République Algérienne Démocratique et Populaire
Ministère de l'enseignement Supérieur et de la Recherche Scientifique
Université Mohammed Seddik Ben Yahia - Jijel

Faculté des Sciences Exacte et Informatique
Département de Mathématique

№ d'ordre : .....

№ de séries : .....

Mémoire de fin d'études

Présenté pour l'obtention du diplôme de

Master

Spécialité : Mathématiques.

Option : EDP et applications.

Thème

Amélioration des performances de l'algorithme de Karmarkar

Présenté par

Belhoula Assia.

Bouhouche Amina.

Devant le jury :

Président	: Khellaf. W	M.C.B Université de Jijel
Encadreur	: Menniche. L	M.C.B Université de Jijel
Examinateur	: Ariche. S	M.C.B Université de Jijel

Promotion **2017/2018**

Remerciement

En premier lieu, nous tenons à remercier **Allah**, notre créateur pour nous avoir donner la force pour accomplir ce travail.

Nous exprimons toutes nos profondes reconnaissances à nos encadreurs.

Madame **L.Menniche** non seulement pour l'honneur d'avoir accepté de diriger ce travail, mais aussi pour sa patience, ses conseils judicieux et sa compréhension.

Un grand remerciement Aux membres du jury **W. Khellaf et S. Ariche** . Pour leur entière disposition, qui nous ont fait l'honneur d'examiner ce travail et de le juger.

Nous remercions également tous ce qui nous ont aide d'une manière ou d'une autre, de par leurs conseils judicieux ou par la documentation ou les données qu'il ont mis à notre disposition.

Nous ne pourrons pas oublier nos chers parents ,sœurs et frères, tous nos amis, tous nos collègues et tous les membres de nos famille.

Dédicace

L'occasion tant attendue d'offrir et de dédier ce modeste travail, qui clôture toute une période de persévérance continue et assidue, aux personnes qui me sont les plus chères en ce monde. Personnes qui par leur présence permanente ou leur soutien en des moments difficiles ont été le catalyseur de toute mon énergie et tous mes efforts.Je dédie ce travail à

A mon très chère père Ali

Mon père, le plus généreux et le plus compréhensif de tous les pères.

A ma très chère mère Djamila

Ma mère, la plus forte et la plus courageuse de toutes les mères.

A mes chères frères

Ishak et Amine et abdesselem et Atef

A mes chères sœurs

khtete, Karima, Mimi, et sihem

A tous mes amis et mes collègues surtout

Amina ,Amel et Karima ma belle sœur

A toute qui m'aiment et que j'aime

Assia

Dédicace

Tout d'abord, louange **Allah** qui ma guide sur le droit chemin tout au longue de travail et m'a inspiré le bons pas et les justes réflexes.

Au terme de ce travail je tien a exprimer tout ma reconnaissance et remerciement

A mes très chèrs grand père Saïd et père Abderrahmane , mon oncle Rachid

Mon père et mon grand père, mon oncle, les plus généreux et les plus compréhensifs de tous les pères.

A mes très chères mères Aljia et Nassira

Mes mères, qui a œuvré pour ma réussite, l'expression de mes sentiments et de mon éternelle gratitude.

A mon chère mari

Houssam

A mon chère frère

Imad Abdelhamide

A mes chères sœurs

Dounia, Nadia

A mes chères amies

Amel, Mounira, Kaouther, Zinab, Nawal, Hadjer, Amel

A toute qui m'aiment et que j'aime surtout mes cousins

Amina

Table des matières

Introduction **i**

1 Préliminaires et notions fondamentales **1**

1.1 Notions d'algèbre linéaire 1

1.2 Notions de convexité 2

1.3 Programmation mathématique 3

1.3.1 Classification du programme mathématique 4

1.3.2 Qualification des contraintes 5

1.3.3 Conditions nécessaires d'optimalité 5

1.3.4 Fonction de Lagrange et dualité lagrangienne classique 6

1.4 Programmation linéaire 7

1.4.1 Forme usuelles d'un programme linéaire 9

1.4.2 Dualité en programmation linéaire 10

2 Présentation de l'algorithme de Karmarkar **14**

2.1 Méthode de Karmarkar 14

2.1.1 Problème linéaire (P0) 14

2.1.2 Résolution de (P0) par Karmarkar 15

2.1.3 Algorithme de Karmarkar 20

2.2 Etude de la convergence 21

2.3 Généralisation de l'algorithme de Karmarkar 23

3 Amélioration des performances de l'algorithme de Karmarkar 26

3.1 Détermination du pas de déplacement 26

3.2 Calcul de la projection 34

3.2.1 Méthode de Choleskey 35

3.2.2 Méthode du gradient conjugué 36

3.3 Exemples numériques 37

Conclusion 44

Bibliographie 45

Introduction

La programmation mathématique constitue un domaine vaste et riche dans le domaine d'optimisation et dans l'analyse numérique et elle traduit plusieurs problèmes pratiques importants.

Après la deuxième guerre mondiale, la méthode du simplexe s'est imposée comme la seule méthode efficace pour la résolution des programmes linéaires. L'apparition de la méthode de Karmarkar en 1984 a permit le développement des méthodes de points intérieurs qui se sont révélées comme une véritable alternative à la méthode du simplexe.

Le principal obstacle pour construire une itération est la détermination du pas de déplacement et le calcul de direction de déplacement.

Concernant la détermination du pas de déplacement, les études sont concluantes une analyse complète des résultats est présentée dans [7, 9, 10].

Dans l'algorithme de Karmarkar, le coût d'une itération est nettement dominé par le calcul de la direction de déplacement qui se ramène à la résolution d'un système d'équations linéaires.

L'amélioration des performances de l'algorithme de Karmarkar dépend de la technique utilisée pour résoudre ce système.

Ce dernier point fait l'objet principal de notre étude.

Ce mémoire est composé de trois chapitres organisés comme suit :

Le premier chapitre contient une présentation générale de quelques notions de base nécessaires par la suite telle que l'analyse convexe, la programmation mathématique, les conditions d'optimalité et la programmation linéaire.

Dans le deuxième chapitre, nous présentons la méthode de Karmarkar dans son contexte général.

Le troisième chapitre contient l'étude détaillée des difficultés d'ordre pratique de la méthode, la présentation et l'analyse des résultats obtenus lors de l'implémentation effective de la variante de Ye-Lustig.

Chapitre 1

Préliminaires et notions fondamentales

Dans ce chapitre, nous rappelons brièvement certaines définitions d'algèbre linéaire, d'analyse convexe et quelques notions fondamentales de la programmation mathématique et la programmation linéaire.

1.1 Notions d'algèbre linéaire

Définition 1.1. *Soit $A = (a_{ij})_{\substack{1\leq i\leq m\\ 1\leq j\leq n}}$ une matrice à m lignes et n colonnes. La matrice $B = (b_{ji})_{\substack{1\leq i\leq m\\ 1\leq j\leq n}}$ à n ligne et m colonnes telle que $b_{ji} = a_{ij}$, $i = \overline{1,m}$, $j = \overline{1,n}$ est appelée la transposée de A et notée par A^t.*

Définition 1.2. *(Matrice symétrique)*
Une matrice A carrée est dite symétrique si elle égale à sa transposée, i.e., $A = A^t$.

Définition 1.3. *(Matrice définie positive)*
Une matrice symétrique A dont les éléments sont des réels, est définie positive si pour tout vecteur $x \in \mathbb{R}^n$ non nul on a $x^tAx > 0$.

Théorème 1.4. *Deux vecteurs $\vec{u}$ et $\vec{v}$ sont colinéaires si et seulement s'il existe $k \in \mathbb{R}$ tel que $\vec{u} = k\vec{v}$.*

Définition 1.5. *(Norme matricielle)*
Une norme matricielle sur $\mathcal{M}_n(\mathbb{R})$ *est une application* $\|.\| : \mathcal{M}_n(\mathbb{R}) \longrightarrow \mathbb{R}^+$ *vérifiant*
•$\|A\| = 0 \Leftrightarrow A = 0$,
• $\|\alpha A\| = |\alpha|\|A\|, \forall \alpha \in \mathbb{R}, \forall A \in \mathcal{M}_n(\mathbb{R})$,
• $\|A + B\| \leq \|A\| + \|B\|, \forall (A, B) \in \mathcal{M}_n(\mathbb{R})^2$*(inégalité triangulaire)*,
•$\|AB\| \leq \|A\|\|B\|, \forall (A, B) \in \mathcal{M}_n(\mathbb{R})^2$.

Définition 1.6. *Soit* $A = (a_{ij})$, $i = \overline{1, n}$, $j = \overline{1, n}$,
on définie p-norme de la matrice A par
•$\|A\|_1 = \max\limits_{j=1}^{n} \sum\limits_{i=1}^{n} |a_{ij}|$
• $\|A\|_\infty = \max\limits_{i=1}^{n} \sum\limits_{j=1}^{n} |a_{ij}|$
• $\|A\|_2 = (\rho(A^t A))^{\frac{1}{2}}$,
où $\rho(A^t A) = \max\limits_{i=1}^{n} |\lambda_i|$, *où tels que* λ_i, $i = 1, ..., n$ *sont les valeurs propres de la matrice* $A^t A$.

Définition 1.7. *(Conditionnement d'une matrice)*
Le conditionnement d'une matrice inversible A, par rapport à une norme matricielle subordonnée $\|.\|$ *est défini par le nombre*

$$K(A) = \|A\| \cdot \|A^{-1}\|.$$

Définition 1.8.
On dit qu'une matrice est bien conditionnée(respectivement mal conditionnée), si sont conditionnement n'est pas plus grand que un (respectivement très grand que un c-à-d $K(A) >> 1$*).*

1.2 Notions de convexité

Définition 1.9. *(Ensemble affine)*
Un sous ensemble $X \in \mathbb{R}^n$ *est dit affine si*

$$\forall x, y \in X, \forall \lambda \in \mathbb{R} : \lambda x + (1 - \lambda) y \in X.$$

Définition 1.10. *(Ensemble convexe)*
Un ensemble X de $\mathbb{R}^n$ est dit convexe si est seulement si

$$\forall x, y \in X : \ \lambda x + (1-\lambda)y \in X, \ \forall \lambda \in [0,1]$$

c'est-à-dire, le segment de droite reliant deux points de X est entièrement contenu dans X.

