries complexes, nombreux résultats théoriques sur la convergence.
Inconvénient : complexité de mise en oeuvre et grand coût en temps de calcul et mémoire.

3.1 Discrétisation de l'équation de Poisson

Nous considérons le problème aux limites suivant :

$$\begin{cases} \triangle u = f dans \Omega \\ u = u ex dans \partial\Omega \end{cases}$$

Dans une première étape il faut proposer une formulation variationnelle du problème aux limites. Cette formulation variationnelle du problème va encoder a elle seule, à la fois l'équation et les conditions aux limites. De manière conceptuelle, il s'agit de trouver une forme bilinéaire a(·,·), une forme linéaire L(·), et un espace de Hilbert V tels que soit équivalent à :
Trouver u $\in V$ tel que a(u,v) = L(v) pour tout v$\in V$.
On multiplie l'équation par v, on intégre sur Ω ; puis on utilise :

l'équation de Green :
Soit Ω un ouvert borné régulier de classe C^2. Si u$\in$ $H^2(\Omega)$ et v$\in$ $H^1(\Omega)$, on a :

$$\int_\Omega \triangle u * v \mathrm{d}x = \int_\Omega \nabla u * \nabla v \mathrm{d}x - \int_{\partial_\Omega} \frac{\partial u}{\partial n} * v * \partial s$$

On obtient la formulation variationnelle :

$$\boxed{\int_\Omega f * v \mathrm{d}x = \int_\Omega \nabla u * \nabla v \mathrm{d}x - \int_{\partial\Omega} \frac{\partial u}{\partial n} * v * \partial s}$$

3.2 Application

En prenant les mêmes hypothèses que celle de la méthode des différences finies. On obtient sur un carré :

Fig. 3.1 – solution exacte

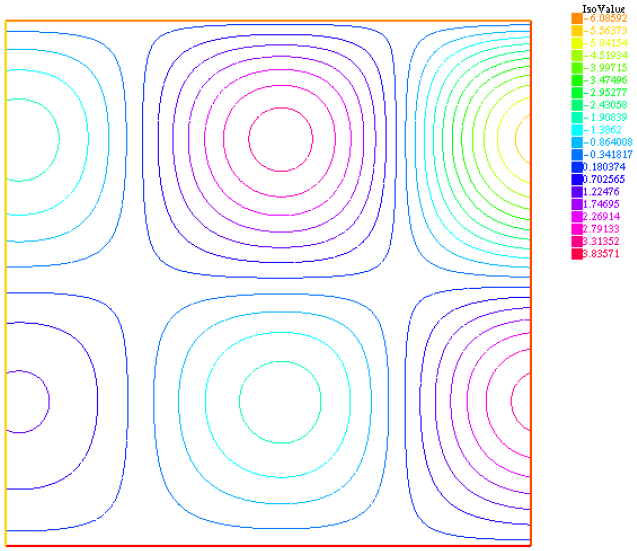
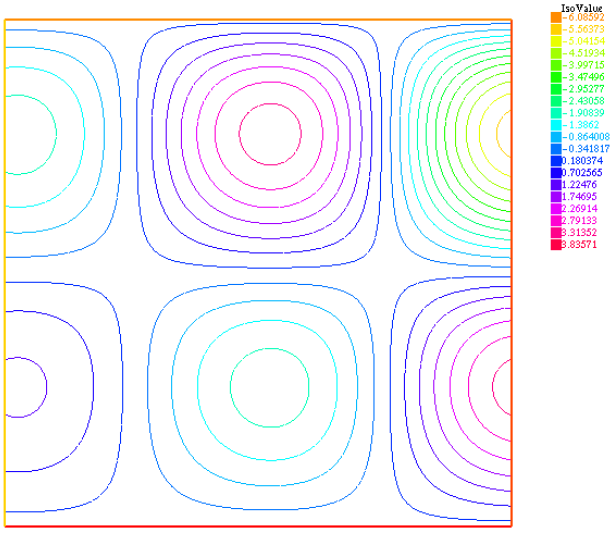
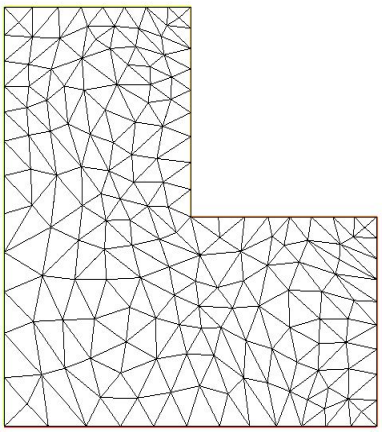
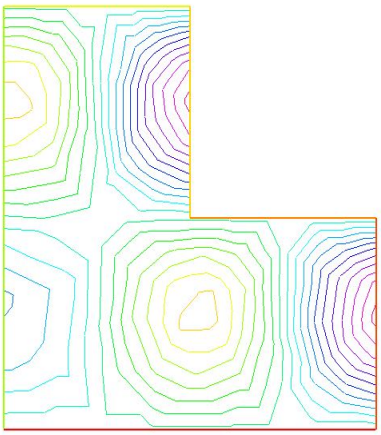


