



Rapport MA302

Analyse Numérique Matricielle

Trinômes : Arnaud BRIGNOL
Chok Leong CHAI
Quy-Nam NGUYEN
Date : 18/05/2009
Professeur : Sebastien BOYAVAL

Projet MA302

BRIGNOL Arnaud
CHAI Chok Leong
NGUYEN Quy-Nam

04/05/2009

Table des matières

I	Enoncé	3
II	Résolution du problème	5
1	Explication du problème ($P2$)	5
1.1	Méthode des différences finies	5
1.2	Approximation de l'équation de Poisson	5
1.2.1	Première méthode	5
1.2.2	Formule de Taylor	7
1.3	Construction de la matrice A_N	8
1.3.1	Cas pour $n = 2$	9
1.3.2	Cas pour $n = 3$	10
1.4	Généralisation	10
1.4.1	La matrice A_N	10
1.4.2	Le vecteur $b \in \mathbb{R}^N$	11
1.4.3	Forme générale du problème ($P2$)	11
2	Propriétés de la matrice A_N et sa décomposition LU	12
2.1	Propriétés de la matrice A_N	12
2.1.1	Symétrique	12
2.1.2	Définie	12
2.1.3	Positive	12
2.1.4	SDP	12
2.2	Décomposition LU	13
2.2.1	Décomposition à la main avec $n = 2$	13
2.2.2	Décomposition LU à l'aide d'un ordinateur	14
3	Inversion de la matrice A_N avec une méthode itérative	22
3.1	Méthode itérative d'inversion de Gauss-Seidel	22
3.1.1	Programmation de la méthode	22
3.2	Méthode directe d'inversion	26
3.2.1	Pivot de Gauss	26
3.2.2	Méthode d'inversion de LU	27
3.3	Comparaison entre les méthodes itératives et les méthodes directes pour inverser A_N	28
3.3.1	Programme de test	28
3.4	Amélioration	30
3.4.1	Méthode itérative de Newton	30
3.5	Modification sur les programmes	34
3.5.1	Programme de Gauss Seidel modifié pour inverser une matrice	34

3.5.2	Programme de Jacobi modifié pour inverser une matrice	35
3.5.3	Programme de LU modifié pour inverser une matrice	36
3.5.4	Nouveaux résultats de comparaison	37
4	Forme quadratique du problème ($P2$)	38
4.1	Dérivation de la fonction quadratique $J(X)$	38
4.2	La solution de qui minimise la forme quadratique	38
5	Méthodes numériques pour la résolution du problème ($P3$)	39
5.1	Construction du vecteur p	39
5.1.1	Vérification du vecteur p	39
5.2	Propositions des méthodes numériques	41
5.2.1	La méthode de Jacobi	41
5.2.2	La méthode de Gauss-Seidel	42
5.2.3	La méthode de relaxation	43
5.2.4	La méthode du gradient à pas optimal	44
5.2.5	La méthode du gradient conjugué	45
5.2.6	La méthode de SOR	48
5.3	Étude de l'efficacité entre les méthodes	49
5.3.1	Programme pour évaluer l'efficacité des méthodes	49
5.4	Résolution du problème ($P3$) avec la méthode de gradient conjugué	54
5.4.1	Programme de résolution	54
6	Conclusion	56

Première partie

Énoncé

On considère le problème de diffusion suivant, sur un domaine carré Ω du plan ($\Omega \subset \mathbb{R}^2$), paramétré par $(x, y) \in [0, 1] \times [0, 1]$:

Trouver $u : (x, y) \in \Omega \mapsto u(x, y) \in \mathbb{R}$ tel que

$$(P1) \begin{cases} -\frac{\partial}{\partial x^2} u(x, y) - \frac{\partial}{\partial y^2} u(x, y) & = f(x, y), \forall (x, y) \in \Omega \\ u(x = 0, y) & = 0 \\ u(x, y = 0) & = 0 \\ u(x = 1, y) & = y^2 \\ u(x, y = 1) & = x^2 \end{cases} \quad (1)$$

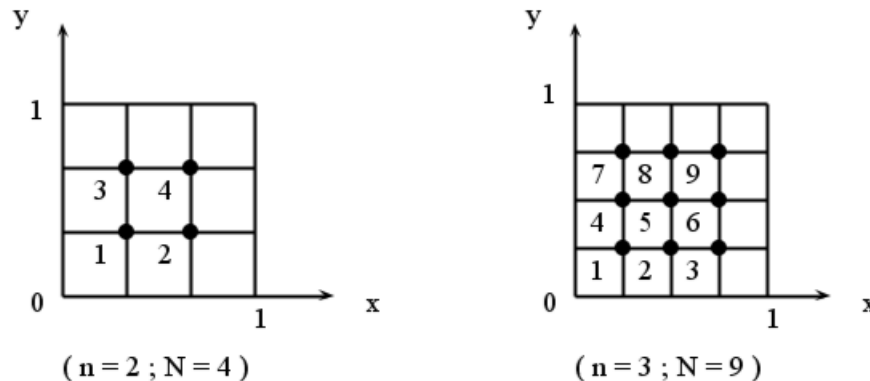
où le terme source f est donné par

$$f(x, y) = -2 \sin(4\pi x) - 2x^2 - 2y^2 + (4\pi)^2 y(y - 1) \sin(4\pi x)$$

Remarque : Ce genre de problèmes aux limites est fréquent, on l'appelle *problème de Dirichlet*. Typiquement, u est le champ de température dans un matériau homogène, avec une source de chaleur f et une température imposée au bord (contact avec un thermostat).

On va maintenant utiliser la méthode des différences finies introduites en cours pour approcher numériquement la solution de (P1). On commence par discrétiser Ω avec $N = n^2$ points disposés et numérotés régulièrement comme suit :

Exemple :



Remarque : Le point 1 a pour coordonnées $(\frac{1}{n+1}, \frac{1}{n+1})$, etc ...

On cherche maintenant une équation (soluble numériquement) satisfaite par des valeurs approchées de u aux points 1, 2, 3, etc ... On rassemble ces valeurs dans un vecteur de taille $N = n^2$

$$U = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} \in \mathbb{R}^N \quad (2)$$

On introduit les matrices de taille $N \times N$ suivantes,

$$A_N = \begin{pmatrix} A & B & 0 & \cdots & 0 \\ B^T & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & A & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & B \\ 0 & \cdots & 0 & B^T & A \end{pmatrix} \in \mathbb{R}^{N \times N} \quad (3)$$

avec A et B des matrices de taille $n \times n$ données par

$$A = \begin{pmatrix} 4 & -1 & 0 & \cdots & 0 \\ -1 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & 4 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 4 \end{pmatrix} \quad B = \begin{pmatrix} -1 & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & -1 & \ddots & \vdots \\ 0 & & \ddots & \ddots & 0 \\ -1 & 0 & \cdots & 0 & -1 \end{pmatrix} \quad (4)$$

Questions :

1. Expliquer pourquoi (P1) peut s'approcher par le problème (P2) suivant :

(P2) Trouver $U \in \mathbb{R}^N$ tel que $\frac{1}{(n+1)^2} A_N U = b$ où l'on donnera la forme du vecteur $b \in \mathbb{R}^N$.

Indication : Commencer par les cas où n est petit ($n = 2, 3, \text{ etc } \dots$).

2. Quelles sont les propriétés de la matrice A_N ? Donner sa décomposition LU , si elle existe.

Indication : Commencer avec $n = 2$ à la main, puis s'aider d'un ordinateur pour n grand. On joindra le code et les résultats.

3. Peut-on inverser la matrice avec une méthode itérative?

Si oui, la décrire et comparer numériquement la convergence et le temps de calcul pour différentes valeurs de N . Que dire par rapport à une méthode directe?

4. En notant $X \in \mathbb{R}^N$ un vecteur de taille N , expliquer pourquoi (P2) est équivalent au problème de moindres carrés (P3) suivant :

(P3) Trouver $X \in \mathbb{R}^N$ minimum de $J(X) = \frac{1}{2} X^T A_N X - (n+1)^2 X^T b$.

5. Proposer des méthodes numériques pour trouver la solution de (P2) en utilisant l'équivalence avec (P3).

Deuxième partie

Résolution du problème

1 Explication du problème (P2)

1.1 Méthode des différences finies

Le problème (P1) ressemble à l'équation de Poisson à 2 dimensions qui s'écrit, $\forall(x, y) \in \Omega$ et avec $\Omega \subset \mathbb{R}^2$, comme :

$$-\Delta u(x, y) = f(x, y) \quad \text{ou} \quad -\nabla^2 u(x, y) = f(x, y) \quad (5)$$

avec Δ l'opérateur laplacien et ∇ l'opérateur nabla vus lors des cours de physiques.

L'équation (5) s'écrit explicitement sous la forme :

$$-\frac{\partial^2}{\partial x^2} u(x, y) - \frac{\partial^2}{\partial y^2} u(x, y) = f(x, y), \forall(x, y) \in \Omega \quad (6)$$

Remarque : De l'équation (6), on voit facilement que le problème (P1) peut être caractérisé par l'équation de Poisson évoqué ci-dessus si on avait une dérivée partielle du second ordre de x ($\frac{\partial^2}{\partial x^2}$) au lieu d'une dérivée partielle du premier ordre de x^2 ($\frac{\partial}{\partial x^2}$) comme indiqué dans le problème (P1), et idem pour le y . D'après confirmation de M. BOYAVAL, on a bien $\frac{\partial^2}{\partial x^2}$ et $\frac{\partial^2}{\partial y^2}$. Donc, l'équation de Poisson correspond au problème (P1).

Il reste à résoudre cette équation aux dérivées partielles. Pour ce faire, on a à notre disposition la méthode des différences finies qui nous permet de trouver une solution approchée.

1.2 Approximation de l'équation de Poisson

1.2.1 Première méthode

Notre but est de trouver la température $u(x, y)$ dans un domaine carré Ω du plan ($\Omega \subset \mathbb{R}^2$) qui est soumise aux conditions de limites et une source de chaleur $f(x, y)$. D'après l'énoncé, le plan Ω est discrétisé en des points disposés régulièrement ligne par ligne et colonne par colonne illustré par la figure ci-dessous.

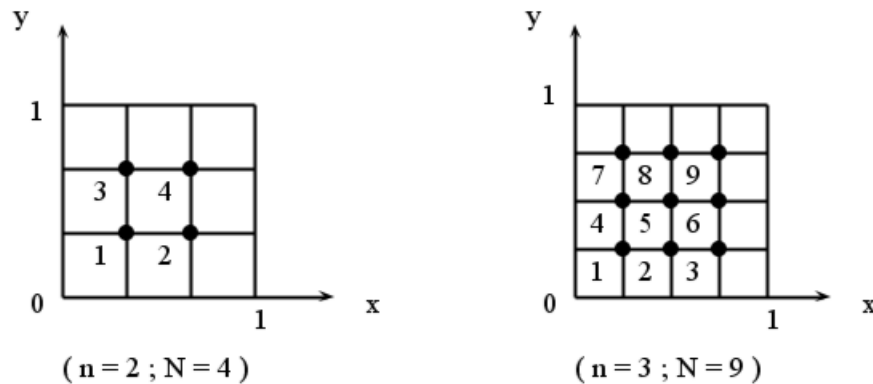


FIGURE 1 – Discrétisation

En généralisant, on obtient la configuration suivante :

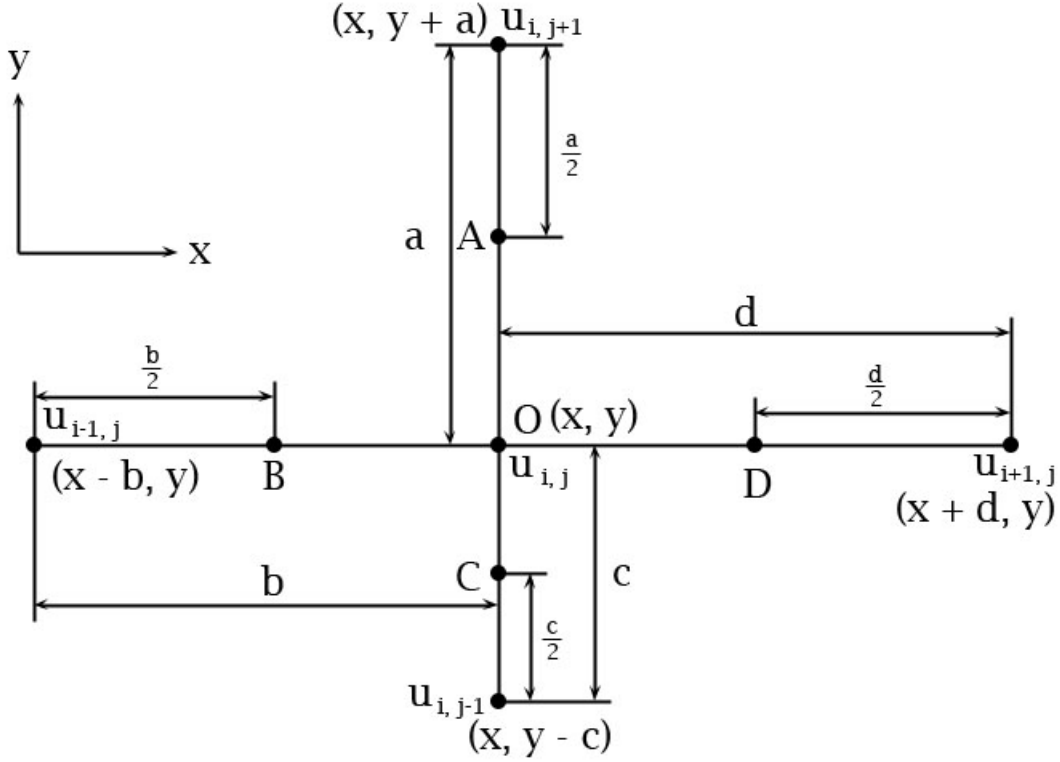


FIGURE 2 – Configuration quelconque

Avec $u_{i,j}$ un point quelconque dans le plan et avec $u_{i-1,j}$, $u_{i,j-1}$, $u_{i+1,j}$, $u_{i,j+1}$ ses plus proches voisins. On peut en déduire ainsi les équations suivantes :

$$\begin{cases} u_{i,j} &= u(x, y) \\ u_{i,j+1} &= u(x, y + a) \\ u_{i-1,j} &= u(x - b, y) \\ u_{i,j-1} &= u(x, y - c) \\ u_{i+1,j} &= u(x + d, y) \end{cases} \quad (7)$$

La dérivée partielle du premier ordre de $u(x, y)$ par rapport à x en points B et D sont, approximativement :

$$\frac{\partial u}{\partial x} \Big|_B = \frac{u_{i,j} - u_{i-1,j}}{b} \quad (8)$$

$$\frac{\partial u}{\partial x} \Big|_D = \frac{u_{i+1,j} - u_{i,j}}{d} \quad (9)$$

Idem, la dérivée partielle du premier ordre de $u(x, y)$ par rapport à y en points A et C sont, approximativement :

$$\frac{\partial u}{\partial y} \Big|_A = \frac{u_{i,j+1} - u_{i,j}}{a} \quad (10)$$

$$\frac{\partial u}{\partial y} \Big|_C = \frac{u_{i,j} - u_{i,j-1}}{c} \quad (11)$$