Définition 1.11. *Un ensemble convexe de la forme*

$$X = \{x \in \mathbb{R}^n : Ax = b, \ x \geq 0\}$$

est appelé un polytope.

Définition 1.12. *(Ensemble borné)*
Un ensemble X de $\mathbb{R}^n$ est borné, s'il existe $k \in \mathbb{R}$ tel que la valeur absolue de chaque composante de chaqu'un élément de X est plus petite que k, $\forall x \in X : \exists k \in \mathbb{R}, |x| \leq k$.

Définition 1.13. *Un polytope borné sera appelé polyèdre convexe.*

Définition 1.14. *L'ensemble S_n est un n-simplexe, s'il est de la forme*

$$S_n = \{x \in \mathbb{R}^n_+ : \sum_{i=1}^{n} x_i = 1\}.$$

Définition 1.15. *(Fonction convexe)*
Une fonction f est convexe sur un ensemble convexe X si

$$\forall x, y \in X : \ f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y), \ \forall \lambda \in [0,1].$$

Définition 1.16. *(Combinaison linéaire convexe)*
Un vecteur $y \in X \subset \mathbb{R}^n$ est une combinaison linéaire convexe des points $\{x_1, ..., x_k\}$, s'il existe des coefficients réels $\lambda_i \geq 0$, $i \in \{1, ..., k\}$, tels que

$$y = \sum_{i=1}^{k} \lambda_i x_i \ \textit{avec} \ \sum_{i=1}^{k} \lambda_i = 1.$$

1.3 Programmation mathématique

La programmation mathématique constitue un domaine vaste et riche dans le domaine d'optimisation et dans l'analyse numérique, et elle traduit plusieurs problèmes pratiques importants.

Définition 1.17. *Un programme mathématique (PM) est un problème d'optimisation écrit sous la forme*

$$(PM)\begin{cases} \min f(x) \\ \textit{sous les contraintes} \\ D = \{x \in \Omega \subset \mathbb{R}^n : g_i(x) \leq 0,\ i = \overline{1,n} \ \ et \ \ h_j(x) = 0,\ j = \overline{1,m}\}. \end{cases}$$

Où f, g_i et h_j sont des fonctions définies de $\mathbb{R}^n$ dans $\mathbb{R}$.

- *Un point x de D est appelé solution réalisable de (PM).*
- *D est l'ensemble des solutions réalisables de (PM).*
- *On appelle solution optimale de (PM), toute solution réalisable x^* réalisant le minimum de f sur D, et $f(x^*)$ sera appelée valeur optimale de (PM).*

1.3.1 Classification du programme mathématique

La classification de (PM) et son traitement numérique sont établis à partir des propriétés fondamentales des fonctions f, g_i et h_j à savoir la convexité, la différentiabilité et la linéarité. Parmi les cas particuliers les plus étudiés on note

- La programmation linéaire (f linéaire, g_i, h_j affine, D ortant positif ($\mathbf{R^n_+}$).
- La programmation convexe (f, g_i, convexe, h_j affine, D convexe).
- La programmation en nombres entiers (D discret).

Avant de donner les conditions d'optimalité de (PM), on exige que les contraintes doivent satisfaires certains critères dit de qualification.

Définition 1.18. *(Minimum local)*
Une solution réalisable x^* est un minimum local de (PM) s'il existe $\varepsilon > 0$ tel que

$$f(x^*) \leq f(x), \forall x \in \mathbf{B}(x^*, \varepsilon) \ \textit{où} \ \mathbf{B}(x^*, \varepsilon) = \{x \in \mathbb{R}^n : \parallel x - x^* \parallel \leq \varepsilon\}.$$

Définition 1.19. *(Minimum global)*
Soit $f : S \subset \mathbb{R}^n \to \mathbb{R}$, on dit que la fonction f admet un minimum global (solution optimale global) en $x^* \in S$ si et seulement si

$$f(x) \geq f(x^*),\ \forall x \in S.$$

1.3.2 Qualification des contraintes

Une contrainte g_i est dite active (ou saturée) en $x^* \in D$ si $g_i(x^*) = 0, \forall i = (1, \ldots, n)$. Un point $x^* \in D$ est dit régulier (on dit également que les contraintes sont qualifiées en x^*) si les gradients des contraintes saturées en x^* sont linéairement indépendants.
Il existe aussi deux critères usuels de qualification en tout point de D, à savoir :
. Si toutes les contraintes sont affines.
. Si D est défini uniquement par des inégalités, on a le critère de Slater suivant : si $g_i(\overline{x})$ est convexe pour tout $i = \overline{1,n}$ et qu'il existe un point $x^0 \in D$ tel que $g_i(x^0) < 0$, $(int(D) \neq \emptyset)$.

1.3.3 Conditions nécessaires d'optimalité

Théorème 1.20. ***(Karush-Kuhn-Tucker)***
Soit $x^* \in D$ satisfaisant l'une des conditions de qualification et supposons que f, g_i, h_j sont $C^1(\mathbb{R}^n)$, on a :
Si x^* est un minimum local du problème (PM) alors il existe un vecteur $y \in \mathbb{R}^m$ et $\lambda \in \mathbb{R}^n_+$ tel que

$$\begin{cases} \nabla f(x^*) + \sum_{i=1}^{n} \lambda_i \nabla g_i(x^*) + \sum_{j=1}^{m} y_j \nabla h_j(x^*) = 0, \ (\textit{condition d'optimalité}) \\ \lambda_i g_i(x^*) = 0, \ i = \overline{1,n}, \ (\textit{condition de complimentarité}) \\ h_j(x^*) = 0, \ j = \overline{1,m}. \end{cases}$$

Si de plus f, g_i, h_j sont convexes, les conditions d'optimalité précédentes sont à la fois nécessaires et suffisantes pour que x^* soit un optimum global pour (PM). Les problèmes avec contraintes de type (PM) peuvent souvent êtres transformés en problèmes sans contraintes à l'aide des multiplicateurs de Karush Kuhn Tucker (KKT). Cette méthode qui nous permet de trouver les point stationnaires x^* de la fonction f, revient en effet à associer à chaque contrainte un scalaire inconnu λ_i, y_j. On forme par la suite une combinaison linéaire des gradients de la fonction objectif et des contraintes comme le montre le système ci-dessus.
Dans le cas particulier où $n = 0$, les conditions précédentes sont appelées les conditions de Lagrange et s'écrivent comme suit
Si x^* est un optimum local, alors il existe $\lambda_j \in \mathbb{R}$, $j = \overline{1,m}$ tel que

$$\begin{cases} \nabla f(x^*) + \sum_{j=1}^{m} \lambda_j \nabla h_j(x^*) = 0 \\ h_j(x^*) = 0, \ j = \overline{1,m}. \end{cases}$$

1.3.4 Fonction de Lagrange et dualité lagrangienne classique

On considère le programme de type suivant

$$(P_g) \begin{cases} \min f(x), \\ g_i(x) \leq 0, \ i = \overline{1,m} \\ x \in S \subset \mathbb{R}^n \end{cases}$$

Associons à chaque contrainte un nombre réel $\lambda_i > 0$ appelé multiplicateur de Lagrange.

La fonction de Lagrange (où lagrangien) associée à ce problème est définie par

$$L(x,y) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x), \ \ x \in S, \ \lambda_i > 0.$$

Définition 1.21. *On appelle (x^*, λ^*) un point selle pour la fonction de Lagrange sur $S \times \mathbb{R}^m_+$, si tout point $(x, \lambda) \in S \times \mathbb{R}^m_+$ on a*

$$\min_{x \in S} L(x,\lambda) = L(x^*,\lambda) \leq \min_{x \in S} \max_{\lambda \in \mathbb{R}^m_+} L(x,\lambda) = L(x^*,\lambda^*) \leq \max_{\lambda \in \mathbb{R}^m_+} L(x,\lambda) = L(x,\lambda^*).$$

Rappelons une propriété caractéristique des points selles

(x^*, λ^*) est un point selle si et seulement si

- $L(x^*,\lambda^*) = \min_{x \in S} L(x,\lambda^*)$
- $g_i(x^*) \leq 0, \ i = \overline{1,m}.$
- $\lambda_i^* g_i(x^*) = 0, \ i = \overline{1,m}.$

De plus on a le résultat suivant, si (x^*, λ^*) est un point selle pour la fonction de Lagrange, alors x^* est un minimum global pour (P_g).

Définissons maintenant pour $\lambda > 0$ la fonction q donnée par

$$q(\lambda) = \min_{x \in D} L(x,\lambda)$$

Et le programme

$$(D_g) \begin{cases} \max q(\lambda) = \max_{\lambda \in \mathbb{R}^m_+} \min L(x,\lambda) \end{cases}$$

La fonction q est appelée la fonction duale et (D_g) le programme dual associé au programme primal (P_g).
Revenons à la programmation mathématique linéaire qui sera un outil important pour démontrer ultérieurement un résultat fondamental.