Fig. 3.2 – solution approchée

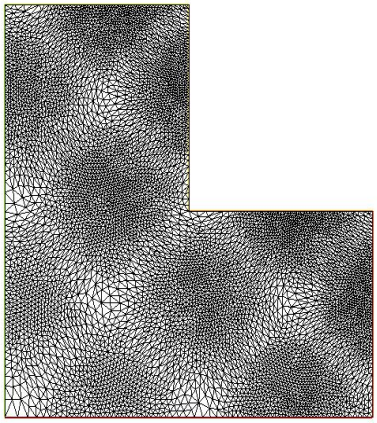


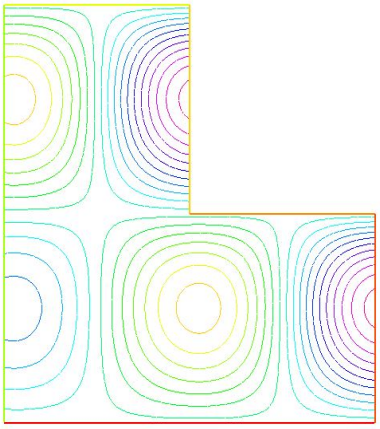
Sur un L, on obtient :
Avec un maillage grossier :





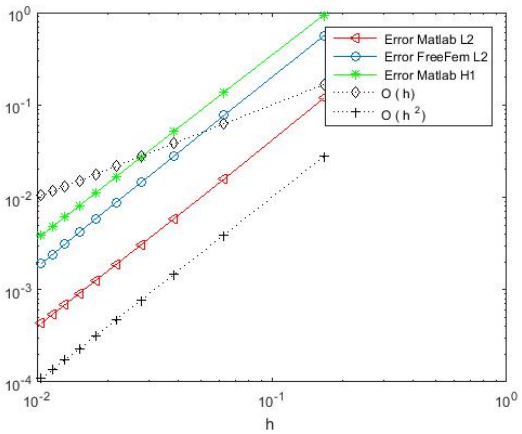
Avec un maillage fin :





Dans la figure suivante, on remarque que les courbes de l'erreur entre la méthode des différences finies et éléments finis sont près l'une de l'autre :