Et enfin, la dérivée partielle du second ordre peut être approximée au point O comme :

$$\frac{\partial^2 u}{\partial x^2} \Big|_O = \frac{\frac{\partial u}{\partial x} \Big|_D - \frac{\partial u}{\partial x} \Big|_B}{\partial x} = \frac{\frac{u_{i+1,j} - u_{i,j}}{d} - \frac{u_{i,j} - u_{i-1,j}}{b}}{\frac{d}{2} + \frac{b}{2}} \quad (12)$$

$$\frac{\partial^2 u}{\partial y^2}|_O = \frac{\frac{\partial u}{\partial y}|_A - \frac{\partial u}{\partial y}|_C}{\partial y} = \frac{\frac{u_{i,j+1} - u_{i,j}}{a} - \frac{u_{i,j} - u_{i,j-1}}{c}}{\frac{a}{2} + \frac{c}{2}} \quad (13)$$

Après avoir simplifié les équations (12) et (13), on obtient :

$$\frac{\partial^2 u}{\partial x^2}|_O = 2 \frac{(u_{i+1,j} - u_{i,j})b - (u_{i,j} - u_{i-1,j})d}{bd(d+b)} \quad (14)$$

$$\frac{\partial^2 u}{\partial y^2}|_O = 2 \frac{(u_{i,j+1} - u_{i,j})c - (u_{i,j} - u_{i,j-1})a}{ac(a+c)} \quad (15)$$

En appliquant ces approximations, l'équation (6) devient :

$$- \left(\frac{2u_{i,j+1}}{a(a+c)} + \frac{2u_{i-1,j}}{b(d+b)} + \frac{2u_{i,j-1}}{c(a+c)} + \frac{2u_{i+1,j}}{d(d+b)} - 2u_{i,j} \left[\frac{1}{db} + \frac{1}{ac} \right] \right) = f(x, y) \quad (16)$$

Dans notre cas, on a :

$$a = b = c = d = \frac{1}{n+1} \quad (17)$$

qui est un pas constant, d'où :

$$- \left(\frac{u_{i,j+1}}{a^2} + \frac{u_{i-1,j}}{a^2} + \frac{u_{i,j-1}}{a^2} + \frac{u_{i+1,j}}{a^2} - \frac{4u_{i,j}}{a^2} \right) = f(x, y) \quad (18)$$

$$u_{i,j} = \frac{1}{4} (u_{i,j+1} + u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + a^2 f(x, y)) \quad (19)$$

Cette dernière équation (19) va nous permettre de construire la matrice A_N .

1.2.2 Formule de Taylor

On peut aussi se servir de la formule de Taylor pour retrouver la même équation (19).

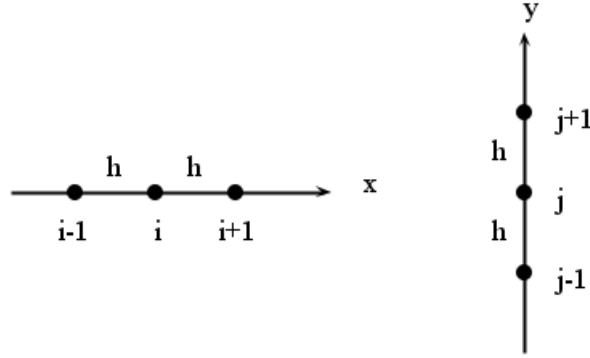


FIGURE 3 – Différences finies le long de l'axe x et de l'axe y

Considérons 3 points séparés espacés d'une distance h (un pas constant) sur l'axe x , notés $i-1$, i et $i+1$. Les valeurs de $u(x, y)$ en ces points sont u_{i-1} , u_i et u_{i+1} . On peut maintenant appliquer la formule de Taylor pour les valeurs de u_{i-1} et de u_{i+1} calculées au point i . On obtient :

$$u_{i-1} = u_i - \frac{\partial u}{\partial x}|_i \times h + \frac{\partial^2 u}{\partial x^2}|_i \times \frac{h^2}{2!} - \frac{\partial^3 u}{\partial x^3}|_i \times \frac{h^3}{3!} + \frac{\partial^4}{\partial x^4}|_i \times \frac{h^4}{4!} + O(h^5) \quad (20)$$

$$u_{i+1} = u_i + \frac{\partial u}{\partial x}|_i \times h + \frac{\partial^2 u}{\partial x^2}|_i \times \frac{h^2}{2!} + \frac{\partial^3 u}{\partial x^3}|_i \times \frac{h^3}{3!} + \frac{\partial^4}{\partial x^4}|_i \times \frac{h^4}{4!} + O(h^5) \quad (21)$$

En additionnant les équations (20) et (21), on a :

$$u_{i-1} + u_{i+1} = 2u_i + \frac{\partial^2 u}{\partial x^2} \Big|_i \times h^2 + \frac{\partial^4 u}{\partial x^4} \Big|_i \times \frac{h^4}{12} + O(h^5) \quad (22)$$

En ne prenant en compte que les termes avec un degré inférieur ou égal à 2, on obtient, après un petit arrangement :

$$\frac{\partial^2 u}{\partial x^2} \Big|_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + O(h^2) \quad (23)$$

De la même manière, on prend 3 points sur l'axe y , on trouve :

$$\frac{\partial^2 u}{\partial y^2} \Big|_j = \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + O(h^2) \quad (24)$$

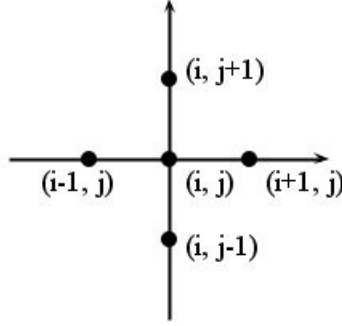


FIGURE 4 – Exemple

En superposant ces 2 équations (23) et (24), on a :

$$\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \Big|_{i,j} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} \quad (25)$$

En substituant l'équation (25) dans l'équation de Poisson (6), et en réorganisant les termes, on obtient finalement :

$$u_{i,j} = \frac{1}{4} (u_{i,j+1} + u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + h^2 f(x, y)) \quad (26)$$

avec $h = a = \frac{1}{n+1}$.

1.3 Construction de la matrice A_N

En utilisant l'équation (19), on peut s'approcher du problème (P2) de l'énoncé. On commence par $n = 2$, puis $n = 3$.

1.3.1 Cas pour $n = 2$

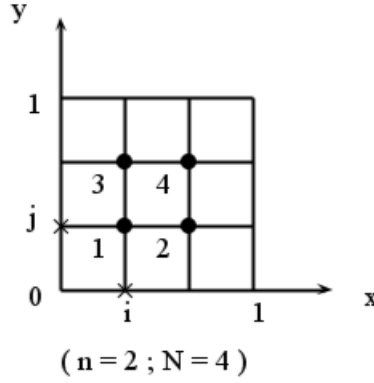


FIGURE 5 – Exemple

Pour trouver u_1 , on fait $u_{i,j} = u_1$. Dans ce cas, on a $u_{i,j+1} = u_3$, $u_{i-1,j} = 0$ (d'après les conditions aux limites données dans (P1)), $u_{i,j-1} = 0$ (idem), $u_{i+1,j} = u_2$. On a donc :

$$u_1 = \frac{1}{4} (u_3 + 0 + 0 + u_2 + a^2 f(x, y))$$

On fait la même chose pour trouver u_2 , u_3 et u_4 . On obtient donc :

$$\begin{cases} u_1 = \frac{1}{4} (u_3 + 0 + 0 + u_2 + a^2 f(x, y)) \\ u_2 = \frac{1}{4} (u_4 + u_1 + 0 + y^2 + a^2 f(x, y)) \\ u_3 = \frac{1}{4} (x^2 + 0 + u_1 + u_4 + a^2 f(x, y)) \\ u_4 = \frac{1}{4} (x^2 + u_3 + u_2 + y^2 + a^2 f(x, y)) \end{cases} \quad (27)$$

Les équations en (27) s'écrivent aussi sous cette forme :

$$\begin{cases} 4u_1 - u_2 - u_3 + 0 \times u_4 = a^2 f(x, y) \\ -u_1 + 4u_2 + 0 \times u_3 - u_4 = y^2 + a^2 f(x, y) \\ -u_1 + 0 \times u_2 + 4u_3 - u_4 = x^2 + a^2 f(x, y) \\ 0 \times u_1 - u_2 - u_3 + 4u_4 = x^2 + y^2 + a^2 f(x, y) \end{cases} \quad (28)$$

Mieux encore, on peut exprimer les équations en (28) sous la forme de matrice :

$$\begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} a^2 f(x, y) \\ y^2 + a^2 f(x, y) \\ x^2 + a^2 f(x, y) \\ x^2 + y^2 + a^2 f(x, y) \end{pmatrix}$$

$$\frac{1}{a^2} \left(\begin{array}{cc|cc} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{array} \right) \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} f(x, y) \\ \frac{y^2}{a^2} + f(x, y) \\ \frac{x^2}{a^2} + f(x, y) \\ \frac{x^2+y^2}{a^2} + f(x, y) \end{pmatrix} \quad (29)$$

Dans ce cas, pour $n = 2$, on a :

$$A_N = \begin{pmatrix} A & B \\ B^T & A \end{pmatrix} \quad \text{et} \quad b = \begin{pmatrix} f(x, y) \\ \frac{y^2}{a^2} + f(x, y) \\ \frac{x^2}{a^2} + f(x, y) \\ \frac{x^2+y^2}{a^2} + f(x, y) \end{pmatrix} \quad (30)$$

avec $A = \begin{pmatrix} 4 & -1 \\ -1 & 4 \end{pmatrix}$ et $B = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$.

1.3.2 Cas pour $n = 3$

En appliquant l'équation (19), on trouve :

$$\begin{cases} u_1 = \frac{1}{4} (u_4 + 0 + 0 + u_2 + a^2 f(x, y)) \\ u_2 = \frac{1}{4} (u_5 + u_1 + 0 + u_3 + a^2 f(x, y)) \\ u_3 = \frac{1}{4} (u_6 + u_2 + 0 + y^2 + a^2 f(x, y)) \\ u_4 = \frac{1}{4} (u_7 + 0 + u_1 + u_5 + a^2 f(x, y)) \\ u_5 = \frac{1}{4} (u_8 + u_4 + u_2 + u_6 + a^2 f(x, y)) \\ u_6 = \frac{1}{4} (u_9 + u_5 + u_3 + y^2 + a^2 f(x, y)) \\ u_7 = \frac{1}{4} (x^2 + 0 + u_4 + u_8 + a^2 f(x, y)) \\ u_8 = \frac{1}{4} (x^2 + u_7 + u_5 + u_9 + a^2 f(x, y)) \\ u_9 = \frac{1}{4} (x^2 + u_8 + u_6 + y^2 + a^2 f(x, y)) \end{cases} \quad (31)$$

qui s'écrivent aussi :

$$\begin{cases} 4u_1 - u_2 + 0 \times u_3 - u_4 + 0 \times u_5 + 0 \times u_6 + 0 \times u_7 + 0 \times u_8 + 0 \times u_9 = a^2 f(x, y) \\ -u_1 + 4u_2 - u_3 + 0 \times u_4 - u_5 + 0 \times u_6 + 0 \times u_7 + 0 \times u_8 + 0 \times u_9 = a^2 f(x, y) \\ 0 \times u_1 - u_2 + 4u_3 + 0 \times u_4 + 0 \times u_5 - u_6 + 0 \times u_7 + 0 \times u_8 + 0 \times u_9 = y^2 + a^2 f(x, y) \\ -u_1 + 0 \times u_2 + 0 \times u_3 + 4u_4 - u_5 + 0 \times u_6 - u_7 + 0 \times u_8 + 0 \times u_9 = a^2 f(x, y) \\ 0 \times u_1 - u_2 + 0 \times u_3 - u_4 + 4u_5 - u_6 + 0 \times u_7 - u_8 + 0 \times u_9 = a^2 f(x, y) \\ 0 \times u_1 + 0 \times u_2 - u_3 + 0 \times u_4 - u_5 + 4u_6 + 0 \times u_7 + 0 \times u_8 - u_9 = y^2 + a^2 f(x, y) \\ 0 \times u_1 + 0 \times u_2 + 0 \times u_3 - u_4 + 0 \times u_5 + 0 \times u_6 + 4u_7 - u_8 + 0 \times u_9 = x^2 + a^2 f(x, y) \\ 0 \times u_1 + 0 \times u_2 + 0 \times u_3 + 0 \times u_4 - u_5 + 0 \times u_6 - u_7 + 4u_8 - u_9 = x^2 + a^2 f(x, y) \\ 0 \times u_1 + 0 \times u_2 + 0 \times u_3 + 0 \times u_4 + 0 \times u_5 - u_6 + 0 \times u_7 - u_8 + 4u_9 = x^2 + y^2 + a^2 f(x, y) \end{cases} \quad (32)$$

Les équations (32) sous forme de matrice :

$$\frac{1}{a^2} \left(\begin{array}{ccc|ccc|ccc} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ \hline -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ \hline 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{array} \right) \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{pmatrix} = \begin{pmatrix} f(x, y) \\ f(x, y) \\ \frac{y^2}{a^2} + f(x, y) \\ f(x, y) \\ f(x, y) \\ \frac{y^2}{a^2} + f(x, y) \\ \frac{x^2}{a^2} + f(x, y) \\ \frac{x^2}{a^2} + f(x, y) \\ \frac{x^2+y^2}{a^2} + f(x, y) \end{pmatrix} \quad (33)$$

Dans ce cas, pour $n = 3$, on a :

$$A_N = \begin{pmatrix} A & B & 0 \\ B^T & A & B \\ 0 & B^T & A \end{pmatrix} \quad \text{et} \quad b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \quad (34)$$

$$\text{avec } A = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix}, B = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, b_1 = \begin{pmatrix} f(x, y) \\ f(x, y) \\ \frac{y^2}{a^2} + f(x, y) \end{pmatrix} = b_2 \text{ et } b_3 = \frac{x^2}{a^2} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + b_1.$$

1.4 Généralisation

1.4.1 La matrice A_N

D'après ce qui précède, on a :

$$A_N = \begin{pmatrix} A & B & 0 & \cdots & 0 \\ B^T & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & A & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & B \\ 0 & \cdots & 0 & B^T & A \end{pmatrix} \in \mathbb{R}^{N \times N} \quad (35)$$

avec A et B des matrices de taille $n \times n$ données par

$$A = \begin{pmatrix} 4 & -1 & 0 & \cdots & 0 \\ -1 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & 4 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 4 \end{pmatrix} \quad B = \begin{pmatrix} -1 & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & -1 & \ddots & \vdots \\ 0 & & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & -1 \end{pmatrix}$$

Remarque : La matrice B qu'on a trouvé est différente de celle donnée dans l'énoncé. La valeur sur la n -ième ligne et la première colonne de notre matrice B est 0 au lieu de -1 . D'après confirmation de M. BOYAVAL, la notre est la bonne.