Dualité lagrangienne

Soit

$$D = \{x \in D \subset \mathbb{R}^n : \ g_i(x) \leq 0, \ i = \overline{1,k}, \ h_j(x) = 0, \ j = \overline{1,m}\}$$

et on considère le problème d'optimisation

$$m = \inf_{x \in S} f(x),$$

Le lagrangien associé est la fonction $L : D \times [0, +\infty[^k \times \mathbb{R}^m \longrightarrow \mathbb{R}$ définie par

$$L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{k} \lambda_i g_i(x) + \sum_{j=1}^{m} \mu_j h_j(x).$$

On pose

$$\alpha(x) = \sup_{\lambda,\mu}[L(x, \lambda, \mu) : \ \lambda \geq 0] = \begin{cases} f(x) \ si \ g_i(x) \leq 0 \ et \ h_j(x) = 0 \\ +\infty \ sinon \end{cases}$$

donc

$$\overline{\alpha} = \inf_{x \in D} \alpha(x) = \inf_{x}[f(x), \ x \in S]$$

Le problème dual est

$$\overline{\beta} = \sup_{(\lambda,\mu)} \inf_{x \in D}[L(x, \lambda, \mu)]$$

On a l'inégalité de dualité

$$-\infty \leq \overline{\beta} \leq \overline{\alpha}$$

1.4 Programmation linéaire

La programmation linéaire dans $\mathbb{R}^n$ traite la résolution des problèmes d'optimisation pour lequels la fonction objectif et les contraintes sont linéaires.

Définition 1.22.

Un programme linéaire (PL) est un système d'équations appelées "contraintes" qui sont linéaires et à partir de ses contraintes on doit optimiser une fonction également linéaire appelée "objectif" (économique) sous la forme

$$(PL)\begin{cases} Opt(\sum_{j=1}^{n} c_j x_j) \\ sous\ les\ contraintes \\ \sum_{j=1}^{n} a_{ij}x_j(\leq,=,\geq)b_i,\ i=\overline{1,m}, \\ x_j \geq 0,\ j=\overline{1,n} \end{cases}.$$

(PL) s'écrit aussi sous la forme matricielle

$$(PL)\begin{cases} Opt(c^t x) \\ sous\ les\ contraintes \\ Ax(\leq,=,\geq)b, \\ x \geq 0. \end{cases}$$

Tels que

- $Opt = (max\ ou\ min)$.
- $c^t x = \sum_{j=1}^{n} c_j x_j$ *est la fonction à optimiser.*
- $c \in \mathbb{R}^n$ *est le vecteur coût.*
- $b \in \mathbb{R}^m$ *est le second membre.*
- $A = (a_{ij})_{\substack{1\leq i\leq m \\ 1\leq j\leq n}} \in \mathbb{R}^{m\times n}$ *est la matrice des contraintes.*

Exemple 1.23.

$$(PL)\begin{cases} max(5x_1 + 10x_2) \\ x_1 + x_2 \leq 100, \\ x_1 + x_2 \geq 200, \\ x_1, x_2 \geq 0. \end{cases}$$

Remarque. *Il n'est pas restrictif de supposer $x \geq 0$. En effet, si la variable x n'est pas contraintes en signe on pourra remplacer x par la différence $x = x^+ - x^-$, $x^+ \geq 0$, $x^- \geq 0$. En effet $x^+ = max[0, x]$, $x^- = max[0, -x]$.*

1.4.1 Forme usuelles d'un programme linéaire

Les programmes linéaires (PL) se présentent sous trois formes différentes

Forme standard

Un programme linéaire (PL) est sous la forme standard si toutes les contraintes sont des égalités, c'est-à-dire qu'il s'écrit sous la forme

$$(PL)\begin{cases} Opt(c^t x) \\ sous\ les\ contraintes \\ Ax = b, \\ x \geq 0. \end{cases}$$

Forme canonique

Un programme linéaire (PL) est sous la forme canonique si toutes les contraintes sont des inégalités, c'est-à-dire qu'il s'écrit sous la forme

$$(PL)\begin{cases} Opt(c^t x) \\ sous\ les\ contraintes \\ Ax(\geq, \leq)b, \\ x \geq 0. \end{cases}$$

Forme mixte

On dit que le problème de programmation linéaire (PL) est sous forme mixte, s'il n'écrit pas ni sous forme canonique, ni sous forme standard.

Remarque. *On peut toujours mettre un programme linéaire quelconques sous forme standard en introduisant des variables supplémentaires positives appelées "variables d'écarts".*

Exemple 1.24. *Soit le problème écrit sous la forme canonique*

$$\begin{cases} min(5x_1 - 3x_2) \\ x_1 - x_2 \geq 2, \\ 2x_1 + 3x_2 \leq 4, \\ -x_1 + 6x_2 = 10, \\ x_1, x_2 \geq 0. \end{cases}$$

En introduisant les variables d'écart $x_3 \geq 0$, $x_4 \geq 0$ *dans la première et la deuxième contrainte, on met le problème précédent sous la forme standard suivante*

$$\begin{cases} min\,(5x_1 - 3x_2 + 0x_3 + 0x_4) \\ x_1 - x_2 - x_3 = 2, \\ 2x_1 + 3x_2 + x_4 = 4, \\ -x_1 + 6x_2 = 10, \\ x_1, x_2, x_3, x_4 \geq 0. \end{cases}$$

1.4.2 Dualité en programmation linéaire

La notion de dualité est un concept fondamental en programmation linéaire qui conduit à un résultat de grande portée théorique et pratique (le théorème de dualité faible et le théorème de dualité forte).

Ainsi, étant donné un programme linéaire (P) appelé *primal*, on peut toujours lui associer un autre problème appelé *dual* (noté par (D)).

Soit le problème *primal*

$$(P) \begin{cases} min \;\; c^t x \\ Ax = b, \\ x \geq 0. \end{cases}$$

Son dual s'écrit sous la forme

$$(D) \begin{cases} max \;\; b^t x \\ A^t y \leq c, \\ y \in \mathbb{R}^m. \end{cases}$$

Le problème linéaire (D) est très liés au problème linéaire (P). On remarque que,

- La matrice des contraintes de (D) est la transposée de la matrice des contraintes de (P).
- Le vecteur coût de (P) est le vecteur de second membre des contraintes de (D).
- Le vecteur coût de (D) est le vecteur de second membre des contraintes de (P).

Définition 1.25. *(Définition du dual dans le cas général)*
Evidemment, on peut définir le dual d'un problème linéaire quelconque (pas nécessairement sous forme standard), le tableau suivant résume les correspondances entre le problème primal et son dual et permet d'écrire directement le dual d'un problème linéaire quelconque.

Primal	*Dual*
Fonction objectif (min)	Second membre
Second membre	Fonction objectif (max)
A matrice des contraintes	A^t matrice des contraintes
Contrainte $i \geq$	Variable $y_i \geq 0$
Contrainte $i =$	Variable $(y_i \geq, \leq 0)$
Variable $x_j \geq 0$	Contrainte $j \leq$
Variable $x_j(\geq, \leq)0$	Contrainte $j =$

TABLE 1.1 – Relation *primal- dual*

Exemple 1.26.
Soit le problème primal suivant

$$(P)\begin{cases} min(100x_1 + 600x_2) \\ 2x_1 + 5x_2 = 3, \\ 2x_1 + 2x_2 = 2, \\ 4x_1 + 6x_2 = 1, \\ x_1, x_2 \geq 0. \end{cases}$$

Alors son dual est

$$(D)\begin{cases} max(3y_1 + 2y_2 + y_3) \\ 2y_1 + 2y_2 + 4y_3 \leq 100, \\ 5y_1 + 2y_2 + 6y_3 \leq 600, \\ (y_1, y_2, y_3) \in \mathbb{R}^3. \end{cases}$$

Remarque.

1· *Le dual du problème dual* (D) *est le problème primal* (P).

2· *On peut transformer un problème de maximisation à un problème de minimisation. Il suffit d'écrire* $max f = -min(-f)$.

Remarque.

On suppose dans tout ce qui suit que la matrice des contraintes A *est de plein rang, i.e.,* $rang(A) = m < n$.

Théorème 1.27. *(Dualité faible)*

Si x *et* y *sont des solutions réalisables de problèmes* (P) *et* (D) *respectivement, alors*

$$c^t x > b^t y \ .$$

Démonstration. On a x et y sont réalisables, alors

$$c \geq A^t y \quad d'o\grave{u} \quad c^t \geq (A^t y)^t,$$

ce qui implique

$$c^t x \geq (A^t y)^t x,$$

alors

$$c^t x \geq y^t A x,$$

d'où

$$c^t x \geq y^t b,$$

donc

$$c^t x \geq (y^t b)^t,$$

donc on obtient

$$c^t x \geq b^t y.$$ ■

Théorème 1.28. *(Dualité forte)*

Si x^* *et* y^* *sont des solutions de* (P) *et* (D) *respectivement telles que* $b^t y^* = c^t x^*$, *alors* x^* *et* y^* *sont des solutions optimales des problèmes* (P) *et* (D) *respectivement.*

Démonstration.

On veut démontrer que x^* et y^* sont des solutions optimales des problèmes (P) et (D) respectivement, c'est-à-dire, on démontre que

$$c^t x^* = min\ c^t x \text{ et } b^t y^* = max\ b^t y.$$

D'après le théorème précédent on a $c^t x \geq b^t y$ donc $min\ c^t x \geq max\ b^t y$
en particulier, $c^t x^* \geq min\ c^t x \geq max\ b^t y \geq b^t y^*$
Et comme $b^t y^* = c^t x^*$ alors

$$c^t x^* \geq min\ c^t x \geq max\ b^t y \geq c^t x^* \Longrightarrow c^t x^* \geq min\ c^t x \geq c^t x^*$$

et

$$b^t y^* \geq min\ c^t x \geq max\ b^t y \geq b^t y^* \Longrightarrow b^t y^* \geq max\ b^t y \geq b^t y^*.$$

D'où le résultat. ■

Chapitre 2

Présentation de l'algorithme de Karmarkar

2.1 Méthode de Karmarkar

L'algorithme de Karmarkar (introduit en 1984) pour la résolution d'un programme linéaire est certainement l'un des progrès les plus significatifs dans le domaine de la programmation mathématique.

L'objet de ce chapitre est la présentation de cet algorithme dans son contexte général[7].

2.1.1 Problème linéaire (P0)

On s'intéresse à la résolution du problème suivant

$$(\text{P0})\begin{cases} min\ c^t x = z^* \\ Ax = 0, \\ x \in S_n = \{\ x \in \mathbb{R}^n : e_n^t x = 1;\ x \geq 0\}. \end{cases}$$

Où $c \in \mathbb{R}^n$, est le vecteur constant appelé vecteur coût et $e_n = (1, 1, ..., 1)^t \in \mathbb{R}^n$.