Fig. 3.3 – comparaison erreur



Annexe A

Code matlab pour le carré

```
clear
clc
```

```
%% initialisation
n=40;a=0;b=1;
k1=1;k2=1;a1=1;a2=1;
g = @(x,y) exp(k1*x + k2*y).*cos(a1*2*pi*x).*sin(a2*2*pi*y);
f = @(x,y) 2*exp(x + y).*cos(2*pi*x).*sin(2*pi*y) + 4*pi*exp(x + y).*cos(2*pi*x).*cos(2*pi*y)-
4*pi*exp(x + y).*sin(2*pi*x).*sin(2*pi*y) - 8*pi^2*exp(x + y).*cos(2*pi*x).*sin(2*pi*y);
[u,x,y] = fd2poissonfinale(f,g,a,b,n);
h = x(1,2) - x(1,1);
```

```
%% Plot solution
contour(u)
title(strcat('Solution touvee, h=',num2str(h)));
figure
contour(g(x,y))
title(strcat('Solution exacte, h=',num2str(h)));
figure
surf(u)
title(strcat('Solution trouvee 3D, h=',num2str(h)));
figure
surf(g(x,y))
title(strcat('Solution exacte 3D, h=',num2str(h)));
```

```
%% Plot erreur
z=u-g(x,y);
figure, set(gcf,'DefaultAxesFontSize',8,'PaperPosition', [0 0 3.5 3.5]),
mesh(x,y,z), colormap([0 0 0]), xlabel('x'), ylabel('y'), zlabel('Erreur'),
title(strcat('Erreur, h=',num2str(h)));
figure
plot(z)
```

```
% Approximation numérique de l'équation de Poisson sur le carré [a, b] x [a, b] avec
% Conditions aux limites de Dirichlet. Utilise un maillage uniforme avec un total de (n + 2)
%x (n + 2)
% de points (c.-à-d. n x n points de grille intérieurs).
%
% pfunc: la fonction f de l'équation du poisson (c'est-à-dire le laplacien de u).
% uex: la fonction frontière représentant les conditions de Dirichlet (c'est-à-dire la solution
% a, b: l'intervalle définissant le carré
% n: n + 2 est le nombre de points dans la direction du maillage.
% Ouput:
% u: la solution numérique de l'équation de Poisson aux points du maillage.
% x, y: le maillage uniforme.

function [u,x,y] = fd2poissonfinale(pfunc,uex,a,b,n)

h = abs(b-a)/(n+1); % le pas
[x,y] = meshgrid(a:h:b); % maillage uniforme, y compris les points limites.

%% Calcul de u sur les frontieres avec les conditions de Dirichlet
ub = zeros(n,n);
idx = 2:n+1;
idy = 2:n+1;
% Les limites West and East
ub(:,1) = feval(uex,x(idx,1),y(idy,1)); % West
ub(:,n) = feval(uex,x(idx,n+2),y(idy,n+2)); % East
% Les limites North and South
ub(1,1:n) = ub(1,1:n) + feval(uex,x(1,idx),y(1,idy)); %North
ub(n,1:n) = ub(n,1:n) + feval(uex,x(n+2,idx),y(n+2,idy)); %South
% Convertir ub en vecteur colonne
ub = (1/h^2)*reshape(ub,n*n,1);

%% La fonction f sur les points interieurs.
f = feval(pfunc,x(idx,idy),y(idx,idy));
% Convertit f en vecteur colonne
f = reshape(f,n*n,1);

%% Creation de la matrice du laplacien qui vaut D2x + D2y
z = [-2;1;zeros(n-2,1)];
D2x = 1/h^2*kron(toeplitz(z,z),eye(n));
D2y = 1/h^2*kron(eye(n),toeplitz(z,z));

%% Resoluion du systeme
u = (D2x + D2y)\(f-ub);

%% Convertir u de vecteur colonne en matrice
u = reshape(u,n,n);

%% La solution finale
north=feval(uex,x(1,1:n+2),y(1,1:n+2));
west=feval(uex,x(2:n+1,1),y(2:n+1,1));
east=feval(uex,x(2:n+1,n+2),y(2:n+1,n+2));
south=feval(uex,x(n+2,1:n+2),y(n+2,1:n+2));
% Concat des conditions aux limites
u = [[north];...
[[west] u ...
[east]];...
[south]];
```

Annexe B

Code matlab pour l'erreur carré

```
clear
clc
%% initialisation
a=0;b=1;
k1=1;k2=1;a1=1;a2=1;
err=0;errh1=0;H=0;
err_freefem=[0.56;0.077;0.028;0.0145;0.00883;0.0059;0.0042;0.0031;0.0024;0.0019];
g = @(x,y) exp(k1*x + k2*y).*cos(a1*2*pi*x).*sin(a2*2*pi*y);
f = @(x,y) 2*exp(x + y).*cos(2*pi*x).*sin(2*pi*y) + 4*pi*exp(x + y).*cos(2*pi*x).*cos(2*pi*y) -
4*pi*exp(x + y).*sin(2*pi*x).*sin(2*pi*y) - 8*pi^2*exp(x + y).*cos(2*pi*x).*sin(2*pi*y);

%% Calcul erreur
k=1;
for n = 5:10:95
    [u,x,y] = fd2poisson(f,g,a,b,n);
    [err , err_freefem, errh1, H,k] = erreurs_matlab_freefem(u,g,err,errh1,H,x,y,a,b,n,k);
end

loglog(err)
loglog (H,err, 'r < - ')
hold on
loglog (H,err_freefem, 'o-')
loglog (H,errh1, 'g - *')
loglog (H,H, ' kd : ')
loglog (H,H.^2 , 'k +: ')
legend ( ' Error Matlab L2 ' ,' Error FreeFem L2 ',' Error Matlab H1 ', 'O ( h) ' , 'O ( h ^2) ')
xlabel ( 'h ')
```

```
function [err , err_freefem, errh1, H,k] = erreurs_matlab_freefem(u,g,err,errh1,H,x,y,a,b,n,k)
    %% Les erreurs L2 Matlab et FreeFem
    err_freefem=[0.56;0.077;0.028;0.0145;0.00883;0.0059;0.0042;0.0031;0.0024;0.0019]; %erreur freefem
    H(k)=(b-a)/(n+1); % le vecteur des pas
    e=reshape(u-g(x,y),(n+2)*(n+2),1);
    err(k)=H(k)*norm(e,2); %l'erreur matlab

    %% Erreur H1
    uint=u(2:end-1,2:end-1);
    gint=g(x,y);gint=gint(2:end-1,2:end-1);
    eh1=reshape(uint-gint,n*n,1);
    % Creation de la matrice du laplacien qui vaut D2x + D2y
    h=H(k);
    z = [-2;1;sparse(n-2,1)];
    D2x = kron(toeplitz(z,z),eye(n));
    D2y = kron(eye(n),toeplitz(z,z));
    A= (D2x+D2y);
    %Calcul erreur H1
    errh1(k)=sqrt((err(k))^2 + abs(eh1'*A*eh1));

    %% Incrémentation
    k=k+1; % l iteration
end
```