1.4.2 Le vecteur $b \in \mathbb{R}^N$

Le vecteur b peut s'écrire sous la forme :

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \in \mathbb{R}^N \quad (36)$$

avec $b_i = \begin{pmatrix} f(x, y) \\ \vdots \\ f(x, y) \\ \frac{y^2}{a^2} + f(x, y) \end{pmatrix} \in \mathbb{R}^n$, pour $i \in [1, n-1]$, et $b_n = \frac{x^2}{a^2} \begin{pmatrix} 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix} + b_1$.

1.4.3 Forme générale du problème (P2)

On arrive à une forme générale du problème (P2) qui s'écrit :

$$\frac{1}{a^2} A_N U = b \quad (37)$$

Or, d'après l'équation (17), on a $a = \frac{1}{n+1}$, d'où :

$$(n+1)^2 A_N U = b \quad (38)$$

Remarque : Cette équation (38) est différente de celle donnée dans le problème (P2) de l'énoncé. D'après confirmation de M. BOYAVAL, cette équation (38) est correcte.

2 Propriétés de la matrice A_N et sa décomposition LU

2.1 Propriétés de la matrice A_N

Pour simplifier l'étude des propriétés de la matrice A_N , on commence avec $n = 2$, puis on généralise pour un n grand.

2.1.1 Symétrique

Pour $n = 2$, on a $A_N = \begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix}$. On voit immédiatement que $A_N = A_N^T$. Cette propriété est toujours vraie pour n grand. Donc, la matrice A_N est symétrique.

2.1.2 Définie

Soit $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$. On a :

$$\begin{aligned} X^T A_N X &= (x_1 \ x_2 \ x_3 \ x_4) \begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \\ X^T A_N X &= 4x_1^2 + 4x_2^2 + 4x_3^2 + 4x_4^2 - 2x_1x_2 - 2x_1x_3 - 2x_2x_4 - 2x_3x_4 \\ X^T A_N X &= 2x_1^2 + (x_1 - x_2)^2 + 2x_2^2 + (x_1 - x_3)^2 + 2x_3^2 + (x_2 - x_4)^2 + 2x_4^2 + (x_3 - x_4)^2 \end{aligned} \quad (39)$$

D'après l'équation (39), on en déduit facilement que :

$$X^T A_N X = 0 \Rightarrow X = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Ceci est toujours vrai pour n grand. Donc, la matrice A_N est bien définie.

2.1.3 Positive

D'après l'équation (39), on peut aussi dire que pour $X \neq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$, on a $X^T A_N X > 0$ puisque $X^T A_N X$ peut s'écrire sous la forme d'une somme de carrés. Ceci est toujours vrai pour n grand.

Donc, A_N est définie positive.

2.1.4 SDP

D'après ce qui précède, la matrice A_N est symétrique définie positive. Les propriétés¹ suivantes sont communes aux matrices symétriques réelles :

1. Toute matrice définie positive est inversible (à déterminant réel strictement positif), et son inverse est elle aussi définie positive.
2. Si M est définie positive et r est un nombre réel strictement positif, alors rM est définie positive.
3. Si M et N sont définies positives, alors $M + N$ est définie positive.
4. Si M et N sont définies positives, et si $MN = NM$ (on dit qu'elles commutent), alors MN est définie positive.
5. Une matrice M est définie positive si et seulement s'il existe une matrice définie positive A telle que $A^2 = M$; dans ce cas, la matrice définie positive A est unique, et on peut la noter $A = M^{1/2}$.

1. Ces propriétés sont tirées de Wikipédia. Source : http://fr.wikipedia.org/wiki/Matrice_d%C3%A9finie_positive#Propri.C3.A9t.C3.A9s

2.2 Décomposition LU

Comme toute matrice SDP (symétrique définie positive) est inversible, on peut donc appliquer le théorème suivant :

Théorème : $\forall A \in \mathbb{R}^{n \times n}$ inversible, $\exists M \in \mathbb{R}^{n \times n}$ inversible telle que MA soit triangulaire supérieure.

On a alors la décomposition LU de $A_N = LU$ où $L = M^{-1}$ et $U = MA$ avec L (*lower* en anglais) la matrice triangulaire inférieure et U (*upper* en anglais) la matrice triangulaire supérieure.

On choisit L telle qu'il n'y ait que des 1 sur sa diagonale.

2.2.1 Décomposition à la main avec $n = 2$

En utilisant la méthode du pivot de Gauss, on peut décomposer la matrice A_N en la multipliant par $P_{N-1}E_{N-1} \cdots P_2E_2P_1E_1$ avec P_k les matrices de permutation. Dans notre cas, on veut que $P_k = I_N$ avec I_N la matrice identité de taille $n^2 = N = 4$.

D'où $U = E_{N-1} \cdots E_2E_1A_N$ avec $L^{-1} = E_{N-1} \cdots E_2E_1$.

Cherchons les matrices E_1 , E_2 et E_3 pour $A_N = \begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix}$.

On a :

$$E_1A_N = \begin{pmatrix} 4 & -1 & -1 & 0 \\ 0 & \frac{15}{4} & -\frac{1}{4} & -1 \\ 0 & -\frac{1}{4} & \frac{15}{4} & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix} \Rightarrow E_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{4} & 1 & 0 & 0 \\ \frac{1}{4} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$E_2E_1A_N = \begin{pmatrix} 4 & -1 & -1 & 0 \\ 0 & \frac{15}{4} & -\frac{1}{4} & -1 \\ 0 & 0 & \frac{56}{15} & -\frac{16}{15} \\ 0 & 0 & -\frac{16}{15} & \frac{56}{15} \end{pmatrix} \Rightarrow E_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \frac{1}{15} & 1 & 0 \\ 0 & \frac{4}{15} & 0 & 1 \end{pmatrix}$$

$$E_3E_2E_1A_N = \begin{pmatrix} 4 & -1 & -1 & 0 \\ 0 & \frac{15}{4} & -\frac{1}{4} & -1 \\ 0 & 0 & \frac{56}{15} & -\frac{16}{15} \\ 0 & 0 & 0 & \frac{24}{7} \end{pmatrix} \Rightarrow E_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{16}{56} & 1 \end{pmatrix}$$

A la fin de ces opérations, on obtient $U = E_3E_2E_1A_N = \begin{pmatrix} 4 & -1 & -1 & 0 \\ 0 & \frac{15}{4} & -\frac{1}{4} & -1 \\ 0 & 0 & \frac{56}{15} & -\frac{16}{15} \\ 0 & 0 & 0 & \frac{24}{7} \end{pmatrix}$. La matrice L est obtenue en

inversant les matrices E_1 , E_2 et E_3 car :

$$L = (E_3E_2E_1)^{-1} = E_1^{-1}E_2^{-1}E_3^{-1} \quad (40)$$

Or, les matrices E_1 , E_2 et E_3 sont relativement facile à inverser² : il suffit de changer le signe pour les éléments en dessous de la diagonale. On a donc :

$$E_1^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{4} & 1 & 0 & 0 \\ -\frac{1}{4} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$E_2^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -\frac{1}{15} & 1 & 0 \\ 0 & -\frac{4}{15} & 0 & 1 \end{pmatrix}$$

2. Livre analyse numérique (11258 A02 65) à la bibliothèque ESIEE

$$E_3^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{16}{56} & 1 \end{pmatrix}$$

Et finalement :

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{4} & 1 & 0 & 0 \\ -\frac{1}{4} & -\frac{1}{15} & 1 & 0 \\ 0 & -\frac{1}{15} & -\frac{16}{56} & 1 \end{pmatrix}$$

D'où :

$$A_N = \begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix} = LU = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{4} & 1 & 0 & 0 \\ -\frac{1}{4} & -\frac{1}{15} & 1 & 0 \\ 0 & -\frac{1}{15} & -\frac{16}{56} & 1 \end{pmatrix} \begin{pmatrix} 4 & -1 & -1 & 0 \\ 0 & \frac{15}{4} & -\frac{1}{4} & -1 \\ 0 & 0 & \frac{56}{15} & -\frac{16}{15} \\ 0 & 0 & 0 & \frac{24}{7} \end{pmatrix} \quad (41)$$

2.2.2 Décomposition LU à l'aide d'un ordinateur

On va maintenant utiliser un ordinateur pour trouver la décomposition LU de A_N avec n grand. On décide de travailler sous Matlab pour la partie de programmation.

Principe du programme

D'après ce qu'on a fait précédemment, on en déduit une formule générale pour trouver les éléments de la matrice L :

$$l_{ij} = \begin{cases} \frac{a_{ij}^{(j)}}{a_{jj}^{(j)}} & \text{si } i > j \\ 1 & \text{si } i = j \\ 0 & \text{si } i < j \end{cases}$$

De cette manière, on construit la matrice L colonne par colonne.

Pour la matrice U , elle est obtenue à la fin de N opérations d'élimination de Gauss comme illustré ci-dessous :

$$U^{(1)} = A_N = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & \cdots & a_{1N}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \cdots & \cdots & a_{2N}^{(1)} \\ \vdots & \vdots & & & \vdots \\ a_{N1}^{(1)} & a_{N2}^{(1)} & \cdots & \cdots & a_{NN}^{(1)} \end{pmatrix}$$

$$U^{(2)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & \cdots & a_{1N}^{(1)} \\ 0 & a_{22}^{(1)} - l_{21} \times a_{12}^{(1)} & \cdots & \cdots & a_{2N}^{(1)} - l_{21} \times a_{1N}^{(1)} \\ \vdots & \vdots & & & \vdots \\ 0 & a_{N2}^{(1)} - l_{N1} \times a_{12}^{(1)} & \cdots & \cdots & a_{NN}^{(1)} - l_{N1} \times a_{1N}^{(1)} \end{pmatrix}$$

$$\vdots$$

$$U^{(N)} = \begin{pmatrix} a_{11}^{(N-1)} & a_{12}^{(N-1)} & \cdots & a_{1,N-1}^{(N-1)} & a_{1N}^{(N-1)} \\ 0 & a_{22}^{(N-1)} & \cdots & a_{2,N-1}^{(N-1)} & a_{2N}^{(N-1)} \\ \vdots & 0 & \ddots & \vdots & \vdots \\ \vdots & \vdots & & a_{N-1,N-1}^{(N-1)} & a_{N-1,N}^{(N-1)} \\ 0 & 0 & \cdots & 0 & a_{NN}^{(N-1)} - l_{NN} \times a_{N-1,N}^{(N-1)} \end{pmatrix} = U$$

Ou avec la formule générale :

$$a_{ij}^{(k+1)} = \begin{cases} a_{ij}^{(k)} & \text{si } i \leq j \\ a_{ij}^{(k)} - l_{ik} \times a_{kj}^{(k)} & \text{si } i \in [k+1, N] \text{ et } j \in [k, N] \end{cases}$$

Listing des programmes

On écrit d'abord la fonction *construire_A_N()* qui construit la matrice A_N . Voici le code :

Algorithm 1 Le programme qui construit la matrice A_N

```
% fonction qui construit la matrice A_N
% programmé 090501
function [A_N] = construire_A_N(n)
% initialiser A_N
A_N=zeros(n*n,n*n);
% contruire la matrice A
A=4*eye(n,n)-diag(ones(n-1,1),1)-diag(ones(n-1,1),-1)
% contruire la matrice B
B=-eye(n,n);
for i=0 :n-1
    A_N(i*n+1 :i*n+n,i*n+1 :i*n+n)=A;
end
for j=1 :n-1
    A_N(j*n+1 :j*n+n,(j-1)*n+1 :j*n)=B';
    A_N((j-1)*n+1 :j*n,j*n+1 :j*n+n)=B;
end
```

On a écrit 2 versions de fonctions *LU2()* et *LU3()* qui calculent la décomposition *LU* d'une matrice. Voici le code :

Algorithm 2 La fonction qui calcule la décomposition LU d'une matrice (Version 1)

```
% code inspiré par "Analyse numérique - Michelle Schatzman", 16312 A 02 65,  
% page 40, programmé le 090430  
% ce fichier sert à calculer les matrices de décomposition LU  
% complexité  $n^3$   
function [L,U]=LU2(A)  
[m,n]=size(A);  
U=A;  
L=eye(n,n);  
for k=1 :n  
    % construire la matrice L  
    for i=k+1 :n  
        L(i,k)=U(i,k)/U(k,k);  
    end  
    % construire la matrice U  
    for ii=k+1 :n  
        for j=k+1 :n  
            U(ii,j)=U(ii,j)-U(ii,k)*U(k,j)/U(k,k);  
        end  
        U(ii,k)=0;  
    end  
end  
L=L  
U=U  
% vérification  
LU=L*U
```

Algorithm 3 La fonction qui calcule la décomposition LU d'une matrice (Version 2)

```
% méthode vue en TD4
% modifié le 090501
% décomposition LU
% complexité  $n^2$ 
function [L,U]=LU3(A)
U=A;
[m,n]=size(A);
L=eye(n,n);
for k=1 :n-1
    u=U(k,k :n);
    pivot=U(k,k);
    % construire la matrice L
    for i=k+1 :n
        L(i,k)=U(i,k)/U(k,k);
    end
    % construire la matrice U
    for j=k+1 :n
        s=U(j,k)/pivot;
        v=U(j,k :n);
        U(j,k :n)=v-s*u;
    end
end
end
L=L
U=U
% vérification
LU=L*U
```

Résultats

On lance les programmes pour $n = 3$ et on obtient les résultats suivants :

```
>> A_N=construire_A_N(3)
A =
     4     -1     0
    -1     4     -1
     0     -1     4
A_N =
     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
>> LU2(A_N)
L =
     1.0000         0         0         0         0         0         0         0         0
    -0.2500     1.0000         0         0         0         0         0         0         0
         0    -0.2667     1.0000         0         0         0         0         0         0
    -0.2500    -0.0667    -0.0179     1.0000         0         0         0         0         0
         0    -0.2667    -0.0714    -0.2871     1.0000         0         0         0         0
```

```

      0      0 -0.2679 -0.0048 -0.3160  1.0000      0      0      0
      0      0      0 -0.2679 -0.0843 -0.0282  1.0000      0      0
      0      0      0      0 -0.2935 -0.0932 -0.2950  1.0000      0
      0      0      0      0      0 -0.2948 -0.0076 -0.3284  1.0000
U =
  4.0000 -1.0000      0 -1.0000      0      0      0      0      0
      0  3.7500 -1.0000 -0.2500 -1.0000      0      0      0      0
      0      0  3.7333 -0.0667 -0.2667 -1.0000      0      0      0
      0      0      0  3.7321 -1.0714 -0.0179 -1.0000      0      0
      0      0      0      0  3.4067 -1.0766 -0.2871 -1.0000      0
      0      0      0      0      0  3.3919 -0.0955 -0.3160 -1.0000
      0      0      0      0      0      0  3.7052 -1.0932 -0.0282
      0      0      0      0      0      0      0  3.3545 -1.1015
      0      0      0      0      0      0      0      0  3.3433
LU =
  4.0000 -1.0000      0 -1.0000      0      0      0      0      0
 -1.0000  4.0000 -1.0000      0 -1.0000      0      0      0      0
      0 -1.0000  4.0000      0      0 -1.0000      0      0      0
 -1.0000      0      0  4.0000 -1.0000      0 -1.0000      0      0
      0 -1.0000      0 -1.0000  4.0000 -1.0000      0 -1.0000      0
      0      0 -1.0000      0 -1.0000  4.0000 -0.0000      0 -1.0000
      0      0      0 -1.0000      0      0  4.0000 -1.0000      0
      0      0      0      0 -1.0000      0.0000 -1.0000  4.0000 -1.0000
      0      0      0      0      0      0 -1.0000 -0.0000 -1.0000  4.0000