Hypothèses de Karmarkar

1· La valeur optimale est nulle, i.e., si x^* est une solution optimale de (P0) alors, $c^t x^* = z^* = 0$.

2· $x^0 = \frac{1}{n} e_n$ (centre de simplexe S_n) est une solution réalisable de (P0).

3· La matrice A est de plein rang, i.e., $rang(A) = m < n$.

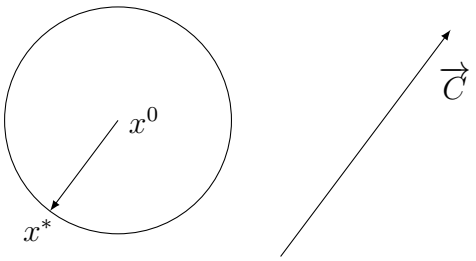
Remarque.

• *Si le système des contraintes $Ax = 0$ n'est pas homogène ou si la valeur optimale est connue mais non nulle, l'équation $e_n^t x = 1$ permet de le ramener à la forme (P0).*

• *On verra par la suite qu'il est toujours possible de se ramener à cette forme pour tout problème linéaire.*

2.1.2 Résolution de (P0) par Karmarkar

Karmarkar exploite le fait que la minimisation d'une fonction linéaire sur une sphère est évidente (il suffit de se déplacer à partir du centre de la sphère dans la direction opposée du vecteur coût d'une distance égale au rayon de la sphère).



La minimisation d'un fonction sur une sphére

Considérons

• N(A)=$\{\ x \in \mathbb{R}^n\ /\ Ax = 0\ \}$ est le noyau de A.

• $\Omega = N(A) \cap S_n$ est l'ensemble des contraintes de (P0).

• $B_r = \{\ x \in \mathbb{R}^n\ /\ e_n^t x = 1;\ \|x - x^0\| \leq r\ \}$ est la plus grande sphère contenue dans S_n.

• $B_R = \{\ x \in \mathbb{R}^n\ /\ e_n^t x = 1;\ \|x - x^0\| \leq R\ \}$est la plus petite sphère contenant S_n, où (on peut vérifier facilement).

$$r = \left(\frac{1}{n(n+1)}\right)^{\frac{1}{2}},\ R = \frac{\sqrt{n+1}}{\sqrt{n}}\ \text{d'où}\ \frac{R}{r} = n+1.$$

Ainsi l'ensemble des contraintes Ω de (P0) est coincé entre deux sphères B_r' et B_R' de $\mathbb{R}^{n-1}$ du centre x^0 et de rayon r et R, respectivement (tracer B_r et B_R sur $N(A)$ respectivement). Car

$$B_r' = \{B_r \cap N(A)\} \subset \Omega \subset \{B_R \cap N(A)\} = B_R'.$$

On a

$$\min_{x\in B'_r} c^t x = z^r \geq \min_{x\in\Omega} c^t x = 0 \geq \min_{x\in B'_R} c^t x = z^R.$$

Notons :

$$z^0 = c^t x^0 \,;\; z^r = c^t x^r \,;\; z^R = c^t x^R.$$

Lemme 2.1. *Le point x^r est une meilleure approximation de x^* que x^0, si elle vérifiée*

$$z^r \leq z^0.$$

Démonstration. x^r et x^R sont colinéaire a cause de la linéarité de l'objectif et l'on a

$$\frac{z^0 - z^R}{z^0 - z^r} = \frac{R}{r} = n - 1$$

donc

$$z^0 - z^r = \frac{z^0 - z^R}{n-1},$$

ou encore

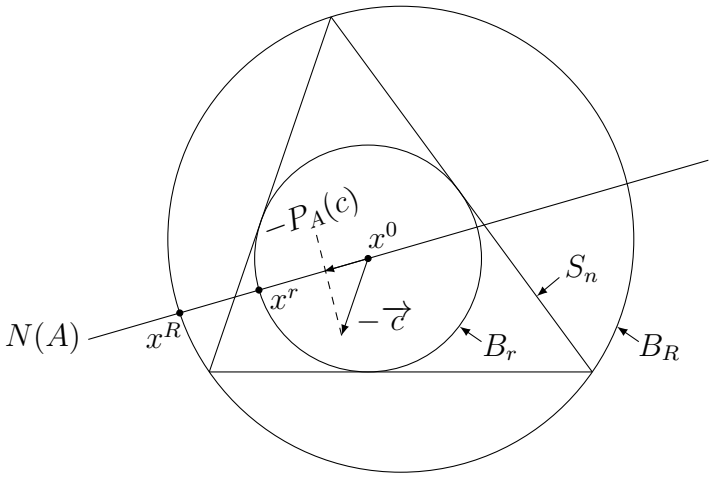
$$\left(1 - \frac{1}{(n-1)}\right) z^0 = z^r - \frac{z^R}{n-1}\,,$$

comme $z^R \leq 0$,

on a

$$z^r \leq \left(1 - \frac{1}{n-1}\right) z^0 < z^0.$$ ■

Ainsi en passant de x^0 à x^r l'objectif est réduit d'un facteur égal à : $1 - \dfrac{1}{n-1}$, et on a la situation suivante



descrption geométréque de la méthode de Karmarkar

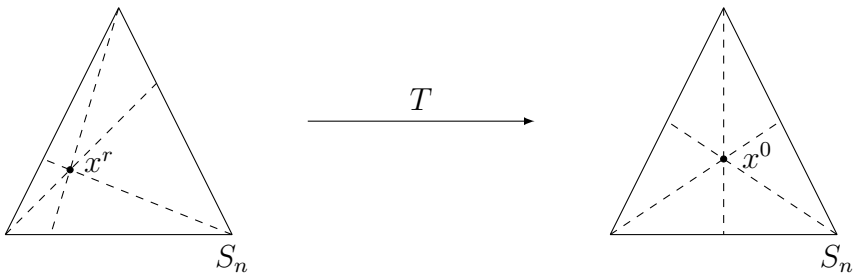
Pour calculer l'itéré suivant x'^r, il est nécessaire de revenir à la forme (P0) avec toutes ses hypothèses.

Pour cela Karmarkar fait appel à sa transformation projective (transformant x^r en x^0), définie par

$$T \ : \ S_n \longrightarrow S_n$$

$$T(x) = y = \frac{D^{-1}x}{e_n^t D^{-1}x} \quad \text{i,e.,} \quad x = T^{-1}(y) = \frac{Dy}{e_n^t Dy} \, ; \, T\,(x^r) = x^0.$$

Où $D = diag(x^r)$, matrice diagonale dont les éléments diagonaux sont les composantes de x^r.
On a donc le schéma suivant



La transformation projectif

Le problème (P0) devient alors

$$(\text{P0T})\begin{cases} min\ c'^t y & \text{où } c' = Dc \\ A'y = 0 & A' = AD, \\ y \in S_n = \{y \in \mathbb{R}^n : e_n^t y = 1\ ;\ y \geq 0\}. \end{cases}$$

Notons par $p = P_B(Dc)$ la projection orthogonale du vecteur Dc sur le noyau de la matrice $B = [AD\ ,\ e_n^t]^t$.

Lemme 2.2. *La projection p est définie par*

$$p = [I - B^t(BB^t)^{-1}B]Dc.$$

Démonstration. On a
$< Dc - p, w >= 0$, pour tout $w \in N(B)$.

Ce qui implique $(Dc - p) \in (N(B))^{\perp} = Im(B^t)$.

D'où $Dc - p = B^t u$ pour tout $u \in \mathbb{R}^m$.

Donc $BDc - Bp = BB^t u$ or $Bp = 0$ car $p \in N(B)$

Ce qui donne $BDc = BB^t u$

La matrice carrée BB^t d'ordre m étant régulière puisque $N(BB^t) = \{0\}$, on peut donc écrire $u = (BB^t)^{-1}BDc$ alors

$Dc - p = B^t(BB^t)^{-1}BDc \Leftrightarrow P = [I - B^t(BB^t)^{-1}B]Dc.$ ■

Lemme 2.3.
La solution optimale (P0T) est donné par $y^k = y^* = x^0 - \alpha r\dfrac{p_k}{\|p_k\|}$
avec $p_k = P_{B_k}(D_k c)$ *la projection de*$(D_k c)$ *sur le noyau de* B_k, *tel que* $B_k = [AD_k, e_n^t]^t$.