Annexe C

Code matlab pour le L

```
clear
clc
```

```
%% initialisation
n=40;div=floor(n/2);a=0;b=1;
k1=1;k2=1;a1=1;a2=1;
g = @(x,y) exp(k1*x + k2*y).*cos(a1*2*pi*x).*sin(a2*2*pi*y);
f = @(x,y) 2*exp(x + y).*cos(2*pi*x).*sin(2*pi*y) + 4*pi*exp(x + y).*cos(2*pi*x).*cos(2*pi*y) -
4*pi*exp(x + y).*sin(2*pi*x).*sin(2*pi*y) - 8*pi^2*exp(x + y).*cos(2*pi*x).*sin(2*pi*y);
```

```
%% resolution
[u,x,y] = L_poisson(f,g,a,b,n);
h = x(1,2) - x(1,1)
```

```
%% la solution exacte
ex=g(x,y);
for i = div+1:n+2
    for j = div+1:n+2
        ex(i,j)=nan;
    end
end
```

```
%% Plot solution
figure, set(gcf,'DefaultAxesFontSize',8,'PaperPosition', [0 0 3.5 3.5]),
mesh(x,y,u), colormap([0 0 0]), xlabel('x'), ylabel('y'), zlabel('u(x,y)'),
title(strcat('Solution trouvee 3D , h=',num2str(h)));
figure
contour(u)
title(strcat('Solution trouvee, h=',num2str(h)));
figure, set(gcf,'DefaultAxesFontSize',8,'PaperPosition', [0 0 3.5 3.5]),
mesh(x,y,ex), colormap([0 0 0]), xlabel('x'), ylabel('y'), zlabel('u(x,y)'),
title(strcat('Solution exacte 3D, h=',num2str(h)));
figure
contour(ex)
title(strcat('Solution exacte, h=',num2str(h)));
```

```
%% Plot erreur
figure, set(gcf,'DefaultAxesFontSize',8,'PaperPosition', [0 0 3.5 3.5]),
mesh(x,y,u-ex), colormap([0 0 0]), xlabel('x'), ylabel('y'), zlabel('Erreur'),
title(strcat('Erreur, h=',num2str(h)));
```

```
function [u,x,y] = L_poisson(pfunc,bfunc,a,b,n)

div=floor(n/2);
h = abs(b-a)/(n+1);          % le pas
[x,y] = meshgrid(a:h:b); % maillage uniforme, y compris les points limites.

%% Calcul de u sur les frontieres avec les conditions de Dirichlet
ub = zeros(n,n);
idx = 2:n+1;
idy = 2:n+1;
% Les limites West and East
ub(:,1) = feval(bfunc,x(idx,1),y(idy,1)); % West
ub(:,n) = feval(bfunc,x(idx,n+2),y(idy,n+2)); % East
% Les limites North and South
ub(1,1:n) = ub(1,1:n) + feval(bfunc,x(1,idx),y(1,idy));
ub(n,1:n) = ub(n,1:n) + feval(bfunc,x(n+2,idx),y(n+2,idy));
% Convertir ub en vecteur colonne
ub = (1/h^2)*reshape(ub,n*n,1);

%% La fonction f sur les points interieurs.
f = feval(pfunc,x(idx,idy),y(idx,idy));
% Convertit f en vecteur colonne
f = reshape(f,n*n,1);

%% Creation de la matrice du laplacien qui vaut D2x + D2y
z = [-2;1;zeros(n-2,1)];
D2x = 1/h^2*kron(toeplitz(z,z),eye(n));
D2y = 1/h^2*kron(eye(n),toeplitz(z,z));

%% Resoluion du systeme
u = (D2x + D2y)\(f-ub);

%% Convertir u de vecteur colonne en matrice
u = reshape(u,n,n);
```

```
%% Creation de L
for i = div+1:n
    for j = div+1:n
        u(i,j)=nan;
    end
end

%% les conditions aux limites
east=feval(bfunc,x(2:n+1,n+2),y(2:n+1,n+2));
u(div+1:n,div)=feval(bfunc,x(div+1:n,div),y(div+1:n,div)); %conditions aux limites pour L board
east(div+1:n,1)=nan;
south=feval(bfunc,x(n+2,1:n+2),y(n+2,1:n+2));
u(div,div+1:n)=feval(bfunc,x(div,div+1:n),y(div,div+1:n)); %conditions aux limites pour L board
south(div+2:n+2)=nan;

%% La solution finale
u = [[feval(bfunc,x(1,1:n+2),y(1,1:n+2))];...
[[feval(bfunc,x(2:n+1,1),y(2:n+1,1))] u ...
[east]];...
[south]];
```