```

Pour $n = 4$, on a :

```
>> A_N=construire_A_N(4)
```

```
A =
```

```

  4  -1  0  0
 -1  4  -1  0
  0  -1  4  -1
  0  0  -1  4

```

```
A_N =
```

```
Columns 1 through 14
```

```

  4  -1  0  0  -1  0  0  0  0  0  0  0  0  0
 -1  4  -1  0  0  -1  0  0  0  0  0  0  0  0
  0  -1  4  -1  0  0  -1  0  0  0  0  0  0  0
  0  0  -1  4  0  0  0  -1  0  0  0  0  0  0
 -1  0  0  0  4  -1  0  0  -1  0  0  0  0  0
  0  -1  0  0  -1  4  -1  0  0  -1  0  0  0  0
  0  0  -1  0  0  -1  4  -1  0  0  -1  0  0  0
  0  0  0  -1  0  0  -1  4  0  0  0  -1  0  0
  0  0  0  0  -1  0  0  0  4  -1  0  0  -1  0
  0  0  0  0  0  -1  0  0  0  4  -1  0  0  -1
  0  0  0  0  0  0  -1  0  0  -1  4  -1  0  0
  0  0  0  0  0  0  0  -1  0  0  -1  4  0  0
  0  0  0  0  0  0  0  0  -1  0  0  0  4  -1
  0  0  0  0  0  0  0  0  0  -1  0  0  -1  4
  0  0  0  0  0  0  0  0  0  0  -1  0  0  -1
  0  0  0  0  0  0  0  0  0  0  0  -1  0  0

```

```
Columns 15 through 16
```

```

  0  0
  0  0
  0  0
  0  0
  0  0
  0  0
  0  0

```

```

0 0
0 0
0 0
0 0
-1 0
0 -1
0 0
-1 0
4 -1
-1 4

```

```
>> LU2(A_N)
```

```
L =
```

```
Columns 1 through 8
```

```

1.0000 0 0 0 0 0 0 0
-0.2500 1.0000 0 0 0 0 0 0
0 -0.2667 1.0000 0 0 0 0 0
0 0 -0.2679 1.0000 0 0 0 0
-0.2500 -0.0667 -0.0179 -0.0048 1.0000 0 0 0
0 -0.2667 -0.0714 -0.0191 -0.2872 1.0000 0 0
0 0 -0.2679 -0.0718 -0.0051 -0.3178 1.0000 0
0 0 0 -0.2679 -0.0013 -0.0060 -0.3201 1.0000
0 0 0 0 -0.2679 -0.0843 -0.0286 -0.0100
0 0 0 0 0 -0.2937 -0.0943 -0.0318
0 0 0 0 0 0 -0.2968 -0.0945
0 0 0 0 0 0 0 -0.2953
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

```

```
Columns 9 through 16
```

```

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1.0000 0 0 0 0 0 0 0
-0.2954 1.0000 0 0 0 0 0 0
-0.0086 -0.3326 1.0000 0 0 0 0 0
-0.0027 -0.0104 -0.3350 1.0000 0 0 0 0
-0.2699 -0.0882 -0.0324 -0.0125 1.0000 0 0 0
0 -0.2985 -0.1007 -0.0365 -0.2974 1.0000 0 0
0 0 -0.3028 -0.1005 -0.0099 -0.3369 1.0000 0
0 0 0 -0.3000 -0.0034 -0.0121 -0.3392 1.0000

```

```
U =
```

```
Columns 1 through 8
```

```

4.0000 -1.0000 0 0 -1.0000 0 0 0
0 3.7500 -1.0000 0 -0.2500 -1.0000 0 0
0 0 3.7333 -1.0000 -0.0667 -0.2667 -1.0000 0
0 0 0 3.7321 -0.0179 -0.0714 -0.2679 -1.0000
0 0 0 0 3.7321 -1.0718 -0.0191 -0.0048
0 0 0 0 0 3.4051 -1.0821 -0.0205
0 0 0 0 0 0 3.3690 -1.0783
0 0 0 0 0 0 0 3.3868

```

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Columns 9 through 16

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
-1.0000	0	0	0	0	0	0	0	0
-0.2872	-1.0000	0	0	0	0	0	0	0
-0.0964	-0.3178	-1.0000	0	0	0	0	0	0
-0.0339	-0.1077	-0.3201	-1.0000	0	0	0	0	0
3.7047	-1.0945	-0.0318	-0.0100	-1.0000	0	0	0	0
0	3.3496	-1.1139	-0.0348	-0.2954	-1.0000	0	0	0
0	0	3.3022	-1.1062	-0.1068	-0.3326	-1.0000	0	0
0	0	0	3.3338	-0.0416	-0.1218	-0.3350	-1.0000	0
0	0	0	0	3.7000	-1.1005	-0.0365	-0.0125	0
0	0	0	0	0	3.3362	-1.1238	-0.0402	0
0	0	0	0	0	0	3.2846	-1.1142	0
0	0	0	0	0	0	0	3.3216	0

LU =

Columns 1 through 8

4.0000	-1.0000	0	0	-1.0000	0	0	0
-1.0000	4.0000	-1.0000	0	0	-1.0000	0	0
0	-1.0000	4.0000	-1.0000	0	0	-1.0000	0
0	0	-1.0000	4.0000	0	0	0	-1.0000
-1.0000	0	0	0	4.0000	-1.0000	0	0
0	-1.0000	0	0	-1.0000	4.0000	-1.0000	0
0	0	-1.0000	0	0	-1.0000	4.0000	-1.0000
0	0	0	-1.0000	0	0	-1.0000	4.0000
0	0	0	0	-1.0000	-0.0000	0.0000	0
0	0	0	0	0	-1.0000	0	0.0000
0	0	0	0	0	0	-1.0000	0
0	0	0	0	0	0	0	-1.0000
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Columns 9 through 16

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-1.0000	0	0	0	0	0	0	0
0	-1.0000	0	0	0	0	0	0
0	0	-1.0000	0	0	0	0	0
0.0000	0.0000	0	-1.0000	0	0	0	0
4.0000	-1.0000	0	0	-1.0000	0	0	0
-1.0000	4.0000	-1.0000	0	0	-1.0000	0	0
0	-1.0000	4.0000	-1.0000	0	0	-1.0000	0
0	0.0000	-1.0000	4.0000	-0.0000	-0.0000	0	-1.0000

-1.0000	0	-0.0000	0	4.0000	-1.0000	0	0
0	-1.0000	0.0000	-0.0000	-1.0000	4.0000	-1.0000	0
0	0	-1.0000	0	0	-1.0000	4.0000	-1.0000
0	0	0	-1.0000	0	-0.0000	-1.0000	4.0000

On constate que les résultats sont bons après vérification $LU = A_N$.

3 Inversion de la matrice A_N avec une méthode itérative

Après des heures de recherche sur Internet et à la bibliothèque, on s'aperçoit qu'il existe plusieurs méthodes itératives pour calculer l'inverse d'une matrice. Parmi les plus connues sont l'itération de Newton, l'itération de Monte Carlo, etc ...

Ces méthodes sont efficaces mais très délicates à implémenter. Donc, pour résoudre ce problème, on opte pour une autre méthode.

3.1 Méthode itérative d'inversion de Gauss-Seidel

En révisant le cours de MA302, on se rend compte qu'on a déjà à notre disposition 2 méthodes itératives pour inverser une matrice : les méthodes de Jacobi et de Gauss-Seidel.

En fait, puisque les méthodes de Jacobi et de Gauss-Seidel vues en cours sont faites pour résoudre le problème de type :

$$AX = b \quad (42)$$

On peut très bien adapter ce genre problème à notre cas, c'est-à-dire, inverser une matrice. Puisqu'on a :

$$A_N A_N^{-1} = I \quad (43)$$

Donc, la matrice X de l'équation (42) peut être assimilée à A_N^{-1} dans notre cas et idem pour la matrice b qui est assimilée à I .

3.1.1 Programmation de la méthode

On choisit d'implémenter la méthode de Gauss-Seidel.

Listing des programmes pour la méthode itérative d'inversion de Gauss-Seidel

La fonction *GaussSeidelCVG()* qui calcule la convergence d'une matrice.

Algorithm 4 La fonction qui calcule la convergence d'une matrice

```
% test de convergence pour la méthode de Gauss-Seidel
% inspiré par le TD6
% programmé le 090502
function [cvg]=GaussSeidelCVG(A)
% cvg=1 si GS CV; 0 sinon
N=-(triu(A)-diag(diag(A)));
M=A+N;
G=inv(M)*N;
if(max(abs(eig(G)))<1)
    cvg=1;
else cvg=0;
end
```

La fonction *Gauss_Seidel_inverse()* qui calcule l'inverse d'une matrice avec la méthode de Gauss-Seidel.

Algorithm 5 La fonction qui calcule l'inverse d'une matrice avec la méthode de Gauss-Seidel

```
% méthode Gauss-Seidel pour trouver inverse d'une matrice
% AA^-1=I => AX=b avec A^-1=X et I=b
% programmé le 090501
function [X,iter]=Gauss_Seidel_inverse(A,tol,itermax)
[m,n]=size(A);
b=eye(n,n);
X=zeros(n,n);
r=b-A*X;
N=-(triu(A)-diag(diag(A)));
M=A+N;
iter=0;
cvg=GaussSeidelCVG(A);
if(cvg==0) error('Pas de convergence');
else
    while(norm(r)>tol&iter<itermax)
        y=inv(M)*r;
        X=X+y;
        r=r-A*y;
        iter=iter+1;
    end
end
iter=iter
X=X
% vérification
AX=A*X
```

Résultats

On exécute cette fonction pour $n = 4$ avec une tolérance de 0.00001.

```
>> A_N=construire_A_N(4)
A =
     4     -1     0     0
    -1     4     -1     0
     0     -1     4     -1
     0     0     -1     4
A_N =
Columns 1 through 14
     4     -1     0     0     -1     0     0     0     0     0     0     0     0     0
    -1     4     -1     0     0     -1     0     0     0     0     0     0     0     0
     0     -1     4     -1     0     0     -1     0     0     0     0     0     0     0
     0     0     -1     4     0     0     0     -1     0     0     0     0     0     0
    -1     0     0     0     4     -1     0     0     -1     0     0     0     0     0
     0     -1     0     0     -1     4     -1     0     0     -1     0     0     0     0
     0     0     -1     0     0     -1     4     -1     0     0     -1     0     0     0
     0     0     0     -1     0     0     -1     4     0     0     0     -1     0     0
     0     0     0     0     -1     0     0     0     4     -1     0     0     -1     0
     0     0     0     0     0     -1     0     0     -1     4     -1     0     0     -1
     0     0     0     0     0     0     -1     0     0     -1     4     -1     0     0
     0     0     0     0     0     0     0     -1     0     0     -1     4     0     0
     0     0     0     0     0     0     0     0     -1     0     0     0     4     -1
     0     0     0     0     0     0     0     0     0     -1     0     0     -1     4
     0     0     0     0     0     0     0     0     0     0     -1     0     0     -1
     0     0     0     0     0     0     0     0     0     0     0     -1     0     0
Columns 15 through 16
     0     0
```

```

0    0
0    0
0    0
0    0
0    0
0    0
0    0
0    0
0    0
0    0
-1   0
0   -1
0    0
-1   0
4   -1
-1   4
>> Gauss_Seidel_inverse(A_N,0.00001,100)
iter =
29
X =
Columns 1 through 8
0.3011    0.1021    0.0380    0.0133    0.1021    0.0694    0.0367    0.0153
0.1021    0.3391    0.1155    0.0380    0.0694    0.1388    0.0847    0.0367
0.0380    0.1155    0.3391    0.1021    0.0367    0.0847    0.1388    0.0694
0.0133    0.0380    0.1021    0.3011    0.0153    0.0367    0.0694    0.1021
0.1021    0.0694    0.0367    0.0153    0.3391    0.1388    0.0620    0.0245
0.0694    0.1388    0.0847    0.0367    0.1388    0.4011    0.1633    0.0620
0.0367    0.0847    0.1388    0.0694    0.0620    0.1633    0.4011    0.1388
0.0153    0.0367    0.0694    0.1021    0.0245    0.0620    0.1388    0.3391
0.0380    0.0367    0.0239    0.0112    0.1155    0.0847    0.0479    0.0209
0.0367    0.0620    0.0479    0.0239    0.0847    0.1633    0.1056    0.0479
0.0239    0.0479    0.0620    0.0367    0.0479    0.1056    0.1633    0.0847
0.0112    0.0239    0.0367    0.0380    0.0209    0.0479    0.0847    0.1155
0.0133    0.0153    0.0112    0.0056    0.0380    0.0367    0.0239    0.0112
0.0153    0.0245    0.0209    0.0112    0.0367    0.0620    0.0479    0.0239
0.0112    0.0209    0.0245    0.0153    0.0239    0.0479    0.0620    0.0367
0.0056    0.0112    0.0153    0.0133    0.0112    0.0239    0.0367    0.0380
Columns 9 through 16
0.0380    0.0367    0.0239    0.0112    0.0133    0.0153    0.0112    0.0056
0.0367    0.0620    0.0479    0.0239    0.0153    0.0245    0.0209    0.0112
0.0239    0.0479    0.0620    0.0367    0.0112    0.0209    0.0245    0.0153
0.0112    0.0239    0.0367    0.0380    0.0056    0.0112    0.0153    0.0133
0.1155    0.0847    0.0479    0.0209    0.0380    0.0367    0.0239    0.0112
0.0847    0.1633    0.1056    0.0479    0.0367    0.0620    0.0479    0.0239
0.0479    0.1056    0.1633    0.0847    0.0239    0.0479    0.0620    0.0367
0.0209    0.0479    0.0847    0.1155    0.0112    0.0239    0.0367    0.0380
0.3391    0.1388    0.0620    0.0245    0.1021    0.0694    0.0367    0.0153
0.1388    0.4011    0.1633    0.0620    0.0694    0.1388    0.0847    0.0367
0.0620    0.1633    0.4011    0.1388    0.0367    0.0847    0.1388    0.0694
0.0245    0.0620    0.1388    0.3391    0.0153    0.0367    0.0694    0.1021
0.1021    0.0694    0.0367    0.0153    0.3011    0.1021    0.0380    0.0133
0.0694    0.1388    0.0847    0.0367    0.1021    0.3391    0.1155    0.0380
0.0367    0.0847    0.1388    0.0694    0.0380    0.1155    0.3391    0.1021
0.0153    0.0367    0.0694    0.1021    0.0133    0.0380    0.1021    0.3011
AX =
Columns 1 through 8
1.0000    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000    -0.0000

```

Algorithm 7 La fonction qui calcule l'inverse d'une matrice

```
% méthode Jacobi pour trouver l'inverse d'une matrice
% AA^-1=I => AX=b avec A^-1=X et I=b
% programmé le 090508
function [X,iter]=Jacobi_inverse(A,tol,itermax)
[m,n]=size(A);
b=eye(n,n);
X=zeros(n,n);
r=b-A*X;
M=diag(diag(A));
N=M-A;
iter=0;
cvg=JacobiCVG(A);
if(cvg==0) error('Pas de convergence');
else
    while(norm(r)>tol&iter<itermax)
        y=inv(M)*r;
        X=X+y;
        r=r-A*y;
        iter=iter+1;
    end
end
iter=iter
X=X
% vérification
AX=A*X
```

3.2 Méthode directe d'inversion

3.2.1 Pivot de Gauss

Cette méthode calcule l'inverse d'une matrice en utilisant l'élimination de Gauss. Elle marche suivant le principe suivant :

$$[A \mid I]$$

On applique la méthode d'élimination de Gauss pour arriver à la configuration suivante :

$$[I \mid A^{-1}]$$

Remarque : Cette fonction *calcul_inverse_gauss()* est récupérée d'un site internet³. On ne l'a pas codée.