Démonstration.
Posons $x = y - x^0$, alors $B_k x = 0$, où $B_k x$ est la matrice des contraintes de (P0T).
(P0T) est alors équivalent au problème suivant

$$(P^*)\begin{cases} min\ c'^t y & \text{où } c' = D_k c \\ A'y = 0 & A' = AD_k, \\ e_n^t y = 1, & \\ \|y - x^0\| \leq \alpha r. & \end{cases}$$

Donc

$$(P^*)\begin{cases} min\ c'^t x \\ B_k x = 0, \\ \|x\|^2 = \sum_{i=1}^{n} x_i^2 \leq (\alpha r)^2. \end{cases}$$

Or x^* est une solution de (P^*) si et seulement si $\exists \lambda \in \mathbb{R}^{m+1}$ et scalaire $\mu \geq 0$ tels que

$$D_k c + B_k^t \lambda + \mu x^* = 0 \qquad (i)$$

En multipliant les deux membres de (i) par B_k on trouve

$B_k D_k c + B_k B_k^t \lambda + \mu B_k x^* = B_k D_k c + B_k B_k^t \lambda = 0$ (puisque $B_k x = 0$)
alors $\lambda = -(B_k B_k^t)^{-1}(B_k D_k c)$
d'où en substituant dans (i),

$$x^* = -\frac{1}{\mu}[I - B_k^t((B_k B_k^t)^{-1})B_k]D_k c = -\frac{1}{\mu}p_k$$

mais x^* vérifie

$$\|x^*\| = \frac{1}{\mu}\|p_k\| = \alpha r$$

donc

$$\frac{1}{\mu} = \frac{\alpha r}{\|p_k\|}$$

d'où

$$x^* = -\alpha r \frac{p_k}{\|p_k\|},$$

finalement

$$y^k = y^* = x^0 - \alpha r \frac{p_k}{\|p_k\|}.$$

■

2.1.3 Algorithme de Karmarkar

Début Algorithme

Initialisation : $x^0 = \frac{e_n}{n}$; ϵ précision donnée ; k=0 ;

Tant que $c^t x^k > \epsilon$ **faire**

pas 0 (Construction de la matrice B_k)

$$\begin{cases} D_k = diag(x^k); \\ B_k = [A_k, e_n^t]^t, \ A_k = AD_k; \end{cases}$$

pas 1 (Calcul de la projection p_k)

$p_k = [I - B_k^t(B_k B_k^t)^{-1} B_k] D_k c$;

pas 2 (Calcul de la direction)

$d_k = \frac{p_k}{\|p_k\|}$;

pas 3 (Calcul de l'itéré y^k dans le problème transformé par T)

$y^k = x^0 - \alpha r d_k$;

avec $r = \frac{1}{\sqrt{n(n+1)}}$, $0 < \alpha < 1$, α est le pas de déplacement ;

pas 4 (Calcul de l'itéré x^{k+1} du problème initial en revenant par la transformation inverse)

$x^{k+1} = \frac{D_k y^k}{e_n^t D_k y^k}$;

$k = k + 1$;

Fin tant que

Fin algorithme.

Remarque.

Du point de vue numérique les deux éléments principaux de cet algorithme sont

i) Le pas de déplacement qui influe significativement sur la vitesse de convergence de l'algorithme, (plus α est grand plus l'algorithme converge vite, α peut être >1).

ii) La direction de déplacement (d_k) dont le coût de calcul est largement dominante. Nous reviendrons sur ces deux points dans le chapitre suivant.

2.2 Etude de la convergence

Pour prouver la convergence du procédé en particulier $c^t x'^r \leq c^t x^r$, Karmarkar introduit la fonction potentiel [2] définie par

$$f(x) = \sum_{i=1}^{n} \left(log \frac{c^t x}{x_i} \right) = nlog(c^t x) - \sum_{i=1}^{n} log(x_i).$$

Où x solution réalisable et prouve

$$f(x'^r) \leq f(x^r) - \delta(\alpha)$$

où

$$\delta(\alpha) = \alpha - \frac{\alpha^2}{2} - \frac{\alpha^2}{1-\alpha},\ 0 < \alpha < 1.$$

Avec α est le pas de déplacement.

Lemme 2.4. *Soit x^k le k-ième itéré de l'algorithme, alors*

$$\frac{c^t x^k}{c^t x^0} \leq (\exp[f(x^k) - f(x^0)])^{\frac{1}{n}},\ où \quad x^0 = \frac{e_n}{n}.$$

Démonstration.

$$\begin{aligned}
exp[f(x^k) - f(x^0)] &= \exp\left[\sum_{i=1}^{n} \log\left(\frac{c^t x^k}{x_i^k} \right) - \sum_{i=1}^{n} \log\left(\frac{c^t x^0}{x_i^0} \right) \right] \\
&= \exp\left[\sum_{i=1}^{n} \log\left(\frac{c^t x^k}{c^t x^0} . \frac{x_i^0}{x_i^k} \right) \right] \\
&= \exp\left[\sum_{i=1}^{n} \left(\log\left(\frac{c^t x^k}{c^t x^0} \right) + \log\left(\frac{x_i^0}{x_i^k} \right) \right) \right] \\
&= \exp\left[n \log\left(\frac{c^t x^k}{c^t x^0} \right) + \sum_{i=1}^{n} \log\left(\frac{x_i^0}{x_i^k} \right) \right] \\
&= \exp\left[\log\left(\frac{c^t x^k}{c^t x^0} \right)^n + \log \prod_{i=1}^{n} \left(\frac{x_i^0}{x_i^k} \right) \right] \\
&= \exp\left[\log\left(\frac{c^t x^k}{c^t x^0} \right)^n \right] \exp\left[\log \prod_{i=1}^{n} \left(\frac{x_i^0}{x_i^k} \right) \right] \\
&= \left(\frac{c^t x^k}{c^t x^0} \right)^n \prod_{i=1}^{n} \left(\frac{x_i^0}{x_i^k} \right).
\end{aligned}$$

D'où

$$\frac{c^t x^k}{c^t x^0} = \left(\prod_{i=1}^{n}\left(\frac{x_i^k}{x_i^0}\right)\right)^{\frac{1}{n}} (exp[f(x^k) - f(x^0)])^{\frac{1}{n}}$$
$$= \left(\prod_{i=1}^{n} n x_i^k\right)^{\frac{1}{n}} (exp[f(x^k) - f(x^0)])^{\frac{1}{n}}.$$

Pour démontrer que $\left(\prod_{i=1}^{n} n x_i^k\right)^{\frac{1}{n}} \leq 1$, on utilise la fonction $f(x^k) = -\log x^k$ telle que $x^k \in S_n$, $x^k \neq 0$. Alors

$$f'(x^k) = -\frac{1}{x^k} \Longrightarrow f''(x^k) = \frac{1}{(x^k)^2} > 0.$$

On a f est convexe, i.e.,

$$f\left(\sum_{i=1}^{n} \lambda_i x_i^k\right) \leq \sum_{i=1}^{n} \lambda_i f(x_i^k)$$

tels que

$$\sum_{i=1}^{n} \lambda_i = 1 \;\; et \;\; 0 < \lambda_i = \frac{1}{n} \;,\; i = 1, .., n \;.$$

Alors

$$-\log\left(\sum_{i=1}^{n} \frac{1}{n} x_i^k\right) \leq -\sum_{i=1}^{n} \frac{1}{n} \log(x_i^k)$$

d'où

$$\log\left(\frac{1}{n}\sum_{i=1}^{n} x_i^k\right) \geq \frac{1}{n}\sum_{i=1}^{n} \log(x_i^k)$$

$$\log\left(\frac{1}{n}\right) \geq \frac{1}{n} \log \prod_{i=1}^{n} (x_i^k)$$

ce qui implique

$$\exp\log\left(\frac{1}{n}\right) \geq \exp\log\left(\prod_{i=1}^{n} x_i^k\right)^{\frac{1}{n}}$$

ce qui donne

$$\left(\prod_{i=1}^{n} x_i^k\right)^{\frac{1}{n}} \leq \frac{1}{n}$$

donc

$$\left(\prod_{i=1}^{n} n x_i^k\right)^{\frac{1}{n}} \leq 1 \ .$$

finalement

$$\frac{c^t x^k}{c^t x^0} \leq (\exp[f(x^k) - f(x^0)])^{\frac{1}{n}} \ .$$

Ce qui complète la preuve. ■

Conséquence : Si $\lim_{k\to+\infty} f(x^k) = -\infty$ alors $\lim_{k\to+\infty} c^t x^k = 0$, autrement dit, pour diminuer $c^t x$, il suffit de réduire suffisamment $f(x)$.

2.3 Généralisation de l'algorithme de Karmarkar

Nous présentons dans ce paragraphe la généralisation de l'algorithme de Karmarkar et ses variantes sous une forme unifiée.
Soit le programme linéaire général écrit sous forme standard

$$(\mathrm{P}) \begin{cases} min\ c^t x = z^* \\ Ax = b;\ b \in \mathbb{R}^m, \\ x \geq 0. \end{cases}$$

La transformation projective de Karmarkar définie par

$T_a : \mathbb{R}^n \longrightarrow S_{n+1}$

$$y = T_a(x) = \begin{cases} y_i = \dfrac{\dfrac{x_i}{a_i}}{1 + \sum_{i=1}^{n} \left(\dfrac{x_i}{a_i}\right)}, \\ y_{n+1} = 1 - \sum_{i=1}^{n} y_i. \end{cases}$$

Nous permet d'associer à (P) le programme linéaire suivant

$$\mathrm{P}(\ z^*\ ) \begin{cases} min\ (D_a c\ ,\ -z^*)^t y \\ A_a y = 0, \\ y \in S_{n+1}. \end{cases}$$

où $A_a = [AD_a\ \ -b]$; $D_a = diag(a)$.

Nous avons donc deux possibilités

a) z^* connue : auquel cas on retrouve la généralisation de l'algorithme de Karmarkar i,e., on obtient P(z^*) vérifiant toutes les hypothèses de Karmarkar, l'algorithme de Karmarkar s'applique immédiatement.

b) z^* inconnue elle est approximée à chaque itération par une valeur z^k qui peut être

• Soit une borne inférieure de z^* (variante de Todd et Burrell [3],[4]) fournie par le dual (D) et le primal (P) est

$$(\mathrm{D}) \begin{cases} max\ b^t y \\ A^t y \leq c, \\ y \in \mathbb{R}^m. \end{cases}$$

D'après le théorème de dualité, on a

$z^k = b^t y^k \leq z^* \leq c^t x^k$, de sorte que x^k et y^k soient solution réalisables de (P) et (D) respectivement (autrement dit, le programme dual (D) constitue une source de bornes inférieures de la valeur optimal z^*). L'algorithme ainsi obtenu génère en plus des solutions primales de (P) et duales de (D), une suite (z^k) d'approximation de z^* telle que

$z^k \leq z^{k+1} \leq z^*$.

Le processus est arrêté dés que $|c^t x^k - z^k| < \epsilon$, où ϵ est la précision voulue.

• Soit une borne supérieure de z^* (variante de Ye-Lustig [6]) donnée par $z^k = c^t x^k \geq z^*$ (par définition).

Le fait que à l'optimum $\|d_k\| \longrightarrow 0$ a incité Lustig à choisir le critère de convergence suivant

$$\frac{\|d_k\|}{|cx^0|} \leq \epsilon, \ (\epsilon \text{ précision donnée}).$$

Remarque.