Annexe D

Code décomposition du domaine

Code principale :

```
clear
clc
%% inistialisation
k1=1;
k2=1;
a1=1;
a2=1;
n=300;a=0;b=1;
g = @(x,y) exp(k1*x + k2*y).*cos(a1*2*pi*x).*sin(a2*2*pi*y);
f = @(x,y) 2*exp(x + y).*cos(2*pi*x).*sin(2*pi*y) + 4*pi*exp(x + y).*cos(2*pi*x).*cos(2*pi*y) -
4*pi*exp(x + y).*sin(2*pi*x).*sin(2*pi*y) - 8*pi^2*exp(x + y).*cos(2*pi*x).*sin(2*pi*y);
[ul,x,y] = l_fd2poisson(f,g,a,b,n);
[ur,x,y] = r_fd2poisson(f,g,a,b,n);
%% Decomposition du domaine
mid=rand(n,1); % vecteur qui divise le domaine en deux parties
erreur=10;plot_erreur=0;ab_erreur=0;k=1  % initialisations
while (erreur >= 0.01) && (k < 1000)

    [ud,x,y,h] = droite_poisson(f,g,mid,a,b,n);     % Calcul de u sur la partie droite
    mid=ud(:,2);                                    % Récupération du bord gauche de ud (les points intérieurs)
    [ug,x,y,h] = gauche_poisson(f,g,mid,a,b,n);     % Calcul de u sur la partie gauche
    mid=ug(:,end-1);                                  % Récupération du bord droite de ug (les points intérieurs)

%% Assemblage des solutions
    [u, u_mono]= assemblage(g,ug,ud,ul,ur,a,b,h,n);

%% Affichage de l'erreur apres chaque itération
    [erreur, plot_erreur, ab_erreur,k] = affichage_plot_erreur(u,u_mono,plot_erreur,ab_erreur,k);

end
```

```
%% plot de la solution
figure
contour(u)
title(['la solution avec decomposition du domaine 2Davec erreur=',num2str(erreur),' et h = ',num2str(h)]);
figure
contour(u_mono)
title(['la solution monodomaine avec h=',num2str(h)]);
figure
surf(u)
title(strcat('Solution avec decomposition du domaine 3D, h=',num2str(h)));
figure
surf(g(x,y))
title(strcat('Solution avec decomposition du domaine 3D, h=',num2str(h)));
```

```
%% plot de l'erreur
figure
plot(ab_erreur,plot_erreur);
title('L erreur en fonction du nombre des itérations');
```

Fonction droite-poisson :

```
% Approximation numérique de l'équation de Poisson sur le carré [a, b] x [a, b] avec
% Conditions aux limites de Dirichlet. Utilise un maillage uniforme avec un total de (n + 2) x (n + 2)
% de points (c.-à-d. n x n points de grille intérieurs).
%
% pfunc: la fonction f de l'équation du poisson (c'est-à-dire le laplacien de u).
% bfunc: la fonction frontière représentant les conditions de Dirichlet (c'est-à-dire la solution exacte).
% a, b: l'intervalle définissant le carré
% n: n + 2 est le nombre de points dans la direction du maillage.
% Ouput:
% u: la solution numérique de l'équation de Poisson aux points du maillage.
% x, y: le maillage uniforme.
%
function [u,x,y,h] = droite_poisson(pfunc,bfunc,mid,a,b,n)
```

```
h = abs(b-a)/(n+1);      % le pas
X=linspace(a-h,b,n+2);
Y=linspace(a,b,n+2);
[x,y] = meshgrid(X,Y); % maillage uniforme, y compris les points limites.
```

```
%% Calcul de u sur les frontieres avec les conditions de Dirichlet
ub = sparse(n,n);
idx = 2:n+1;
idy = 2:n+1;
% Les limites West and East
ub(:,1) = mid;                              % West qui prend mid
ub(:,n) = feval(bfunc,x(idx,n+2),y(idy,n+2)); % East
% Les limites North and South
ub(1,1:n) = ub(1,1:n) + feval(bfunc,x(1,idx),y(1,idy));      %South
ub(n,1:n) = ub(n,1:n) + feval(bfunc,x(n+2,idx),y(n+2,idy));  %North
% Convertir ub en vecteur colonne
ub = (1/h^2)*reshape(ub,n*n,1);
```

```
%% La fonction f sur les points interieurs.
f = feval(pfunc,x(idx,idy),y(idx,idy));
% Convertit f en vecteur colonne
f = reshape(f,n*n,1);
```

```
%% Creation de la matrice du laplacien qui vaut D2x + D2y
z = [-2;1;sparse(n-2,1)];
D2x = 1/h^2*kron(toeplitz(z,z),eye(n));
D2y = 1/h^2*kron(eye(n),toeplitz(z,z));
```

```
%% Resoluion du systeme
u = (D2x + D2y)\(f-ub);
```

```
%% Convertir u de vecteur colonne en matrice
u = reshape(u,n,n);
```