3. Source : <http://www.commentcamarche.net/forum/affich-6254366-methode-calculant-l-inverse-d-une-matrice>

Algorithm 8 Méthode directe de Gauss pour calculer l'inverse d'une matrice

```
% source : http://www.commentcamarche.net/forum/affich-6254366-methode-alcu
% nt-l-inverse-d-une-matrice
% Date : 090501
function B = calcul_inverse_gauss(A)
n=size(A,1);
B=eye(n);
for p=1 :n
    vec=[(1 :p-1) n (p :n-1)];
    test=1;
    while A(p,p)==0
        if test==n
            error('La matrice n"est pas inversible')
        end
        A=A(vec, :);
        B=B(vec, :);
        test=test+1;
    end
    B(p, :)=B(p, :)/A(p,p);
    A(p, :)=A(p, :)/A(p,p);
    for q=p+1 :n
        B(q, :)=B(q, :)-A(q,p)*B(p, :);
        A(q, :)=A(q, :)-A(q,p)*A(p, :);
    end
end
for p=n :-1 :2
    for q=p-1 :-1 :1
        B(q, :)=B(q, :)-A(q,p)*B(p, :);
        A(q, :)=A(q, :)-A(q,p)*A(p, :);
    end
end
end
```

3.2.2 Méthode d'inversion de LU

On a aussi une autre possibilité pour inverser une matrice en utilisant une méthode directe. Le principe est le suivant :

$$A = LU \Rightarrow A^{-1} = (LU)^{-1} = U^{-1}L^{-1}$$

On inverse d'abord les matrices triangulaires L et U qui sont 'faciles' à inverser, puis on les multiplie pour avoir l'inverse de la matrice originale.

Listing des programmes

La fonction `inverse_L()` qui inverse une matrice triangulaire inférieure.

Algorithm 9 La fonction qui inverse une matrice triangulaire inférieure

```
% inspiré d'une doc "Stability of Methods for Matrix Inversion"
% inverser une matrice triangulaire inférieure
% programmé le 090502
function [X]= inverse_L(L)
[m,n]=size(L);
for j=n :-1 :1
    X(j,j)=1/L(j,j);
    X(j+1 :n,j)=X(j+1 :n,j+1 :n)*L(j+1 :n,j);
    X(j+1 :n,j)=-X(j,j)*X(j+1 :n,j);
end
```

La fonction *inverse_U* qui inverse une matrice triangulaire supérieure.

Algorithm 10 La fonction qui inverse une matrice triangulaire supérieure

```
% inspiré d'une doc "Stability of Methods for Matrix Inversion"
% inverser une matrice triangulaire supérieure
% programmé le 090502
function [X]= inverse_U(U)
[m,n]=size(U);
UT=U';
X=zeros(n,n);
for j=n :-1 :1
    X(j,j)=1/UT(j,j);
    X(j+1 :n,j)=X(j+1 :n,j+1 :n)*UT(j+1 :n,j);
    X(j+1 :n,j)=-X(j,j)*X(j+1 :n,j);
end
X=X';
```

La fonction *inverse_LU()* qui inverse une matrice par l'intermédiaire de la décomposition *LU*.

Algorithm 11 La fonction qui calcule l'inverse d'une matrice par la décomposition *LU*

```
% fonction qui calcule l'inverse d'une matrice par la décomposition LU
% programmé le 090508
function [X]=inverse_LU(A)
[L,U]=LU3(A);
X=inverse_U(U)*inverse_L(L);
```

3.3 Comparaison entre les méthodes itératives et les méthodes directes pour inverser A_N

Ici, on va comparer numériquement la convergence et le temps de calcul pour différentes valeurs de n où $N = n^2$.

3.3.1 Programme de test

On écrit un programme sous Matlab pour tester toutes les méthodes listées précédemment.

Algorithm 12 Le programme qui calcule le temps de calcul pour toutes les méthodes

```
% programme qui mesure le temps de calcul et teste la convergence
% codé le 090508
clear;
nmax=15;
% déclarer et initialiser les variables pour stocker le temps écoulé pour
% chaque méthode
tGS=zeros(1,nmax);
tJ=zeros(1,nmax);
tG=zeros(1,nmax);
tLU=zeros(1,nmax);
t_ref=[1 :1 :nmax]
% test pour la méthode de Gauss-Seidel
for i=2 :nmax
    A=construire_A_N(i);
    tic;
    X=Gauss_Seidel_inverse(A,0.00001,1000);
    tGS(i)=toc;
    if (norm(A*X-eye(i*i,i*i))>0.00001)
        error('Pas de convergence');
    end
end
% test pour la méthode de Jacobi
for i=2 :nmax
    A=construire_A_N(i);
    tic;
    X=Jacobi_inverse(A,0.00001,1000);
    tJ(i)=toc;
    if (norm(A*X-eye(i*i,i*i))>0.00001)
        error('Pas de convergence');
    end
end
% test pour la méthode directe de Gauss
for i=2 :nmax
    A=construire_A_N(i);
    tic;
    X=calcul_inverse_gauss(A);
    tG(i)=toc;
    if (norm(A*X-eye(i*i,i*i))>0.00001)
        error('Pas de convergence');
    end
end
% test pour la méthode directe par la décomposition LU
for i=2 :nmax
    A=construire_A_N(i);
    tic;
    X=inverse_LU(A);
    tLU(i)=toc;
    if (norm(A*X-eye(i*i,i*i))>0.00001)
        error('Pas de convergence');
    end
end
plot(t_ref,tGS,'r-',t_ref,tJ,'b+',t_ref,tG,'gx',t_ref,tLU,'c*');
legend('Gauss-Seidel','Jacobi','Gauss direct','LU');
xlabel('n');
ylabel('Temps en seconde');
title('Temps de calcul');
```

Résultats

Voici les résultats obtenus pour $n = 2$ jusqu'à $n = 15$ (ou $N = 4$ jusqu'à $N = 225$) si on applique les méthodes telles quelles.

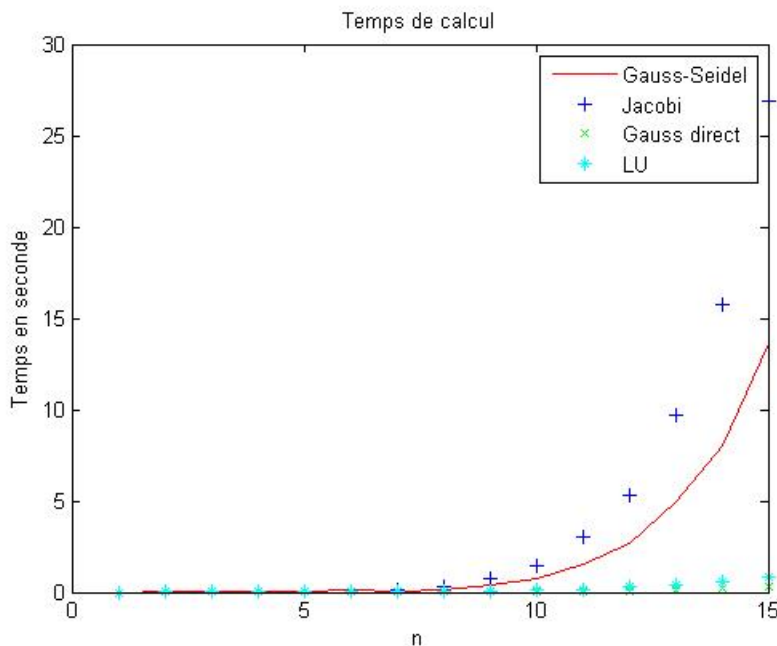


FIGURE 6 – Les résultats de temps de calcul

Observations

D'après les résultats obtenus, on voit que les méthodes de Gauss-Seidel et de Jacobi convergent. Or, elles ne sont pas optimales pour calculer l'inverse d'une matrice si on les compare avec des méthodes directes. Le temps de calcul augmente exponentiellement pour N grand (supérieure à 225).

En général, l'avantage d'une méthode itérative est de minimiser le temps de calcul pour la résolution d'un problème. Elle est faite pour remplacer la méthode directe qui devient impossible à utiliser quand on veut résoudre un problème avec une matrice de dimension très grande. Mais ici, c'est le contraire. Le temps de calcul des méthodes itératives qu'on a proposées croît plus vite que celui des méthodes directes. Il faut donc rechercher encore.

3.4 Amélioration

3.4.1 Méthode itérative de Newton

Pour améliorer la méthode itérative d'inversion de la matrice A_N , on opte pour une autre solution qui est la méthode itérative de Newton⁴.

Le principe de cette méthode est comme suivant :

Soit A un réel quelconque. Son inverse est X tel que $A = \frac{1}{X} \Rightarrow 0 = \frac{1}{X} - A$. Or, le principe de Newton-Raphson nous donne :

$$\begin{aligned} X_{n+1} &= X_n - \frac{\left(\frac{1}{X_n} - A\right)}{\left(-\frac{1}{X_n^2}\right)} \\ X_{n+1} &= X_n + X_n(1 - AX_n) \\ X_{n+1} &= X_n(2 \times 1 - AX_n) \end{aligned} \quad (44)$$

4. Pour plus de détails, voir le site http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19660018719_1966018719.pdf

Il est évident que si la méthode converge, le deuxième terme à droite de l'équation (44) va s'annuler ou tend vers zéro, ce qui signifie que la valeur de X_n va s'approcher de l'inverse de A .

Pour trouver l'inverse d'une matrice, il suffit de remplacer A par une matrice et de choisir judicieusement la matrice inverse initiale X_0 . La méthode converge rapidement vers la solution si la matrice initiale X_0 remplit la condition suivante :

Soit λ_{max} la plus grande valeur propre de AA^T et $X_0 = \alpha A^T$, où $0 < \alpha < \frac{1}{\lambda_{max}}$.

Algorithm 13 La méthode itérative de Newton

```
% méthode itérative d'inversion de Newton
% inspiré par les sources d'Internet :
% -http://comet.lehman.cuny.edu/vpan/pdf/201.pdf
% -http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19660018719_1966018
% 719.pdf
% programmé 090508
function [X,iter]=newton_iter_inverse(A,tol,itermax)
[m,n]=size(A);
X=1/(max(abs(eig(A*A')))+1)*A';
I=eye(m,n);
iter=0;
while(iter<itermax&&norm(I-A*X)>tol)
    X=X*(2*I-A*X);
    iter=iter+1;
end
```

Programme de test

On compare cette méthode avec des méthodes directes (élimination de Gauss, décomposition LU) et aussi la fonction de *inv()* de Matlab.

Algorithm 14 Comparaison entre la méthode itérative de Newton et des méthodes directes

```
% programme qui mesure le temps de calcul et teste la convergence
% codé le 090508
clear;
nmax=20;
% déclarer et initialiser les variables pour stocker le temps écoulé pour
% chaque méthode
tN2=zeros(1,nmax);
tG=zeros(1,nmax);
tLU=zeros(1,nmax);
tMATLAB=zeros(1,nmax);
t_ref=[1 :1 :nmax]
% test pour la méthode de Newton iter inverse 2
for i=2 :nmax
    A=construire_A_N(i);
    tic;
    X=newton_iter_inverse(A,0.00001,1000);
    tN2(i)=toc;
    if (norm(A*X-eye(i*i,i*i))>0.00001)
        error('Pas de convergence');
    end
end
% test pour la méthode directe de Gauss
for i=2 :nmax
    A=construire_A_N(i);
    tic;
    X=calcul_inverse_gauss(A);
    tG(i)=toc;
    if (norm(A*X-eye(i*i,i*i))>0.00001)
        error('Pas de convergence');
    end
end
% test pour la méthode directe par la décomposition LU
for i=2 :nmax
    A=construire_A_N(i);
    tic;
    X=inverse_LU(A);
    tLU(i)=toc;
    if (norm(A*X-eye(i*i,i*i))>0.00001)
        error('Pas de convergence');
    end
end
% test pour la méthode inv() de MATLAB
for i=2 :nmax
    A=construire_A_N(i);
    tic;
    X=inv(A);
    tMATLAB(i)=toc;
    if (norm(A*X-eye(i*i,i*i))>0.00001)
        error('Pas de convergence');
    end
end
plot(t_ref,tN2,'r-',t_ref,tG,'gx',t_ref,tLU,'c*',t_ref,tMATLAB,'b+');
legend('Newton iter','Gauss direct','LU','inv() MATLAB');
xlabel('n');
ylabel('Temps en seconde');
title('Temps de calcul');
```

Résultats

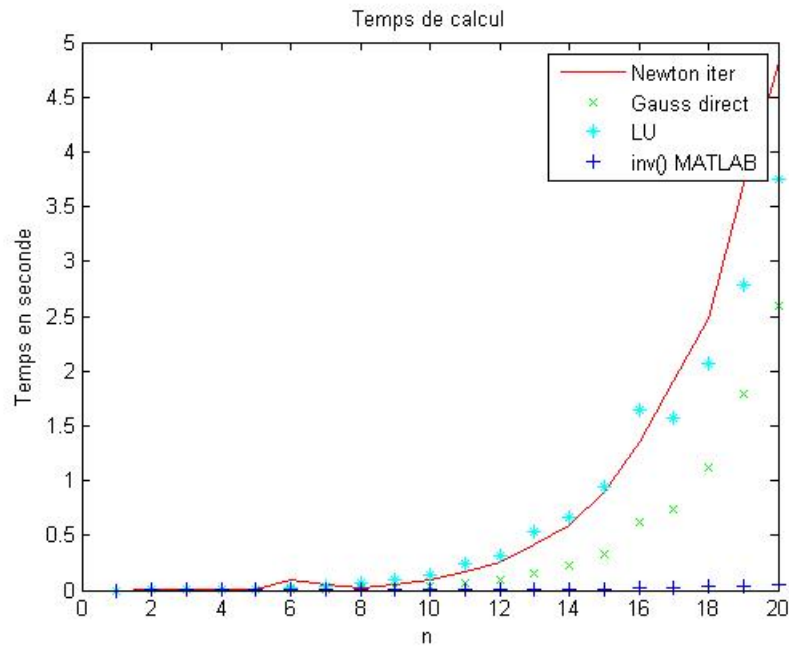


FIGURE 7 – Les résultats du temps de calcul

Observations

On voit que la méthode itérative de Newton est beaucoup plus rapide que les méthodes itératives de Gauss-Seidel et de Jacobi. Son temps de calcul est comparable à celui de la méthode directe par la décomposition LU pour $n = 2$ jusqu'à $n = 20$ (soit $N = 4$ jusqu'à $N = 400$).