Il existe une autre alternative pour généraliser l'algorithme de Karmarkar dans le cas z^* est inconnue, il s'agit de la méthode primale-duale [9, 12] qui consiste à combiner (P) avec son dual (D) et chercher simplement une solution réalisable du problème suivant

$$(PD) \begin{cases} Ax = b\ ;\ x \geq 0 \\ A^t y \leq c\ ;\ y\ \in \mathbb{R}^m, \\ c^t x - b^t y = 0. \end{cases}$$

Des transformations très simples permettent de voir que ce problème vérifier toutes les hypothèses de Karmakar, malheureusement la taille du problème augmente énormément

ce qui est indésirable en pratique.

Nous avons retenue à l'algorithme de Ye-Lustig pour deux raisons

a) Sa simplicité algorithmique.

b) Son succès numérique vis-à-vis des autres variantes voir [6], [7].

Algorithme de Ye – Lustig

Début Algorithme

Initialisation k $= 0$; $x^0 > 0$;

Tant que $\dfrac{\|d_k\|}{|c^t x^0|} > \epsilon$ **faire**

pas 1 $A_k = [AD_k \ -b]$; $D_k = diag(x^k)$;

pas 2 $p_k = P_{A_k}[D_k c; -c^t x^k]^t$; $d_k = \dfrac{p_k}{\|p_k\|}$;

pas 3 $y'^{k+1} = \dfrac{e_{n+1}}{n+1} - \alpha r d_k$;

pas 4 $x^{k+1} = T_{a_k}^{-1}(y'^{k+1}) = \dfrac{D_k y'^{k+1}[n]}{y'^{k+1}(n+1)}$;

$k = k + 1$;

Fin tant que

Fin algorithme.

Chapitre 3

Amélioration des performances de l'algorithme de Karmarkar

Comme nous avons signalé dans le chapitre précédent, l'efficacité pratique de l'algorithme de Karmarkar ainsi que toutes ses variantes dépend essentiellement du choix optimal du pas de déplacement α [4, 11] et du calcul (économique) de la direction de déplacement d_k [1].

3.1 Détermination du pas de déplacement

Les études concernant ce point sont concluantes [7, 8]. Nous résumons ici les principaux résultats.

Le pas α peut être choisi de deux manières différentes

a) α **constant**

La valeur de α est fixée une fois pour toute durant l'exécution, les valeurs usuelles sont issues des démonstrations de convergence. Karmarkar suggère $\alpha_1 = 0.25$.

En modifiant uniquement la forme de la fonction potentiel Padeberg [7] propose la valeur $\alpha_2 = 0.79$ ($\alpha_2 > \alpha_1$).

Numériquement plus α est grand plus l'algorithme converge vite. On peut donc prendre α voisin de 1 puisque théoriquement $0 < \alpha < 1$.

Dans cette partie, on présente le programme de la méthode de Karmarkar, ce programme est écrit en Langage Matlab 7, la précision est égale $\epsilon = 10^{-4}$ pour le pas de déplacement α constant.

On donne des exemples pour les différentes valeur de $\alpha \in \{0.25; 0.5; 0.75; 0.99\}$.
Dans chaque exemple, on affiche le nombre d'itérations et le temps de calcul (en second) et la valeur de l'objectif z^*.

Exemple 3.1. *Soit le problème linéaire suivant*

$$(1)\begin{cases} min(x_1 + x_2) \\ 2x_1 + x_2 \leq 2, \\ -2x_1 + 4x_2 \leq 3, \\ x_1, x_2 \geq 0. \end{cases}$$

Le problème (1) s'écrit sous forme standard suivante

$$(2)\begin{cases} min(x_1 + x_2 + 0.x_3 + 0.x_4) \\ 2x_1 + x_2 + x_3 = 2, \\ -2x_1 + 4x_2 + x_4 = 3, \\ x_1, x_2, x_3, x_4 \geq 0. \end{cases}$$

(2) vérifie toutes les hypothèses de Karmarkar.
En utilisant l'algorithme de Karmarkar pour le problème (2).

Programme de la méthode de Karmarkar en Matlab pour l'exemple (3.1)

```
clear all;
clc;
k=0;n=4;m=2;eps=0.0001;alpha=0.99;
r=1/(sqrt(n*(n+1)));
en=[1;1;1;1];
A=[0 -1 -1 -2;-5 1 -3 -2];
C=[1;1;0;0];x0=(1/n)*en;
xk=x0 ;
tic
while ((C'*xk ) > eps)
    Dk=diag(xk);
    Ak=A*Dk;
    B=[(Ak);(en)'];
    a= B*B';
    l=chol(a);
    d=l'*l;
    y=inv(l')*(B*Dk*C);
    u=inv(l)*y;
    Pk=Dk*C-B'*u;
    dk=Pk/norm(Pk);
    yk=x0-alpha*r*dk;
    xk=(Dk*yk)/(en'*Dk*yk);
    o=C'*xk;
    k=k+1;
end
toc
disp(k)
disp(o)
```

On résume les résultats obtenus dans le tableau suivant

α	0.25	0.50	0.75	0.99
nombre d'itérations	666	161	68	35
temps (S)	0.102800	0.035015	0.016353	0.012778
valeur de l'objectif	0.000099	0.00099	0.000088	0.000093

TABLE 3.1 – Tableau des résultats de l'exemple (1) pour α constant.

Exemple 3.2. *Soit le problème linéaire suivant*

$$(3)\begin{cases} min(5x_1 - 3x_2) \\ x_1 - x_2 \geq 2, \\ 2x_1 + 3x_2 \leq 4, \\ -x_1 + 6x_2 = 10, \\ x_1, x_2 \geq 0. \end{cases}$$

Le problème (3) s'écrit sous forme standard suivant

$$(4)\begin{cases} min(5x_1 - 3x_2 + 0.x_3 + 0.x_4) \\ x_1 - x_2 - x_3 = 2, \\ 2x_1 + 3x_2 + x_4 = 4, \\ -x_1 + 6x_2 = 10, \\ x_1, x_2, x_3, x_4 \geq 0. \end{cases}$$

(4) vérifie toutes les hypothèses de Karmarkar.

En utilisant l'algorithme de Karmarkar pour le problème (4).

Programme de la méthode de Karmarkar en Matlab pour l'exemple (3.2)

```
clear all
clc
k=0;n=4;m=2;eps=0.0001;alpha=0.99;
r=1/(sqrt(n*(n+1)));
en=[1;1;1;1];
A=[-1 -3 -3 -2;-2 -1 -4 -3;-11 -4 -10 -10];
C=[5;-3;0;0];
x0=(1/n)*en;xk=x0 ;
tic
while ((C'*xk) > eps)
    Dk=diag(xk);
    Ak=A*Dk;
    B=[(Ak);(en)'];
    a= B'*B ;
   l=chol(a) ;
   d=l'*l ;
    y=inv(l')*(B*Dk*C);
    u=inv(l)*y ;
    Pk=Dk*C-B'*u ;
    dk=Pk/norm(Pk) ;
    yk=x0-alpha*r*dk ;
    xk=(Dk*yk)/(en'*Dk*yk);
   o=C'*xk ;
k=k+1;
end
toc
disp(k)
disp(o)
```

Les résultats obtenus sont donnés dans le tableau suivant

α	0.25	0.50	0.75	0.99
nombre d'itérations	107	61	45	37
temps (S)	0.027401	0.020844	0.018529	0.017561
valeur de l'objectif	0.000096	0.000077	0.000080	0.000091

TABLE 3.2 – Tableau des résultats de l'exemple(2) pour α constant

Remarque.

On ramarque que plus α est grand plus le nombre d'itérations diminu.

Ici la vitesse de convergence est mesurée par le nombre d'itérations.

b) α **variable**

Il s'agit de choisir α de sorte que $y(\alpha) = \frac{e_n}{n} - \alpha r d_k$ soit admissible tout en tenant en compte de la monotonie de la suite $(c^t x^k)$ ce qui revient à prendre le plus grand α possible tel que

- $y(\alpha) > 0$.
- $c^t x^{k+1} < c^t x^k$.

En effet,

Posons $I = \{1....n\}$ et $I^+ = \{i \in I : d_k^i > 0\}$, supposons $d_k \neq 0$ (puisque si d_k est nul, le point x^k est optimal). Alors l'ensemble I^+ est non vide : il suffit de remarque que $e_n^t d_k = 0$.

On a $y(\alpha) > 0 \Longleftrightarrow \left(\frac{1}{n}\right) - \alpha r d_k^i > 0, \forall i \in I$, d'où $\alpha < min\left\{\frac{1}{nrd_k^i}, i \in I^+\right\} = \alpha_{max}$.

Evidemment, tout $\alpha < \alpha_{max}$, garantira la faisabilité mais pas forcément la réduction de l'objectif d'une itération à l'autre. C'est pourquoi nous avons imposé une deuxième condition sur α donnant des résultats meilleurs en pratique.

Il s'agit de trouver une condition sur α garantissant la monotonie $(c^t x^{k+1} < c^t x^k)$.