Fonction gauche-poisson :

```
% Approximation numérique de l'équation de Poisson sur le carré [a, b] x [a, b] avec
% Conditions aux limites de Dirichlet. Utilise un maillage uniforme avec un total de (n + 2) x (n + 2)
% de points (c.-à-d. n x n points de grille intérieurs).
%
% pfunc: la fonction f de l'équation du poisson (c'est-à-dire le laplacien de u).
% bfunc: la fonction frontière représentant les conditions de Dirichlet (c'est-à-dire la solution exacte).
% a, b: l'intervalle définissant le carré
% n: n + 2 est le nombre de points dans la direction du maillage.
% Ouput:
% u: la solution numérique de l'équation de Poisson aux points du maillage.
% x, y: le maillage uniforme.
%
function [u,x,y,h] = gauche_poisson(pfunc,bfunc,mid,a,b,n)
```

```
h = abs(b-a)/(n+1);      % le pas
X=linspace(a,b+h,n+2);
Y=linspace(a,b,n+2);
[x,y] = meshgrid(X,Y);
x=x-1;                   % car la partie gauche est [-1,0+h]x[0,1]
```

```
%% Calcul de u sur les frontieres avec les conditions de Dirichlet
ub = sparse(n,n);
idx = 2:n+1;
idy = 2:n+1;
% Les limites West and East
ub(:,1) = feval(bfunc,x(idx,1),y(idy,1));     % West
ub(:,n) = mid;                                % East qui prend mid
% Les limites North and South
ub(1,1:n) = ub(1,1:n) + feval(bfunc,x(1,idx),y(1,idy));      %South
ub(n,1:n) = ub(n,1:n) + feval(bfunc,x(n+2,idx),y(n+2,idy));  %North
% Convertir ub en vecteur colonne
ub = (1/h^2)*reshape(ub,n*n,1);
```

```
%% La fonction f sur les points interieurs.
f = feval(pfunc,x(idx,idy),y(idx,idy));
% Convertit f en vecteur colonne
f = reshape(f,n*n,1);
```

```
%% Creation de la matrice du laplacien qui vaut D2x + D2y
z = [-2;1;sparse(n-2,1)];
D2x = 1/h^2*kron(toeplitz(z,z),eye(n));
D2y = 1/h^2*kron(eye(n),toeplitz(z,z));
```

```
%% Resoluion du systeme
u = (D2x + D2y)\(f-ub);
```

```
%% Convertir u de vecteur colonne en matrice
u = reshape(u,n,n);
```

Fonction l-fd2poisson :

```
% Approximation numérique de l'équation de Poisson sur le carré [a, b] x [a, b] avec
% Conditions aux limites de Dirichlet. Utilise un maillage uniforme avec un total de (n + 2) x (n + 2)
% de points (c.-à-d. n x n points de grille intérieurs).
%
% pfunc: la fonction f de l'équation du poisson (c'est-à-dire le laplacien de u).
% uex: la fonction frontière représentant les conditions de Dirichlet (c'est-à-dire la solution exacte).
% a, b: l'intervalle définissant le carré
% n: n + 2 est le nombre de points dans la direction du maillage.
% Ouput:
% u: la solution numérique de l'équation de Poisson aux points du maillage.
% x, y: le maillage uniforme.
%
function [u,x,y] = l_fd2poisson(pfunc,uex,a,b,n)
```

```
h = abs(b-a)/(n+1);      % le pas
X=linspace(a,b+h,n+2);
Y=linspace(a,b,n+2);
[x,y] = meshgrid(X,Y);
x=x-1;                   % car la partie gauche est [-1,0+h]x[0,1]
```

```
%% Calcul de u sur les frontieres avec les conditions de Dirichlet
ub = sparse(n,n);
idx = 2:n+1;
idy = 2:n+1;
% Les limites West and East
ub(:,1) = feval(uex,x(idx,1),y(idy,1)); % West
ub(:,n) = feval(uex,x(idx,n+2),y(idy,n+2)); % East
% Les limites North and South
ub(1,1:n) = ub(1,1:n) + feval(uex,x(1,idx),y(1,idy)); %North
ub(n,1:n) = ub(n,1:n) + feval(uex,x(n+2,idx),y(n+2,idy)); %South
% Convertir ub en vecteur colonne
ub = (1/h^2)*reshape(ub,n*n,1);
```

```
%% La fonction f sur les points interieurs.
f = feval(pfunc,x(idx,idy),y(idx,idy));
% Convertit f en vecteur colonne
f = reshape(f,n*n,1);
```