Or, elle est loin d'être optimale si on la compare avec les fonctions $inv()$ et $/$ de Matlab.

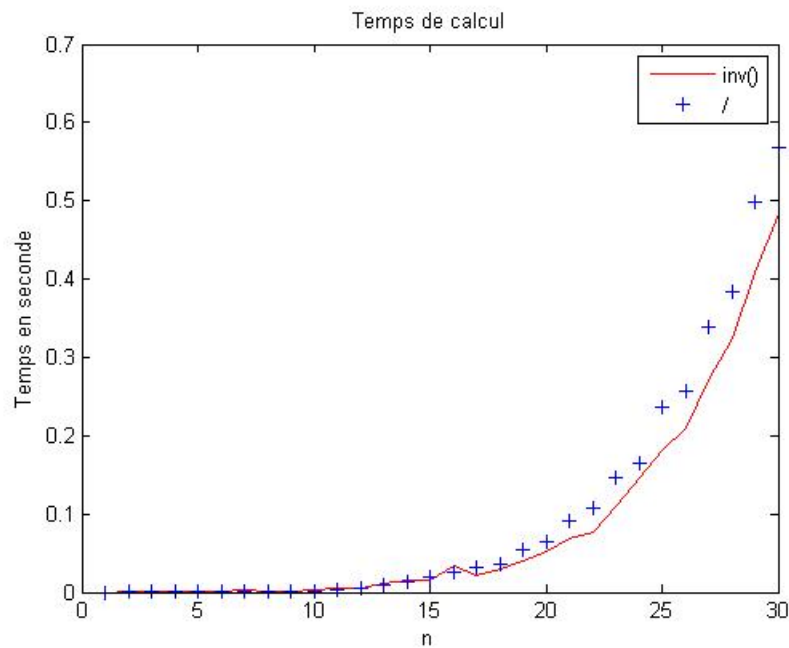


FIGURE 8 – Temps de calcul pour les fonctions de Matlab avec la même matrice A_N

3.5 Modification sur les programmes

Après avoir eu le TD 10 de MA302 avec M. BOYAVAL le 15 mai 2009, on s'est informé que le but de ce projet est d'essayer de programmer les méthodes itératives sans utiliser les fonctions pré-définies dans Matlab ou Scilab pour exécuter les opérations de multiplication ou de manipulation des matrices. Donc, on modifie les programmes précédents et on refait les comparaisons.

3.5.1 Programme de Gauss Seidel modifié pour inverser une matrice

Voici le code du programme modifié.

Algorithm 15 Fonction Gauss-Seidel pour inverser une matrice

```
% méthode de gauss seidel pour inverser une matrice
% cette version de programme n'utilise pas les fonctions de Matlab pour
% manipuler les matrices dans la boucle principale du programme
% programmé 090515
function [u,iter]=gs_inv(A,tol,itermax)
[m,n]=size(A);
u=ones(m,n);
u_old=ones(m,n);
error=zeros(m,n);
b=eye(m,n);
e=0;
fin=0;
for k=1 :itermax
    if fin==0
        for c=1 :n
            for i=1 :m % une colonne
                somme=0;
                for j=1 :i-1
                    somme=somme+A(i,j)*u(j,c);
                    u(i,c)=(b(i,c)-somme)/A(i,i);
                end
                for j=i+1 :m
                    somme=somme+A(i,j)*u(j,c);
                    u(i,c)=(b(i,c)-somme)/A(i,i);
                end
                u(i,c)=(b(i,c)-somme)/A(i,i);
            end
        end
        for c=1 :n
            for i=1 :m
                error(i,c)=abs(u_old(i,c)-u(i,c));
                u_old(i,c)=u(i,c);
            end
        end
        % calcul de la norme 2 de u(n)-u(n-1)
        for c=1 :n
            for i=1 :m
                e=e+error(i,c)*error(i,c);
            end
        end
        iter=k;
        if sqrt(e)<tol
            fin=1;
            break;
        else
            e=0;
        end
    end
end
end
```

3.5.2 Programme de Jacobi modifié pour inverser une matrice

Voici le code du programme modifié.

Algorithm 16 Fonction Jacobi pour inverser une matrice

```
% méthode de jacobi pour inverser une matrice
% cette version de programme n'utilise pas les fonctions de Matlab pour
% manipuler les matrices dans la boucle principale du programme
% programmé 090515
function [u,iter]=jcb_inv(A,tol,itermax)
[m,n]=size(A);
u=ones(m,n);
u_old=ones(m,n);
error=zeros(m,n);
b=eye(m,n);
e=0;
fin=0;
for k=1 :itermax
    if fin==0
        for c=1 :n
            for i=1 :m % une colonne
                u(i,c)=(b(i,c)-(A(i,1 :i-1)*u(1 :i-1,c)+A(i,i+1 :n)*u(i+1 :n,c)))/A(i,i);
            end
        end
        for c=1 :n
            for i=1 :m
                error(i,c)=abs(u_old(i,c)-u(i,c));
                u_old(i,c)=u(i,c);
            end
        end
        % calcul de la norme 2 de u(n)-u(n-1)
        for c=1 :n
            for i=1 :m
                e=e+error(i,c)*error(i,c);
            end
        end
        iter=k;
        if sqrt(e)<tol
            fin=1;
            break;
        else
            e=0;
        end
    end
end
end
```

3.5.3 Programme de LU modifié pour inverser une matrice

Voici le programme modifié.

Algorithm 17 Fonction qui inverse une matrice par la décomposition LU

```
% fonction qui calcule l'inverse d'une matrice par la décomposition LU
% cette version de programme n'utilise pas l'opérateur de Matlab pour
% multiplier les matrices dans la boucle principale du programme
% programmé le 090515
function [X]=inverse_lu_explicite(A)
[m,n]=size(A);
X=zeros(m,n);
[L,U]=LU3(A);
u_inv=inverse_U(U);
l_inv=inverse_L(L);
for i=1 :m
    for j=1 :n
        s=0;
        for k=1 :n
            s=s+u_inv(i,k)*l_inv(k,j);
        end
        X(i,j)=s;
    end
end
end
```

3.5.4 Nouveaux résultats de comparaison

Comme le temps de calcul augmente pour ces méthodes modifiées, on limite la taille de A_N allant de $n = 2$ jusqu'à $n = 10$ (soit $N = 4$ jusqu'à $N = 100$).

Résultats

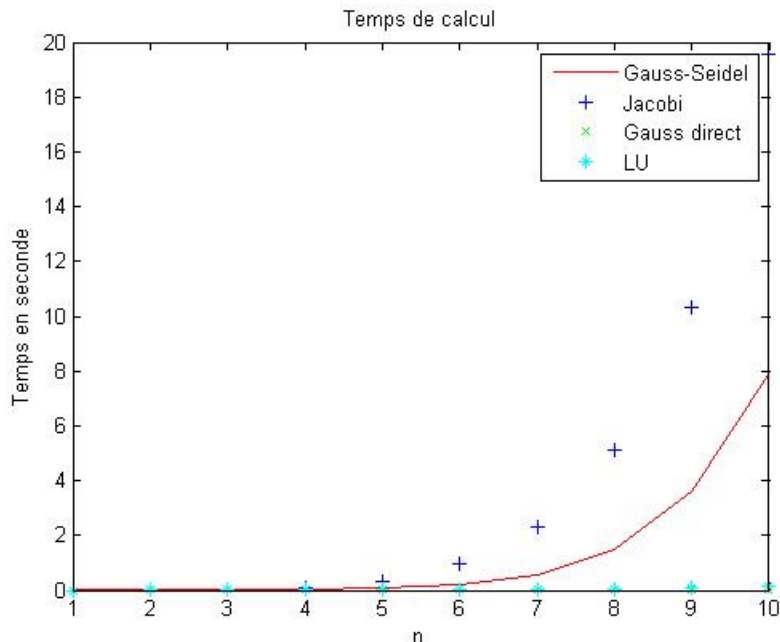


FIGURE 9 – Les résultats du temps de calcul

Observations

On constate que ces dernières versions de programmes sont beaucoup moins efficaces en temps de calcul que les programmes implémentant directement les fonctions pré-définies dans Matlab.

4 Forme quadratique du problème (P2)

Le problème (P2) est équivalent au problème de minimisation de (P3) suivant :

$$(P3) \text{ Trouver } X \in \mathbb{R}^N \text{ minimum de } J(X) = \frac{1}{2}X^T A_N X - \frac{1}{(n+1)^2}X^T b$$

pour $X \in \mathbb{R}^N$, car d'après une dérivation de la fonction quadratique du problème (P3), on peut retrouver la forme de l'équation (38) du problème posé en (P2).

4.1 Dérivation de la fonction quadratique $J(X)$

Après avoir dérivé $J(X)$, on trouve :

$$\nabla J(X) = \frac{1}{2}A_N^T X + \frac{1}{2}A_N X - \frac{1}{(n+1)^2}b \quad (45)$$

Comme A_N est symétrique, on a :

$$\nabla J(X) = A_N X - \frac{1}{(n+1)^2}b \quad (46)$$

Puisque $J(X)$ est elliptique et quadratique d'après le cours, elle admet un minimum où sa dérivée est nulle. Donc, pour $\nabla J(X) = 0$, on retrouve :

$$(n+1)^2 A_N X = b \quad (47)$$

4.2 La solution de qui minimalise la forme quadratique

La solution X de l'équation (47) est une solution qui minimalise la fonction quadratique $J(X)$. En fait, soit e un terme d'erreur, on a :

$$\begin{aligned} J(X+e) &= \frac{1}{2}(X+e)^T A_N (X+e) - \frac{1}{(n+1)^2}(X+e)^T b \\ &= \frac{1}{2}X^T A_N X + e^T A_N X + \frac{1}{2}e^T A_N e - \frac{1}{(n+1)^2}X^T b - \frac{1}{(n+1)^2}e^T b \\ &= \frac{1}{2}X^T A_N X - \frac{1}{(n+1)^2}X^T b + \frac{1}{(n+1)^2}e^T b + \frac{1}{2}e^T A_N e - \frac{1}{(n+1)^2}e^T b \\ &= J(X) + \frac{1}{2}e^T A_N e \end{aligned}$$

Comme A_N est définie positive, le deuxième terme à droite est positif pour tout $e \neq 0$, donc la solution X minimalise $J(X)$.

5 Méthodes numériques pour la résolution du problème (P3)

L'équation (47) peut aussi s'écrire sous la forme :

$$\begin{aligned} A_N X &= \frac{1}{(n+1)^2} b \\ A_N X &= p \end{aligned} \tag{48}$$

où $p = \frac{1}{(n+1)^2} b$.

5.1 Construction du vecteur p

On écrit un programme qui construit automatiquement le vecteur p . Voici le code :

Algorithm 18 Fonction qui construit le vecteur p

```
% la fonction qui construit le vecteur p du problème
% programmé le 090516
function [p]=construire_p(n)
p=zeros(n*n,1);
b=zeros(n,1);
for j=1 :n
    for i=1 :n
        b(i)=calculer_bi(i,j,n);
    end
    p((j-1)*n+1 :j*n)=b/(n+1)^2;
end
function [c]=f(x,y)
c=-2*sin(4*pi*x)-2*x^2-2*y^2+(4*pi)^2*y*(y-1)*sin(4*pi*x);
function [bi]=calculer_bi(i,j,n)
if j<n && i<n
    bi=f(i/(n+1),j/(n+1));
elseif j<n && i==n
    bi=j^2+f(i/(n+1),j/(n+1));
elseif j==n && i<n
    bi=i^2+f(i/(n+1),j/(n+1));
elseif j==n && i==n
    bi=i^2+j^2+f(i/(n+1),j/(n+1));
end
```

5.1.1 Vérification du vecteur p

On exécute la fonction `construire_p()` pour $n = 2$ et $n = 3$, puis on les vérifie par calcul à la main.

Résultats

```
>> p=construire_p(2)
p =
    3.5198
   -3.5815
    3.5568
   -2.8778
>> p=construire_p(3)
p =
   -0.0156
   -0.0391
   -0.0156
```

-0.0391
-0.0625
0.1484
-0.0156
0.1484
0.9844

Vérification à la main

Pour $n = 2$ On a :

$$b = \begin{pmatrix} f(x, y) \\ \frac{y^2}{a^2} + f(x, y) \\ \frac{x^2}{a^2} + f(x, y) \\ \frac{x^2+y^2}{a^2} + f(x, y) \end{pmatrix}$$

$$p = a^2 b = \begin{pmatrix} a^2 f(x, y) \\ y^2 + a^2 f(x, y) \\ x^2 + a^2 f(x, y) \\ x^2 + y^2 + a^2 f(x, y) \end{pmatrix}$$

avec $a = \frac{1}{(n+1)}$. Le vecteur p peut être écrit explicitement sous la forme :

$$p = \begin{pmatrix} \frac{1}{(n+1)^2} \times f\left(\frac{1}{n+1}, \frac{1}{n+1}\right) \\ \frac{1}{(n+1)^2} + \frac{1}{(n+1)^2} \times f\left(\frac{2}{n+1}, \frac{1}{n+1}\right) \\ \frac{1}{(n+1)^2} + \frac{1}{(n+1)^2} \times f\left(\frac{1}{n+1}, \frac{2}{n+1}\right) \\ \left(\frac{2}{n+1}\right)^2 + \left(\frac{2}{n+1}\right)^2 + \frac{1}{(n+1)^2} \times f\left(\frac{2}{n+1}, \frac{2}{n+1}\right) \end{pmatrix}$$

$$p = \begin{pmatrix} \frac{1}{(3)^2} \times f\left(\frac{1}{3}, \frac{1}{3}\right) \\ \frac{1}{(3)^2} + \frac{1}{(3)^2} \times f\left(\frac{2}{3}, \frac{1}{3}\right) \\ \frac{1}{(3)^2} + \frac{1}{(3)^2} \times f\left(\frac{1}{3}, \frac{2}{3}\right) \\ \left(\frac{2}{3}\right)^2 + \left(\frac{2}{3}\right)^2 + \frac{1}{(3)^2} \times f\left(\frac{2}{3}, \frac{2}{3}\right) \end{pmatrix}$$

$$p = \begin{pmatrix} 3.5198 \\ -3.5815 \\ 3.5568 \\ -2.8778 \end{pmatrix}$$

Les résultats sont bons pour $n = 2$.