En prenant toujours $D_k = diag(x^k)$ et en tenant compte de l'équation du simplexe $(e_n^t x^k = 1)$ on peut facilement montrer qu 'on a

$$\begin{aligned} x^{k+1} &= \frac{D_k y(\alpha)}{e n^t D_k y(\alpha)} = \frac{x^k - \alpha n r D_k d_k}{1 - \alpha n r e_n^t D_k d_k} \\ &= \frac{x^k(1 - \alpha n e_n^t D_k d_k) + \alpha n r x^k k e_n^t D_k d_k - \alpha n r D_k d_k}{1 - \alpha n r e_n^t D_k d_k} \\ &= x^k + \alpha(\frac{n r x^k e_n^t D_k d_k - n r D_k d_k}{1 - \alpha n r e_n^t D_k d_k}) \\ &= x^k + \alpha\gamma(n r x^k e_n^t D_k d_k - n r D_k d_k), \end{aligned}$$

d'où

$$c^t x^{k+1} = c^t x^k + \alpha\gamma(\rho c^t x^k - \mu),$$

on a

$$c^t x^{k+1} < c^t x^k$$

alors

$$\alpha\gamma(\rho c^t x^k - \mu) < 0$$

d'où

$$(\mu - \rho c^t x^k) > 0,$$

avec

$$\gamma = \frac{1}{1 - \alpha r e_n^t D_k d_k}$$
$$\mu = nrc^t D_k d_k$$
$$\rho = nre_n^t D_k d_k,$$

Ce qui donne

$$\alpha_k = \begin{cases} \beta\alpha_{max} & si\ (\mu - \rho c^t x^t) > 0, \\ -\beta\alpha_{max} & si\ (\mu - \rho c^t x^t) \leq 0, \\ 0 < \beta < 1. \end{cases}$$

Où

$\alpha_{max} = min\left(\frac{1}{(nrd_k^i)}, i \in I^+\right)$; $I^+ = \{i \in I^+ : d_k^i > 0\}$

Les valeurs α_k ainsi obtenues sont très voisines de celles obtenues précédemment.

Il est donc convenable de retenir le procédé technique qui est moins coûteux en pratique.

Exemple 3.3.

L'exemple 3.1 pour α varié. On a l'algorithme suivant

Programme de la méthode de Karmarkar pour α varié

```
clear all;
clc;
k=0;n=4;m=2;eps=0.0001;beta=0.99;alpham=3;
r=1/(sqrt(n*(n+1)));en=[1;1;1;1];
A=[0 -1 -1 -2;-5 1 -3 -2];C=[1;1;0;0];
x0=(1/n)*en;xk=x0;
tic
while ((C'*xk ) > eps)
    Dk=diag(xk);
    Ak=A*Dk;
    B=[(Ak);(en)'];
    a= B*B';
    l=chol(a);
    d=l'*l;
    y=inv(l')*(B*Dk*C);
    u=inv(l)*y;
    Pk=Dk*C-B'*u;
    dk=Pk/norm(Pk);
    nye=n*r*C'*Dk*dk;
    re=n*r*en'*Dk*dk;
   for i=1:4
    v=(dk(i));
   if dk(i)>0
    alpham1=1/(n*r*v);
   if alpham1 < alpham
    alpham=alpham1;
   end
   end
   end
    alpham;

    if ((nye-re*C'*xk)>0)
      alphak=beta*alpham
    else
      alphak=-beta*alpham
    end
      yk=x0-alphak*r*dk;
      xk=(Dk*yk)/(en'*Dk*yk);
      o=C'*xk;
      k=k+1;
    end
      toc
      disp(k)
```

Les résultats trouvés sont résumés dans le tableau suivant

Méthode	itération	temps (s)	pas α
α variable	8	0.019649	$1.5626 \leq \alpha \leq 1.5890$

TABLE 3.3 – Tableau des résultats de l'exemple 3.1 pour α varié

Remarque.

1) Le pas de déplacement trouvé est supérieur à 1, ce qui favorise l'utilisation du procédé (b).

2) Pour les valeurs de $\alpha > 1$ la convergence de l'algorithme est démontré dans plusieurs articles [8].

3.2 Calcul de la projection

Nous avons

$$p_k = (I - A_k^t(A_k A_k^t)^{-1} A_k)(D_k c, -c^t x^k)^t,$$

posons

$$u_k = (A_k A_k^t)^{-1} A_k (D_k c, -c^t x^k)^t,$$

donc

$$p_k = (D_k c, -c^t x^k) - A_k^t u_k.$$

Le calcul de p_k dépend ainsi de la manière pour laquelle, on résout le système linéaire suivant

$$A_k A_k^t u_k = A_k (D_k c, -c^t x^k)^t.$$

Dans la matrice $A_k A_k^t$ est symétrique définie positive on distingue à ce propos, deux types de méthodes

• Méthode directes : basées essentiellement sur la factorisation de cholesky et concernent les systèmes à matrice pleine et de taille moyenne.

• Méthode itératives : concernent surtout les systèmes à grande dimension à matrice plutôt creuse, et sont en général de type gradient conjugué.

Comment choisir une méthode ?

Le choix d'une méthode de résolution d'un système linéaire, dépend naturellement des propriétés de la matrice en cause ; les deux éléments principaux à prendre en compte à cet égard sont le creux et le conditionnement de cette matrice.

(i) Le creux d'une matrice concerne le nombre et éventuellement la prépartition de ses éléments nuls appelés les "zéros" de la matrice.

C'est ainsi qu'on distingue les matrices creuses (beaucoup de zéros) et les matrices pleines (peu de zéros).

Les matrices creuses se prètent généralement bien à l'utilisation des méthodes itératives, alors que les méthodes directes sont plutôt réservées aux systèmes linéaires à matrices pleines (ne serait-ce que parce que les matrices pleines quelconques, ne vérifient pas en général les hypothèses de convergence des méthodes itératives).

De plus, l'espace mémoire nécessaire pour enregistrer une matrice pleine d'ordre n est évidemment plus important que celui d'une matrice creuse.

(ii) Le conditionnement d'une matrice A permet de mesurer la sensibilité de la solution x du système linéaire $Ax = b$, vis-à-vis des petites perturbations sur les données A et b. En effet, On démontre facilement [12] qu'en modifiant légèrement le second membre du système, la variation de la solution x est mesurée par l'inégalité

$$\frac{\|\Delta x\|}{\|x\|} \leq cond(A) \left(\frac{\|\Delta b\|}{\|b\|} \right) \tag{3.1}$$

De même, si l'on perturbe A on trouve

$$\frac{\|\Delta x\|}{\|x + \Delta x\|} \leq cond(A) \left(\frac{\|\Delta A\|}{\|A\|} \right) \tag{3.2}$$

Ainsi, "un système linéaire est bien (ou mal) conditionné", selon que le conditionnement de sa matrice est "petit" (ou "grand").

3.2.1 Méthode de Choleskey

Soit un système linéaire $Ax = b$ où A est une matrice réelle symétrique définie positive d'ordre n et b un élément de $\mathbb{R}^n$.

Le principe de cette méthode consiste tout simplement à décomposer (factorises) la

matrice A sous la forme $A = LL^t$ où L est une matrice triangulaire inférieure d'ordre n.

On remplace ainsi le système initial par deux systèmes triangulaires dont la résolution est beaucoup plus simple c.à.d on a
$Ax = b \Leftrightarrow LL^t x = b \Leftrightarrow Ly = b$ (1), $L^t x = y$ (2)
En général la matrice L est déterminée par identification des membres de l'égalité
$A = LL^t$.

3.2.2 Méthode du gradient conjugué

La méthode du gradient conjugué a été introduite par Stiefel et Hestnes en 1952 [5]. Afin de clarifier la présentation nous allons donné quelques rappels sur cette méthode. Considérons donc comme précédemment une matrice symétrique définie positive d'ordre n et un vecteur $b \in \mathbb{R}^n$, à A et b on associe la fonctionnelle q définie par

$$q(x) = \frac{1}{2} < Ax, x > + < b, x >$$

Le probléme de minimisation (PQ) $\{min\ q(x), x \in \mathbb{R}^n\}$ admet une solution unique (car q est strictement convexe) est également solution du système linéaire $Ax = b$ $(*)$
Le principe de cette méthode pour résoudre (PQ) ou $(*)$ est celui des méthodes de descente dans lesquels la solution est construite par le procédé itératif suivant

$$x^o \in \mathbb{R}^n \text{ (choisi arbitrairement)}$$
$$x^{k+1} = x^k + \alpha_k p_k.$$

Où la direction de descente p_k est généralement déduite de la direction du gradient de la fonction au point x^k. Le scalaire α_k appelé pas de déplacement est définie par

$$\alpha_k = \min_{\alpha > 0}\ q(x^k + \alpha p_k)$$

On vérifie facilement que dans le cas quadratique

$$\alpha_k = \frac{\|r_k\|^2}{< Ap_k, p_k >} \text{ avec } r_k = b - Ax^k$$

Il reste à définir la direction p_k
Plus précisément le déroulement de l'algorithme est le suivant
Après1 l'étape d'initialisation on construit les directions de descente p_k qui sont A-conjugué d'ou le nom de la méthode i.e vérifiant la propriété

$$< Ap_k, p_1 >= 0 \text{ si } k \neq 1$$

Autrement dit, supposons connue p_k alors on cherche p_{k+1} sous la forme $p_{k+1} = r_{k+1} + \beta_{k+1} p_k$ de sorte que $< Ap_{k+1}, p_k >= 0$.
On en déduit que

$$\beta_{k+1} = -\frac{< Ap_k, r_{k+1} >}{< Ap_k, r_k >}$$

on démontre en particulier les relations d'orthogonalité

$$< r_k, r_1 >= 0 \text{ si } k \neq 1, < r_k, p_1 >= 0 \text{ si } k > 1$$

D'ou $\beta_{k+1} = \dfrac{\|r_{k+1}\|^2}{\|r_k\|^2}$
Donnons maintenant l'algorithme détaillé correspondant à cette méthode

Algorithme du gradient conjugue

Début Algorithme
Initialisation : x^0 donné ;$p_0 = r_0 = b - Ax^0$; ϵ *prcision* donnée ; $k = 0$.
Tantque $\|r_k\| > \epsilon$ **faire**
$\alpha_k = \dfrac{\|r_k\|^2}{< Ap_k, p_k >}$;
$x^{k+1} = x^k + \alpha_k p_k$;
$r_{k+1} = r_k - \alpha_k A p_k$;
$\beta_{k+1} = \dfrac{\|r_{k+1}\|^2}{\|r_k\|^2}$;
$p_{k+1} = r_{k+1} + \beta_{k+1} p_k$
$k = k + 1$;
Fin tant que
Fin algorithme

3.3 Exemples numériques

Dans cette partie, nous présentons des tests numérique issus de l'implémentation de l'algorithme de Ye-Lustig pour confirmer et consolider les performances numériques de nos approches ces testes numériques. et sont réalisés en langage Matlab sur Pentium(R) Dual-Core CPU 4.00 Go T4300 @ 2.1GHZ .