Fonction r-fd2poisson :

```
% Approximation numérique de l'équation de Poisson sur le carré [a, b] x [a, b] avec
% Conditions aux limites de Dirichlet. Utilise un maillage uniforme avec un total de (n + 2) x (n + 2)
% de points (c.-à-d. n x n points de grille intérieurs).
%
% pfunc: la fonction f de l'équation du poisson (c'est-à-dire le laplacien de u).
% uex: la fonction frontière représentant les conditions de Dirichlet (c'est-à-dire la solution exacte).
% a, b: l'intervalle définissant le carré
% n: n + 2 est le nombre de points dans la direction du maillage.
% Ouput:
% u: la solution numérique de l'équation de Poisson aux points du maillage.
% x, y: le maillage uniforme.
%
function [u,x,y] = r_fd2poisson(pfunc,uex,a,b,n)

h = abs(b-a)/(n+1);      % le pas
X=linspace(a-h,b,n+2);
Y=linspace(a,b,n+2);
[x,y] = meshgrid(X,Y); % maillage uniforme, y compris les points limites.
%% Calcul de u sur les frontieres avec les conditions de Dirichlet
ub = sparse(n,n);
idx = 2:n+1;
idy = 2:n+1;
% Les limites West and East
ub(:,1) = feval(uex,x(idx,1),y(idy,1)); % West
ub(:,n) = feval(uex,x(idx,n+2),y(idy,n+2)); % East
% Les limites North and South
ub(1,1:n) = ub(1,1:n) + feval(uex,x(1,idx),y(1,idy)); %North
ub(n,1:n) = ub(n,1:n) + feval(uex,x(n+2,idx),y(n+2,idy)); %South
% Convertir ub en vecteur colonne
ub = (1/h^2)*reshape(ub,n*n,1);

%% La fonction f sur les points interieurs.
f = feval(pfunc,x(idx,idy),y(idx,idy));
% Convertit f en vecteur colonne
f = reshape(f,n*n,1);

%% Creation de la matrice du laplacien qui vaut D2x + D2y
z = [-2;1;sparse(n-2,1)];
D2x = 1/h^2*kron(toeplitz(z,z),eye(n));
D2y = 1/h^2*kron(eye(n),toeplitz(z,z));

%% Resoluion du systeme
u = (D2x + D2y)\(f-ub);

%% Convertir u de vecteur colonne en matrice
u = reshape(u,n,n);

%% La solution finale
north=feval(uex,x(1,1:n+2),y(1,1:n+2));
west=feval(uex,x(2:n+1,1),y(2:n+1,1));
east=feval(uex,x(2:n+1,n+2),y(2:n+1,n+2));
south=feval(uex,x(n+2,1:n+2),y(n+2,1:n+2));
% Concat des conditions aux limites
u = [[north];...
[[west] u ...
[east]];...
[south]];
```

Fonction assemblage :

```
function [u, u_mono]= assemblage(g,ug,ud,ul,ur,a,b,h,n)
%% La solution sur la partie gauche avec les conditions aux limites
   %% Concat des conditions aux limites
    h = abs(b-a)/(n+1);      % le pas
    X=linspace(a,b+h,n+2);
    Y=linspace(a,b,n+2);
    [x,y] = meshgrid(X,Y);
    x=x-1;                   % car la partie gauche est [-1,0+h]x[0,1]

    north=feval(g,x(1,1:n+2),y(1,1:n+2));
    west=feval(g,x(2:n+1,1),y(2:n+1,1));
    east=feval(g,x(2:n+1,n+2),y(2:n+1,n+2));
    south=feval(g,x(n+2,1:n+2),y(n+2,1:n+2));

    %% Assemblage de la solution sur la partie gauche
    ug = [[north];...
    [[west] ug ...
    [east]];...
    [south]];


%% La solution de la partie droite avec les conditions aux limites
    %% Concat des conditions aux limites
    h = abs(b-a)/(n+1);      % le pas
    X=linspace(a-h,b,n+2);
    Y=linspace(a,b,n+2);
    [x,y] = meshgrid(X,Y); % maillage uniforme, y compris les points limites.

    north=feval(g,x(1,1:n+2),y(1,1:n+2));
    west=feval(g,x(2:n+1,1),y(2:n+1,1));
    east=feval(g,x(2:n+1,n+2),y(2:n+1,n+2));
    south=feval(g,x(n+2,1:n+2),y(n+2,1:n+2));

   %% Assemblage de la solution sur la partie droite
   ud = [[north];...
   [[west] ud ...
   [east]];...
   [south]];

%% La solution finale et la solution exacte sur tout le domaine
    u=[ug(:,1:end-2) ud(:,3:end)];
    u_mono=[ul(:,1:end-2) ur(:,3:end)];
```