Pour $n = 3$ On a :

$$p = \begin{pmatrix} \frac{1}{(n+1)^2} \times f(x, y) \\ \frac{1}{(n+1)^2} \times f(x, y) \\ \frac{y^2 + \frac{1}{(n+1)^2} \times f(x, y)}{\frac{1}{(n+1)^2} \times f(x, y)} \\ \frac{y^2 + \frac{1}{(n+1)^2} \times f(x, y)}{\frac{1}{(n+1)^2} \times f(x, y)} \\ \frac{x^2 + \frac{1}{(n+1)^2} \times f(x, y)}{\frac{1}{(n+1)^2} \times f(x, y)} \\ \frac{x^2 + \frac{1}{(n+1)^2} \times f(x, y)}{\frac{1}{(n+1)^2} \times f(x, y)} \\ \frac{x^2 + y^2 + \frac{1}{(n+1)^2} \times f(x, y)}{\frac{1}{(n+1)^2} \times f(x, y)} \end{pmatrix}$$

$$p = \begin{pmatrix} \frac{1}{(4)^2} \times f\left(\frac{1}{4}, \frac{1}{4}\right) \\ \frac{1}{(4)^2} \times f\left(\frac{2}{4}, \frac{1}{4}\right) \\ \frac{1}{(4)^2} + \frac{1}{(4)^2} \times f\left(\frac{3}{4}, \frac{1}{4}\right) \\ \frac{1}{(4)^2} \times f\left(\frac{1}{4}, \frac{2}{4}\right) \\ \frac{1}{(4)^2} \times f\left(\frac{2}{4}, \frac{2}{4}\right) \\ \left(\frac{2}{4}\right)^2 + \frac{1}{(4)^2} \times f\left(\frac{3}{4}, \frac{2}{4}\right) \\ \frac{1}{(4)^2} + \frac{1}{(4)^2} \times f\left(\frac{1}{4}, \frac{3}{4}\right) \\ \left(\frac{2}{4}\right)^2 + \frac{1}{(4)^2} \times f\left(\frac{2}{4}, \frac{3}{4}\right) \\ \left(\frac{3}{4}\right)^2 + \left(\frac{3}{4}\right)^2 + \frac{1}{(4)^2} \times f\left(\frac{3}{4}, \frac{3}{4}\right) \end{pmatrix}$$

$$p = \begin{pmatrix} -0.0156 \\ -0.0391 \\ -0.0156 \\ -0.0391 \\ -0.0625 \\ 0.1484 \\ -0.0156 \\ 0.1484 \\ 0.9844 \end{pmatrix}$$

Par vérification, on voit que les résultats sont bons.

5.2 Propositions des méthodes numériques

On a à notre disposition au moins 6 méthodes itératives apprises en cours pour trouver la solution de (P2) en utilisant l'équivalence avec (P3). Elles sont :

- La méthode de Jacobi
- La méthode de Gauss-Seidel
- La méthode de relaxation
- La méthode de gradient à pas optimal
- La méthode de gradient conjugué
- La méthode de SOR (Successive Over Relaxation)

Pour toutes ces méthodes, on implémente la condition d'arrêt telle que :

$$\|u^{(k)} - u^{(k+1)}\|_2 < \textit{tolérance}$$

avec la *tolérance* définie par l'utilisateur. Cette condition occupe moins de temps de calcul que la condition $\|A_N u^{(k)} - b\|_2$ car on évite des multiplications matricielles.

5.2.1 La méthode de Jacobi

La formule qui nous permet de programmer la méthode de Jacobi est la suivante :

$$u_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} u_j^{(k)} - \sum_{j=i+1}^n a_{ij} u_j^{(k)} \right) \quad (49)$$

Algorithm 19 Méthode itérative de Jacobi

```
% méthode de jacobi après avoir eu le td 10
% programmé 090515
function [u,iter]=jcb(A,b,tol,itermax)
[m,n]=size(A);
u=ones(n,1);
u_old=ones(n,1);
error=zeros(n,1);
e=0;
fin=0;
for k=1 :itermax
    if fin==0
        for i=1 :n
            %u(i)=(b(i)-(A(i,1 :i-1)*u(1 :i-1)+A(i,i+1 :n)*u(i+1 :n)))/A(i,i);
            s=0;
            for j=1 :n
                if j~=i
                    s=s+A(i,j)*u(j);
                end
            end
            u(i)=(b(i)-s)/A(i,i);
        end
        for i=1 :n
            error(i)=abs(u_old(i)-u(i));
            u_old(i)=u(i);
        end
        % calcul de la norme 2 de u(n)-u(n-1)
        for i=1 :n
            e=e+error(i)*error(i);
        end
        iter=k;
        if sqrt(e)<tol
            fin=1;
            break;
        else
            e=0;
        end
    end
end
end
```

5.2.2 La méthode de Gauss-Seidel

La formule qui nous permet de programmer la méthode de Gauss-Seidel est la suivante :

$$u_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} u_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} u_j^{(k)} \right) \quad (50)$$

Algorithm 20 Méthode itérative de Gauss-Seidel

```
% méthode de gauss seidel après avoir eu le td 10
% programmé 090515
function [u,iter]=gs(A,b,tol,itermax)
[m,n]=size(A);
u=ones(n,1);
u_old=ones(n,1);
error=zeros(n,1);
e=0;
fin=0;
for k=1 :itermax
    if fin==0
        for i=1 :n
            somme=0;
            for j=1 :i-1
                somme=somme+A(i,j)*u(j);
                u(i)=(b(i)-somme)/A(i,i);
            end
            for j=i+1 :n
                somme=somme+A(i,j)*u(j);
                u(i)=(b(i)-somme)/A(i,i);
            end
            u(i)=(b(i)-somme)/A(i,i);
        end
        for i=1 :n
            error(i)=abs(u_old(i)-u(i));
            u_old(i)=u(i);
        end
        % calcul de la norme 2 de u(n)-u(n-1)
        for i=1 :n
            e=e+error(i)*error(i);
        end
        iter=k;
        if sqrt(e)<tol
            fin=1;
            break;
        else
            e=0;
        end
    end
end
end
```

5.2.3 La méthode de relaxation

Avec cette méthode, la procédure de calcul pour une itération est la suivante :

$$\begin{cases} d^{(k)} = e_k & \text{avec } e_k \text{ un vecteur canonique avec un 1 au } k\text{ième ligne} \\ r^{(k)} = Au^{(k)} - b \\ p^{(k)} = \frac{(r^{(k)}, d^{(k)})}{(Ad^{(k)}, d^{(k)})} \\ u^{(k+1)} = u^{(k)} - p^{(k)}d^{(k)} \end{cases}$$

On s'aperçoit qu'avec cette méthode, si la matrice A est de taille N , c'est après N itérations qu'on trouve le résultat de la première itération de la méthode de Gauss-Seidel.

Algorithm 21 Méthode itérative de relaxation

```
% méthode de relaxation
% programmé 090515
function [u,iter]=rlx(A,b,tol,itermax)
[m,n]=size(A);
u=ones(n,1);
u_old=ones(n,1);
I=eye(m,n);
error=zeros(n,1);
e=0;
fin=0;
for k=1 :itermax
    if fin==0
        for i=1 :n
            d=I(:,i);
            u_old=u;
            r=A*u-b;
            Ad=A*d;
            dTr=0;
            dTAd=0;
            for j=1 :n
                dTr=dTr+r(j)*d(j);
                dTAd=dTAd+Ad(j)*d(j);
            end
            p=dTr/dTAd;
            u=u_old-p*d;
        end

        for i=1 :n
            error(i)=abs(u_old(i)-u(i));
            u_old(i)=u(i);
        end
        % calcul de la norme 2 de u(n)-u(n-1)
        for i=1 :n
            e=e+error(i)*error(i);
        end
        iter=k*n;
        if sqrt(e)<tol
            fin=1;
            break;
        else
            e=0;
        end
    end
end
end
```

5.2.4 La méthode du gradient à pas optimal

Avec cette méthode, la procédure de calcul pour une itération est la suivante :

$$\begin{cases} r^{(k)} = Au^{(k)} - b \\ p^{(k)} = \frac{(r^{(k)}, r^{(k)})}{(Ar^{(k)}, r^{(k)})} \\ u^{(k+1)} = u^{(k)} - p^{(k)}r^{(k)} \end{cases}$$

Algorithm 22 Méthode itérative de gradient à pas optimal

```
% méthode de gradient à pas optimal
% programmé 090515
function [u,iter]=gpo(A,b,tol,itermax)
[m,n]=size(A);
u=ones(n,1);
u_old=ones(n,1);
error=zeros(n,1);
e=0;
fin=0;
for k=1 :itermax
    if fin==0
        r=A*u-b;
        Ar=A*r;
        u_old=u;
        rTr=0;
        rTAr=0;
        for j=1 :n
            rTr=rTr+r(j)*r(j);
            rTAr=rTAr+Ar(j)*r(j);
        end
        p=rTr/rTAr;
        u=u-p*r;

        for i=1 :n
            error(i)=abs(u_old(i)-u(i));
            u_old(i)=u(i);
        end
        % calcul de la norme 2 de u(n)-u(n-1)
        for i=1 :n
            e=e+error(i)*error(i);
        end
        iter=k;
        if sqrt(e)<tol
            fin=1;
            break;
        else
            e=0;
        end
    end
end
end
```

5.2.5 La méthode du gradient conjugué

Avec cette méthode, on fixe :

$$\begin{cases} r^{(0)} = Au^{(0)} - b \\ d^{(0)} = r^{(0)} \end{cases}$$

Puis, la procédure de calcul pour une itération est la suivante :

$$\begin{cases} p^{(k)} = \frac{(r^{(k)}, d^{(k)})}{(Ad^{(k)}, d^{(k)})} \\ u^{(k+1)} = u^{(k)} - p^{(k)}d^{(k)} \\ r^{(k+1)} = Au^{(k+1)} - b \\ \beta^{(k)} = \frac{(r^{(k+1)}, r^{(k+1)})}{(r^{(k)}, r^{(k)})} \\ d^{(k+1)} = \beta^{(k)}d^{(k)} + r^{(k+1)} \end{cases}$$

Algorithm 23 Méthode itérative de gradient conjugué

```
% méthode de gradient conjugué
% programmé 090515
function [u,iter]=gconj(A,b,tol,itermax)
[m,n]=size(A);
u=ones(n,1);
u_old=ones(n,1);
error=zeros(n,1);
e=0;
fin=0;
% calcul de r et de d
r=A*u-b;
d=r;
% gradient conjugué
for k=1 :itermax
    if fin==0
        Ad=A*d;
        dTr=0;
        dTAd=0;
        for j=1 :n
            dTr=dTr+r(j)*d(j);
            dTAd=dTAd+Ad(j)*d(j);
        end
        p=dTr/dTAd;
        u_old=u;
        u=u_old-p*d;
        r_old=r;
        r=A*u-b;
        rTr=0;
        r_oldTr_old=0;
        for j=1 :n
            rTr=rTr+r(j)*r(j);
            r_oldTr_old=r_oldTr_old+r_old(j)*r_old(j);
        end
        d=rTr/r_oldTr_old*d+r;
        for i=1 :n
            error(i)=abs(u_old(i)-u(i));
            u_old(i)=u(i);
        end
        % calcul de la norme 2 de u(n)-u(n-1)
        for i=1 :n
            e=e+error(i)*error(i);
        end
        iter=k+1;
        if sqrt(e)<tol
            fin=1;
            break;
        else
            e=0;
        end
    end
end
end
```

5.2.6 La méthode de SOR

Cette méthode ressemble à celle de Gauss-Seidel. On ajoute un paramètre ω qui permet à la méthode de converger plus vite à la solution. D'après le cours, on a $0 < \omega < 2$. La formule qui nous permet de programmer cette méthode est la suivante :

$$u_i^{(k+1)} = (1 - \omega)u_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij}u_j^{(k+1)} - \sum_{j=i+1}^n a_{ij}u_j^{(k)} \right) \quad (51)$$

Algorithm 24 Méthode itérative de SOR

```

% méthode de sor
% programmé 090515
function [u,iter]=sor(A,b,tol,itermax,w)
[m,n]=size(A);
u=ones(n,1);
u_old=ones(n,1);
error=zeros(n,1);
e=0;
fin=0;
for k=1 :itermax
    if fin==0
        for i=1 :n
            somme=0;
            for j=1 :i-1
                somme=somme+A(i,j)*u(j);
                %u(i)=(b(i)-somme)/A(i,i);
            end
            for j=i+1 :n
                somme=somme+A(i,j)*u(j);
                %u(i)=(b(i)-somme)/A(i,i);
            end
            %u(i)=(b(i)-somme)/A(i,i);
            u(i)=w*(b(i)-somme)/A(i,i)+(1-w)*u(i);
        end
        for i=1 :n
            error(i)=abs(u_old(i)-u(i));
            u_old(i)=u(i);
        end
        % calcul de la norme 2 de u(n)-u(n-1)
        for i=1 :n
            e=e+error(i)*error(i);
        end
        iter=k;
        if sqrt(e)<tol
            fin=1;
            break;
        else
            e=0;
        end
    end
end
end

```

L'inconvénient de cette méthode est qu'il faut trouver le bon paramètre ω "à la main" afin d'en optimiser le temps de calcul.

Trouver le bon paramètre ω

Voici un programme qui permet de trouver le bon paramètre ω visuellement.

Algorithm 25 Programme qui permet de visualiser le nombre d'itérations en fonction de ω

```
% test sor avec w varie entre 0.1 et 1.9
clear;
n=3;
A=construire_A_N(n);
%b=2*2*ones(n*n,1);
b=construire_p(n);
w=0.1 :0.1 :1.9;
[i,N]=size(w);
iter=zeros(N,1);
for j=1 :N
    [u,iter(j)]=sor(A,b,0.0001,150,w(j));
end
plot(w,iter);
legend('SOR');
xlabel('w');
ylabel('Nombre d'itérations');
title('Méthode SOR');
```

Résultats

Voici les résultats après avoir exécuté le programme ci-dessus plusieurs fois pour des n différentes.

n	$N = n \times n$	ω optimal
5	25	1.4
10	100	1.6
15	225	1.7
20	400	1.8

Ce sont des résultats approximatifs.

5.3 Étude de l'efficacité entre les méthodes

Après avoir programmé toutes ces méthodes, il ne reste qu'à analyser leur efficacité en terme de nombre d'itérations et de temps de calcul pour de différentes valeurs de n avant de les implémenter pour résoudre le problème ($P3$).