En prenant le pas de déplacement α varié et la projection est calculée par les deux méthode précédents (Cholesky et gradient conjugué).

Notons par

Chol : méthode de Cholesky

g.c : méthode de gradient conjugué.

Nbr : le nombre d'itération nécessaire pour obtenir la solution optimale.

temps : le temps de calcul en second.

(m,n) : la dimension des exemples testés.

Exemple 3.4. *(Problème Cube)*

$$A[i,j] = \begin{cases} 1 & si \quad i = j \ ou \ j = i + m \\ 0 & sinon \end{cases}$$

$b[i] = 2 \quad i = 1, ..., m.$

$c[j] = -1 \quad j = 1, ..., n. \qquad (n = 2m).$

Exemple 3.5. *(Problème Hilbert)*

$A[i,j] = \dfrac{1}{i+j}$; $\quad A[j, m+j] = 1$

$b[i] = \sum \dfrac{1}{i+j}$; $c[i] = b[i] + \dfrac{1}{i+1}$;

$i = 1, ..., m$; $j = 1, ..., n, \ \ n = 2m.$

Programme de la méthode de Ye-Lustig en utilisant la méthode de Chol

```
function [k,o]= Chol(m,n)
k=0;eps=0.0001;
dk=1;alpham=3;
beta=0.99;
r=1/(sqrt(n*(n+1)));
en=ones(n,1);
A=input('Entrez les valeurs de la matrice A');
C=input('Entrez les valeurs de vecteur C');
b=minput('Entrez les valeurs de vecteur b ');
x0=(1/n)*en;
EN=ones(n+1,1);
xk=x0;
o=C'*x0;
tic
 while (norm(dk)/abs(o) > eps)
  Dk=diag(xk);
  Ak=[(A*Dk) -b];
  a=(Ak)*(Ak)';
  l= chol(a);
  d=l'*l;
  y=inv(l')*(Ak*[Dk*C;-C'*xk]);
  uk=inv(l)*y;
  pk=[Dk*C;-C'*xk]-(Ak)'*uk;
  dk=pk.*[Dk*C;-C'*xk];
  nye=n*r*C'*Dk*dk;
  re=n*r*en'*Dk*dk;
```

```
  for i=1:n
    v=(dk(i));
   if dk(i)>0
    alpham1=1/(n*r*v);
   if alpham1 < alpham
    alpham=alpham1;
   end
   end
   end
   alpham;
   if ((nye-re*C'*xk)>0)
   alphak=beta*alpham;
   else
   alphak=-beta*alpham;
   end
   yk=(EN/(n+1))-alphak*r*(dk/norm(dk));
   Y=yk(n+1);
   y=yk(1:n);
   xk=(Dk*y)./(Y);
   o=C'*xk;
   k=k+1;
 end
disp(o)
disp(k)
toc
end
```

Programme de la méthode de Ye-Lustig en utilisant la méthode de g.c

```
function [k,o]= gc(m,n)
k=0;eps=0.0001;
dk=1;alpham=3;
beta=0.99;
r=1/(sqrt(n*(n+1)));
en=ones(n,1);
A=input('Entrez les valeurs de la matrice A');
C=input('Entrez les valeurs de vecteur C');
b=minput('Entrez les valeurs de vecteur b ');
x0=(1/n)*en;
EN=ones(n+1,1);
xk=x0;
o=C'*x0;
tic
while (norm(dk)/abs(o) > eps)
     Dk=diag(xk);
   Ak=[(A*Dk) -b];
   a=Ak*(Ak)';
   X0=Ak*[Dk*C;-C'*xk];Xk=X0;
   P0=Ak*[Dk*C;-C'*xk]-a*X0;
   r0=P0;
   Pk=P0;rk=r0;
   while (norm(rk)>eps)
   alphakk=((norm(rk)^2)/((a*Pk)'*( Pk)));
   Xkk=(Xk)+alphakk*(Pk);
   rkk=rk-alphakk*a*Pk;
```

```
    betak=(norm(rkk))^2/(norm(rk))^2;
    Pkk=rkk + betak*Pk;
    pk=[Dk*C;-C'*xk]-(Ak)'*Xkk;
    dk=pk.*[Dk*C;-C'*xk];
     nye=n*r*C'*Dk*dk;
     re=n*r*en'*Dk*dk;
    for i=1:n
     v=(dk(i));
     if dk(i)>0
     alpham1=1/(n*r*v);
     if alpham1 < alpham
     alpham=alpham1;
     end
     end
     end
     alpham;
     if ((nye-re*C'*xk)>0)
     alphak=beta*alpham;
     else
     alphak=-beta*alpham;
     end
     yk=(EN/(n+1))-alphak*r*(dk/norm(dk));
     Y=yk(n+1);
     y=yk(1:n);
     xk=(Dk*y)./(Y);
     rk=rkk;

     Xk=Xkk;
     Pk=Pkk;
     o=C'*xk;
     k=k+1;
   end
end
disp(k)
disp(o)
toc
end
```

Dans les deux algorithmes itératifs (Chol et g.c) la précition est $\epsilon = 10^{-4}$

Les résultats numériques concernant ces exemple sont présentés respectivement dans les tableaux ci-dessous

(m,n)	Cohl			g.c		
	Nbr	$objectif$	$temps(S)$	Nbr	$objectif$	$temps(S)$
(5,10)	15	-0.0013	0.002934	15	-0.0013	0.002083
(10,20)	18	-0.0050	0.004269	18	-0.0050	0.002786
(25,50)	20	-0.0244	0.070004	20	-0.0244	0.080609
(50,100)	19	-0.0734	0.081174	19	-0.0734	0.077486
(75,150)	16	-0.1486	0.182210	16	-0.1486	0.101008
(99,198)	14	-0.2191	0.146616	14	-0.2191	0.107702

TABLE 3.4 – Tableau des résultats de l'exemple 3.4

(m,n)	Cohl			g.c		
	Nbr	$objectif$	$temps(S)$	Nbr	$objectif$	$temps(S)$
(5,10)	4	0.00000040	0.9	4	0.00000037	18.5
(10,20)	5	0.00000155	5.1	5	0.00000150	38
(25,50)	6	0.00000203	65	6	0.00000197	132.2
(50,100)	6	0.00000546	468.7	6	0.00000546	579.3
(75,150)	7	0.00000934	1627.4	7	0.0000931	1749.3
(99,198)	8	0.00008366	3668.6	8	0.00008361	3769.7

TABLE 3.5 – Tableau des résultats de l'exemple 3.5

Remarque.

En basant sur les résultats précédente on peut faire les remarques suivants

• *En présence d'une matrice creuse et bien conditionnée (problème Cube) c'est la méthode de gradient conjugué qui l'emporte sur Cholesky.*

• *Il est normal que la factorisation de Cholesky soit souvent plus performante que la méthode de gradient conjugué puisque d'une part, la matrice AA^t peut être pleine et mal conditionné même si A ne l'est pas, et d'autre part pour les problèmes de taille moyenne, les méthodes directes sont mieux à l'utilisation. L'efficacité des méthodes itératives est à espérer pour les problèmes de taille importante que nous n'avons pas pu testé a cause des capacités limitées de la machine utilisée.*

Conclusion

Notre travail concerne l'étude théorique, algorithmique et numérique de la méthode de Karmarkar et sa variante de Ye-Lustig pour résoudre les problèmes de programmation linéaire.

Nous mettons l'accent sur le calcul du pas de déplacement et le calcul économique de la projection .

Ce travail est consolidé par des tests numériques réalisés sur les deux algorithmes.

Bibliographie

[1] **Dennis Je Jr, Morshedi Am, Turner K**, *A variable-metric variant of the karmarkar algorithm for linear programming.* Rice univ. matihematical sci.dep./Houston TX 77251-1892/USA (1986)

[2] **Diderrich GT**, *Some remarks of Karmarkar's potential function.* Johnson controls inc. /Milwaukee wl 53201/USA (1986).

[3] **Gay Dm** , *A variant of Karmarkar's linear programming algorithm for problems in standard form.* AT et T Bell Laboratorie/Murray Hill NJ 07974/USA(1985).

[4] **Goldfarb D, Mehrotra S**, *A self-correcting version of Karmarkar's algorithm*, res./New York Ny 10027/USA (1987).

[5] **Golub. G.H, C.F. Vanloan**, *Matrix computations.* Johns Hopkins University press (1983).

[6] **Irven J. Lustig**, *A partical approach to Karmarkar's algorithm*, Technical report sol 85-5 june 1985.
Systems optimization laboratory, dep of operations res.

[7] **Keraghel. A**, *Etude adaptative et comparative des principales variantes dans l'algorithme de Karmarkar.* Thèse de Doctoral de mathématiques appliquées de l'université Joseph Fourier de Grenoble (1989).

[8] **Lisser. A, N. Maculan, M. Minoux**, *Large steps preserving polynomiality in Karmarkar's algorithm.*
Lamsade université de paris-Dauphine cahier №77 mai 87 unité associée au C.N.R.S №825.

[9] **Lisser. A, Tolla P**, *Variants of Karmarkar's algorithm.* Lamsade, univ. Paris-Dauphine(1986).

[10] **L. Menniche**, *Etude théorique et numérique d'une classe de méthodes de points intérieurs pour la programmation linéaire*, (2017).

[11] **Shanno Df**, *Computing Karmarkar projection quickly.* Rutcor/New Brunswick Nj/USA(1985).

[12] **N.F. Stewart, F.Jensen**, *Solution numérique des problèmes matriciels.* Les presses de l'université de Montréal c.p.6128 Montrèal 101 Canada Editions Eyrolles, Paris(1975).