Annexe E

Code Freefem pour le carre

Code de la solution approchée :

```
//Declaration des variables
int N=100;
real k1=1.0;
real k2=1.0;
real a1=1;
real a2=1;

//La definition du maillage
mesh Th=square(N,N);
plot(Th);
//La fonction de f qui est la derivee seconde de la solution exacte
func f = 2*exp(x + y)*cos(2*pi*x)*sin(2*pi*y) + 4*pi*exp(x + y)*cos(2*pi*x)*cos(2*pi*y) -
4*pi*exp(x + y)*sin(2*pi*x)*sin(2*pi*y) - 8*pi^2*exp(x + y)*cos(2*pi*x)*sin(2*pi*y);

func uex = exp(k1*x + k2*y)*cos(a1*2*pi*x)*sin(a2*2*pi*y);

//La definition de l'espace P1
//au maillage Th
fespace Vh(Th,P1);

//Les fonctions uh et vh qui sont des elements de Vh
Vh uh,vh;

//La formulation variationnelle
problem poisson(uh,vh,solver=LU)=
int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))
+int2d(Th)(f*vh)
+on(1,2,3,4,uh=uex);//les bords avec square

//L'affichage du resultat
plot(uh,value=true);

// Comparaison////solution exacte
Vh err=uh-uex;
real errnorm= sqrt(int2d(Th)( square(uh-uex) ));
cout <<  " err = " << errnorm << endl;
```

Code de la solution exacte :

```
//Declaration des variables
real k1=1.0;
real k2=1.0;
real a1=1;
real a2=1;
real x0 = 0;
real x1 = 1;
real y0 = 0;
real y1 = 1;
int n = 20;
real m = 20;
//La definition du maillage sue carré [x0,x1] x [y0,y1]
mesh Th = square(n, m, [x0+(x1-x0)*x, y0+(y1-y0)*y]);

//La definition de l'espace P1
//au maillage Th
fespace Vh(Th,P1);

//Les fonctions uh et vh qui sont des elements de Vh
Vh uex = exp(k1*x + k2*y)*cos(a1*2*pi*x)*sin(a2*2*pi*y);

//L'affichage du resultat
plot(uex,value=true);
```

Annexe F

Code Freefem pour le L

```
real[int] xx(10), yy(10);
int k=0;

        //Declaration des variables
        real error=0.01;
        real k1=1.0;
        real k2=1.0;
        real a1=1;
        real a2=1;

        //La definition du maillage
        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 ;} ;
        mesh Th = buildmesh (a(6) + b(4) + c(4) +d(4) + e(4) + f(6)) ;

        //La fonction de f qui est la derivee seconde de la solution exacte
        func ft = 2*exp(x + y)*cos(2*pi*x)*sin(2*pi*y) + 4*pi*exp(x + y)*cos(2*pi*x)*cos(2*pi*y) -
4*pi*exp(x + y)*sin(2*pi*x)*sin(2*pi*y) - 8*pi^2*exp(x + y)*cos(2*pi*x)*sin(2*pi*y);

        func uex = exp(k1*x + k2*y)*cos(a1*2*pi*x)*sin(a2*2*pi*y);

        //La definition de l'espace P1
        //au maillage Th
        fespace Vh(Th,P1);

        //Les fonctions uh et vh qui sont des elements de Vh
        Vh uh,vh;

        //La formulation variationnelle
        problem poisson(uh,vh,solver=LU)=
        int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))
        +int2d(Th)(ft*vh)
        + on(1,2,3,4,5,6,uh=uex);//les bords
// Adaptmesh loop
for (int i = 0; i < 4; i++){
    poisson;
    Th = adaptmesh(Th, uh, err=error);
    error = error/2;
}

// Plot
plot(uh);
Vh err=uh-uex;
real errnorm= sqrt(int2d(Th)( square(uh-uex) ));
cout <<  " err = " << errnorm << endl;
```

Bibliographie

[1] https ://cel.archives-ouvertes.fr/file/index/docid/556967/filename/MethodesNumeriques$_E ricGoncalv$

[2] http ://www.cmap.polytechnique.fr/IMG/pdf/Setif-FreeFem$_I .pdf$

[3] http ://www.f-legrand.fr/scidoc/docimg/numerique/elliptique/poisson/poisson.html

[4] http ://pierre.guerin.work.free.fr/Projets$_D EA/Introduction.pdf$

[5] http ://www.cmap.polytechnique.fr/ alouges/polycopie2.pdf

.