5.3.1 Programme pour évaluer l'efficacité des méthodes

Voici le programme sous Matlab :

Algorithm 26 Programme d'évaluation des méthodes

```
% programme qui mesure le temps de calcul et le nombre d'itérations et
% teste la convergence
% codé le 090516
clear;
nmax=10;
% déclarer et initialiser les variables pour stocker le temps écoulé pour
% chaque méthode
t_jcb=zeros(1,nmax);
t_gs=zeros(1,nmax);
t_rlx=zeros(1,nmax);
t_gpo=zeros(1,nmax);
t_gconj=zeros(1,nmax);
t_sor=zeros(1,nmax);
t_ref=[1 :1 :nmax];
% déclarer et initialiser les variables pour stocker le nb d'itérations pour
% chaque méthode
iter_jcb=zeros(1,nmax);
iter_gs=zeros(1,nmax);
iter_rlx=zeros(1,nmax);
iter_gpo=zeros(1,nmax);
iter_gconj=zeros(1,nmax);
iter_sor=zeros(1,nmax);
iter_ref=[1 :1 :nmax];
% test pour la méthode de Jacobi
for i=2 :nmax
    A=construire_A_N(i);
    p=construire_p(i);
    tic;
    [X,iter_jcb(i)]=jcb(A,p,0.00001,10000);
    t_jcb(i)=toc;
    if (norm(A*X-p)>0.001)
        error('Pas de convergence');
    end
end
% test pour la méthode de Gauss-Seidel
for i=2 :nmax
    A=construire_A_N(i);
    p=construire_p(i);
    tic;
    [X,iter_gs(i)]=gs(A,p,0.00001,10000);
    t_gs(i)=toc;
    if (norm(A*X-p)>0.001)
        error('Pas de convergence');
    end
end
% test pour la méthode de relaxation
for i=2 :nmax
    A=construire_A_N(i);
    p=construire_p(i);
    tic;
    [X,iter_rlx(i)]=rlx(A,p,0.00001,10000);
    t_rlx(i)=toc;
    if (norm(A*X-p)>0.01)
        error('Pas de convergence');
    end
end
end
```

Algorithm 27 (Suite)

```
% test pour la méthode de gradient à pas optimal
for i=2 :nmax
    A=construire_A_N(i);
    p=construire_p(i);
    tic;
    [X,iter_gpo(i)]=gpo(A,p,0.00001,10000);
    t_gpo(i)=toc;
    if (norm(A*X-p)>0.001)
        error('Pas de convergence');
    end
end
% test pour la méthode de Gradient conjugué
for i=2 :nmax
    A=construire_A_N(i);
    p=construire_p(i);
    tic;
    [X,iter_gconj(i)]=gconj(A,p,0.00001,10000);
    t_gconj(i)=toc;
    if (norm(A*X-p)>0.001)
        error('Pas de convergence');
    end
end
% test pour la méthode de SOR
for i=2 :nmax
    A=construire_A_N(i);
    p=construire_p(i);
    if i<5
        w=1.4;
    elseif i<10
        w=1.6;
    elseif i<15
        w=1.7;
    else
        w=1.8;
    end
    tic;
    [X,iter_sor(i)]=sor(A,p,0.00001,10000,w);
    t_sor(i)=toc;
    if (norm(A*X-p)>0.001)
        error('Pas de convergence');
    end
end
figure(1);
plot(t_ref,t_jcb,'r-',t_ref,t_gs,'b+',t_ref,t_rlx,'g :',t_ref,t_gpo,'cx',t_ref,t_gconj,'m*',t_ref,t_sor,'k');
legend('Jacobi','Gauss-Seidel','Relaxation','GPO','Gradient Conjugué','SOR');
xlabel('n');
ylabel('Temps en seconde');
title('Temps de calcul');
figure(2);
plot(iter_ref,iter_jcb,'r-',iter_ref,iter_gs,'b+',iter_ref,iter_rlx,'g :',iter_ref,iter_gpo,'cx',iter_ref,iter_gconj,'m*',iter_ref,iter_sor,'k');
legend('Jacobi','Gauss-Seidel','Relaxation','GPO','Gradient Conjugué','SOR');
xlabel('n');
ylabel('Nombre d'itérations');
title('Nombre d'itérations');
```

Résultats

Voici les résultats.

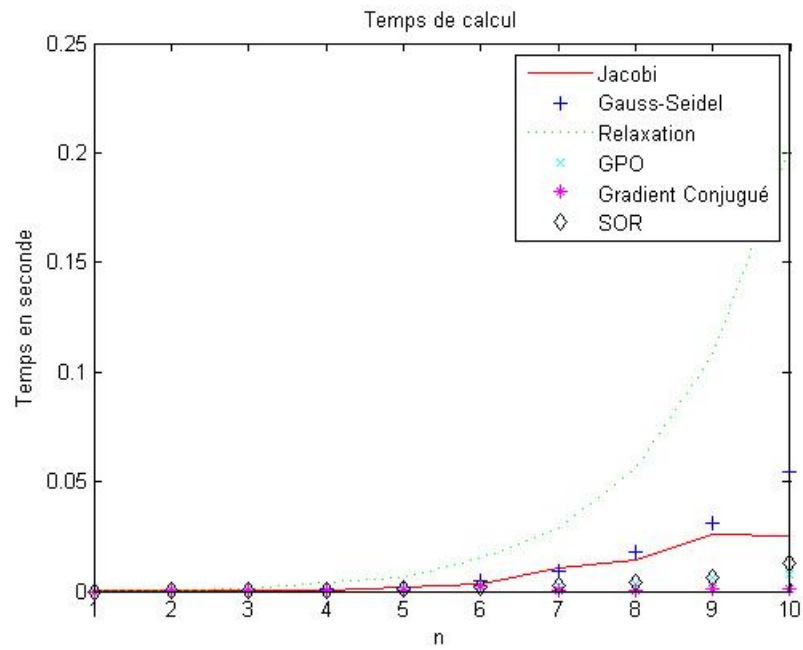


FIGURE 10 – Le temps de calcul pour les différentes méthodes

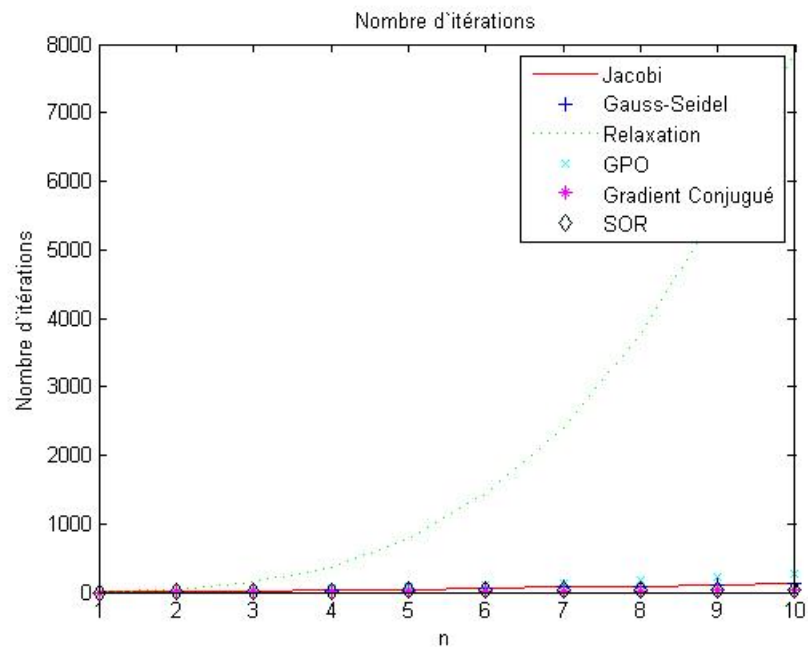


FIGURE 11 – Le nombre total d'itérations pour les différentes méthodes

Observations

On voit que la méthode de relaxation est la moins efficace en terme de temps de calcul et de nombre d'itérations parmi toutes les méthodes proposées. Pour pouvoir analyser plus en détail les autres méthodes, on exclut la méthode

de relaxation et ré-exécute le programme d'évaluation.

Voici les résultats sans prendre en compte la méthode de relaxation :

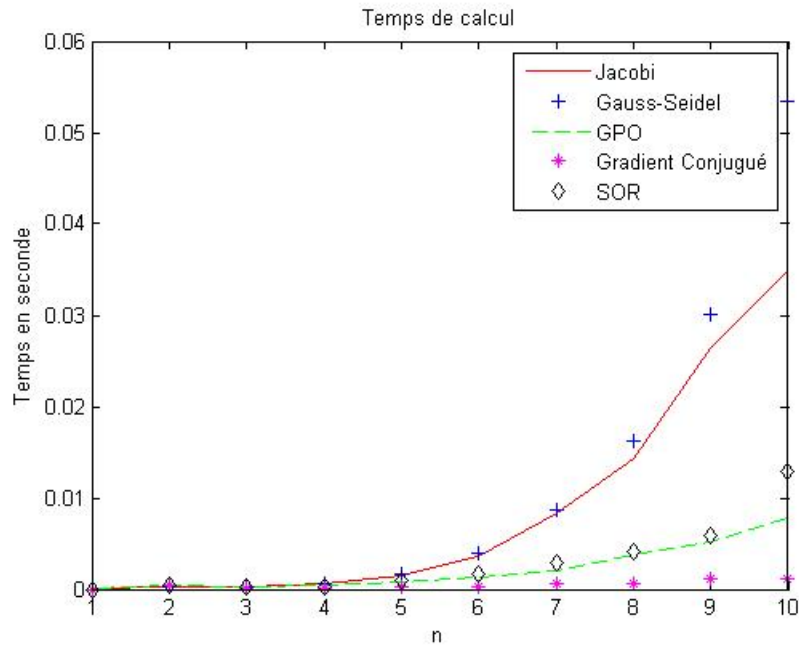


FIGURE 12 – Le temps de calcul

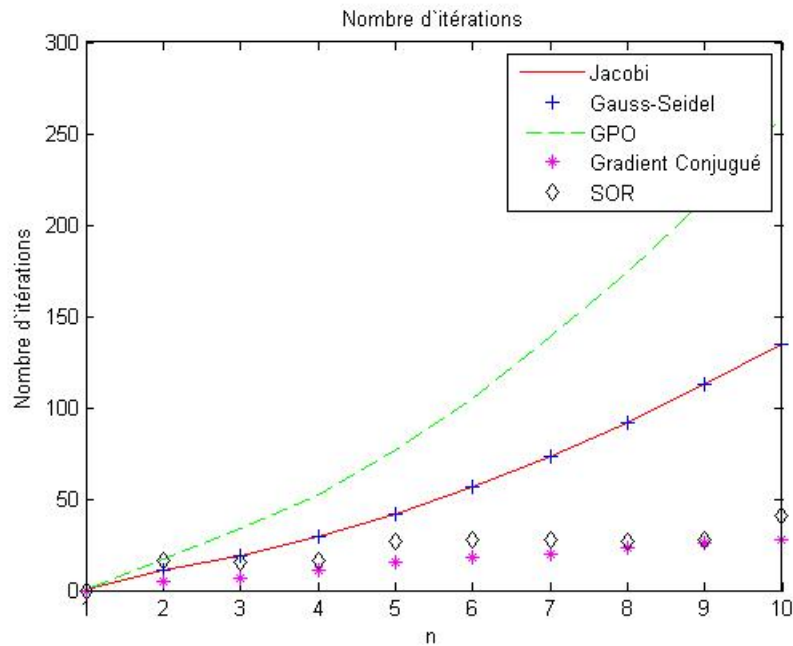


FIGURE 13 – Le nombre d'itérations

D'après ces résultats, on conclut que :

Méthodes	Efficacité en ordre (temps de calcul)	Efficacité en ordre (nombre d'itérations)
Jacobi	4 (0.0348s pour $n = 10$)	3 (135 itérations pour $n = 10$)
Gauss-Seidel	5 (0.0535s pour $n = 10$)	3 (135 itérations pour $n = 10$)
Relaxation	6 (0.2044s pour $n = 10$)	5 (7900 itérations pour $n = 10$)
Gradient à pas optimal	2 (0.0086s pour $n = 10$)	4 (258 itérations pour $n = 10$)
Gradient conjugué	1 (0.0010s pour $n = 10$)	1 (28 itérations pour $n = 10$)
SOR	3 (0.0124s pour $n = 10$)	2 (41 itérations pour $n = 10$)

La méthode la plus efficace en terme de temps de calcul et de nombre d'itérations est celle du gradient conjugué. On décide donc de l'utiliser pour résoudre notre problème à n grand.

5.4 Résolution du problème ($P3$) avec la méthode de gradient conjugué

On peut donc enfin résoudre notre problème en utilisant la méthode de gradient conjugué qui apparaît comme la plus efficace.

5.4.1 Programme de résolution

On essaie de trouver une solution pour $u(x, y)$ avec $n = 50$ (ou $N = 2500$). Voici le programme pour trouver les valeurs approchées de $u(x, y)$.

Algorithm 28 Programme de résolution

```
% programme pour résoudre le problème (P3)
% programmé 090516
clear ;
n=50 ;
tol=0.00001 ;
A=construire_A_N(n) ;
p=construire_p(n) ;
tic ;
[u,iter]=gconj(A,p,tol/10,1000) ;
t=toc ;
% test de convergence
if norm(A*u-p)>tol
    error('Pas de convergence') ;
end
M=zeros(n,n) ;
for i=1 :n
    for j=1 :n
        M(i,j)=u(n*(i-1)+j) ;
    end
end
x=1/(n+1) :1/(n+1) :n/(n+1) ;
y=1/(n+1) :1/(n+1) :n/(n+1) ;
surf(x,y,M) ;
xlabel('x') ;
ylabel('y') ;
title('La solution approchée de u') ;
t
iter
```

Résultats

A la sortie de ce programme, on obtient :

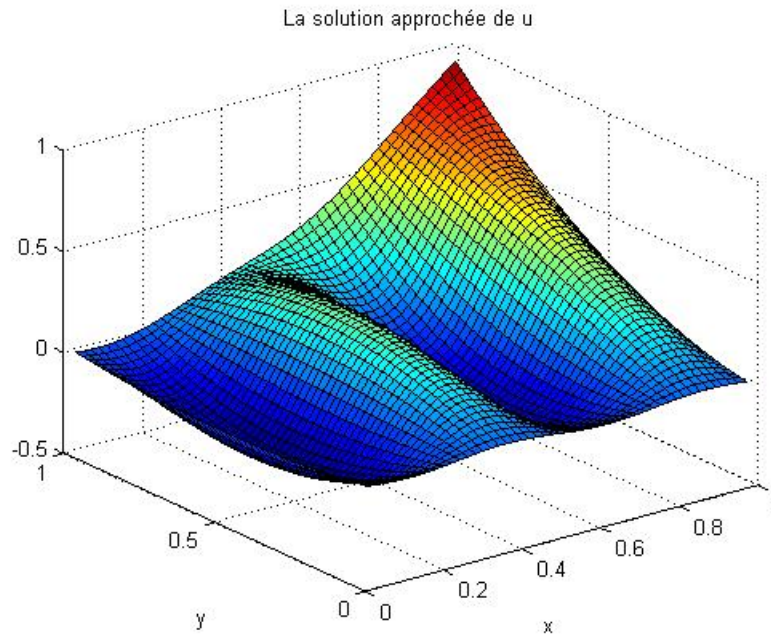


FIGURE 14 – Solution approchée de $u(x, y)$

Observations

Ce programme met environ 4 secondes pour calculer la solution de $u(x, y)$ si $n = 50$ avec 143 itérations en total. On peut dire que cette méthode est assez efficace.

Par contre, cette méthode (gradient conjugué) n'est toujours pas suffisante pour résoudre toute sorte de problème de ce genre, par exemple quand on a une matrice de taille très grande, à l'ordre de millions. Elle va prendre des minutes même des heures pour calculer la solution approchée car on vient de voir tout au long de ce projet que le temps de calcul augmente exponentiellement avec la taille de la matrice.

La recherche pour une meilleure méthode vaut toujours le coup. Or, dû à la contrainte de temps, on est obligé de s'arrêter.

6 Conclusion

Ce projet a été très enrichissant. Il demande beaucoup de travail à notre part pour la recherche des solutions et la rédaction du rapport. Malgré les difficultés, on arrive à finir ce projet à temps.

Étant de futurs ingénieurs, ces connaissances de résolution numérique sont très importantes pour nous. En faisant ce projet, on a l'occasion de mettre en application le cours théorique à un problème pratique et concret. Ce qui nous permet de mieux apprécier l'utilité des méthodes qui nous ont été enseignées.

Enfin, on remarque qu'il existe toujours une meilleure solution ou méthode pour résoudre un problème si on se donne le temps de l'étudier et de l'analyser.