FRANÇOIS GLINEUR

ETUDE DES METHODES DE POINT INTERIEUR APPLIQUEES A LA PROGRAMMATION LINEAIRE ET A LA PROGRAMMATION SEMIDEFINIE

Service de Mathématiques et
de Recherche Opérationnelle

Promoteurs
Professeur J. TEGHEM
Docteur D. TUYTTENS

5$^{\text{ème}}$ Informatique & Gestion
Année académique 1996-1997

FACULTE POLYTECHNIQUE DE MONS

Année académique 1996-1997

Travail de fin d'études

FRANÇOIS GLINEUR
5ème Informatique & Gestion

ETUDE DES METHODES DE POINT INTERIEUR APPLIQUEES A LA PROGRAMMATION LINEAIRE ET A LA PROGRAMMATION SEMIDEFINIE

Mots-clefs :
Optimisation mathématique,
Programmation mathématique,
Programmation linéaire,
Programmation non-linéaire.

Promoteurs

Professeur J. TEGHEM
Docteur D. TUYTTENS

Service de Mathématiques et de Recherche Opérationnelle
Juin 1997

REMERCIEMENTS

Je tiens tout particulièrement à exprimer ici ma reconnaissance envers mes promoteurs, Monsieur le Professeur Teghem et Monsieur Tuyttens, tant pour leur direction efficace et leur écoute attentive que pour leurs remarques constructives, qui ont grandement contribué à améliorer la qualité de ce travail de fin d'études.

Je tiens également à remercier l'ensemble du personnel du service de Mathématiques et de Recherche Opérationnelle pour le soutien moral et matériel qu'il m'a apporté tout au long de ce travail, en particulier Monsieur Fortemps pour sa relecture attentive et ses commentaires judicieux.

A. INTRODUCTION

A.1 PRESENTATION GENERALE

La programmation mathématique, branche de l'optimisation, s'occupe de la minimisation sous contraintes d'une fonction à plusieurs variables, schéma très général s'appliquant à de nombreuses situations pratiques dans beaucoup de domaines (minimisation de coûts, de durées, etc.)

Dans le cas d'une fonction et de contraintes linéaires (programmation linéaire), on dispose d'une méthode efficace de résolution : l'algorithme du simplexe, découvert par Dantzig en 1947. Cet algorithme a connu depuis lors de nombreuses améliorations, et est utilisé dans la majorité des logiciels commerciaux.

Cependant, un nouveau type de méthodes de résolution a fait son apparition en 1984 : les méthodes de point intérieur. La plupart des idées sous-jacentes à ces méthodes proviennent du domaine de la programmation non-linéaire. Parmi leurs avantages, citons

- Efficacité théorique : il est possible de prouver que ces méthodes s'exécutent en temps polynomial (ce qui n'est pas le cas de l'algorithme du simplexe, de nature exponentielle)
- Efficacité pratique : les temps de calcul de ces méthodes sont compétitifs (et battent l'algorithme du simplexe dans le cas de problèmes de grande taille)
- Traitement de très grands problèmes : ces méthodes permettent de résoudre des problèmes de très grande taille qu'aucun autre algorithme connu ne pourrait traiter en un temps acceptable
- Adaptation au cas non-linéaire : il est possible d'adapter ces méthodes à la programmation non-linéaire (plus particulièrement à la programmation convexe), ce qui permet de traiter de nouveaux types de problèmes pour lesquels on ne connaissait jusqu'à présent aucune méthode de résolution efficace.

L'objectif de ce travail de fin d'études est de donner un aperçu de ce domaine relativement récent, en fournissant une description claire et compréhensible de ces méthodes, tant du point de vue théorique que de celui de leur implémentation sur ordinateur et de leurs performances pratiques.

Nous nous attacherons d'abord à l'examen du cas linéaire, le plus étudié et le plus utilisé, avant de nous concentrer sur un cas particulier important de la programmation non-linéaire, la programmation semidéfinie, possédant de nombreuses applications. Enfin, nous terminerons en décrivant une implémentation sur ordinateur, réalisée dans le cadre de ce travail de fin d'études, utilisant les méthodes de point intérieur appliquées à la programmation semidéfinie pour résoudre un problème de classification de données (analyse discriminante par ellipsoïdes).

A.2 STRUCTURE DE CE TRAVAIL DE FIN D'ETUDES

Après cette brève introduction, nous consacrerons un premier chapitre à quelques généralités sur le domaine que nous allons étudier, ainsi qu'à quelques rappels au sujet de la méthode du simplexe. La suite sera divisée en deux parties : *programmation linéaire* et *programmation semidéfinie*.

La *première partie* débute par un important chapitre consacré aux développements théoriques relatifs aux méthodes de point intérieur pour la programmation linéaire. En particulier, on y trouvera une présentation de la démarche conceptuelle de création de ces méthodes de point intérieur.

Le chapitre suivant traitera les aspects relatifs à l'implémentation de ces méthodes sur ordinateur. En effet, l'efficacité d'un logiciel utilisant ces méthodes dépendra fortement du soin que l'on aura apporté à son implémentation. On trouvera ensuite un chapitre renfermant quelques compléments théoriques. Enfin, nous terminerons cette première partie par un chapitre fournissant les résultats de tests numériques ainsi que quelques comparaisons entre différentes méthodes et logiciels.

La *seconde partie* débute par les développements théoriques relatifs à la programmation semidéfinie. Nous y insisterons tout particulièrement sur le parallèle avec la programmation linéaire qui peut être établi pour la plupart des concepts. Le chapitre suivant fournira une série de problèmes d'optimisation modélisables par la programmation semidéfinie. Nous terminerons cette partie par la présentation de l'implémentation que nous avons réalisée pour résoudre un problème de classification par la programmation semidéfinie.

Enfin, nous clôturerons ce travail par un chapitre dédié aux conclusions, suivi des références, d'une bibliographie et des annexes.

A.3 TABLE DES MATIERES

A.4 TABLE DES ILLUSTRATIONS

B. GENERALITES

B.1 RECHERCHE OPERATIONNELLE ET PROGRAMMATION MATHEMATIQUE

B.1.1 Présentation et définitions

Les définitions de la recherche opérationnelle que l'on rencontre couramment ressemblent généralement à

> *Recherche opérationnelle :* ensemble des techniques rationnelles d'analyse et de résolution de problèmes concernant, notamment, l'activité économique et visant à élaborer les décisions les plus efficaces pour aboutir au meilleur résultat.
>
> Petit Larousse illustré (1996).

Cette discipline trouve ses origines durant la seconde guerre mondiale, où elle servit entre autres à élaborer des stratégies de défense aérienne et à résoudre des problèmes logistiques. Elle est à présent un outil de gestion couramment employé, et s'applique à des domaines aussi variés que l'ordonnancement de production, les systèmes de transport, la gestion des stocks, les télécommunications, etc. En tant que branche des mathématiques appliquées, elle recherche principalement des procédures de calcul efficaces et a de ce fait connu de formidables progrès suite au développement des ordinateurs modernes.

Le but de la recherche opérationnelle est de modéliser une situation afin de déduire de cette représentation idéalisée la ou les meilleures décisions à prendre. La notion de « meilleure décision » correspond dans le modèle à celle qui maximise un ou plusieurs critères déterminés. C'est pourquoi le domaine mathématique de l'optimisation est étroitement lié à la recherche opérationnelle, puisqu'il s'occupe du problème de la maximisation (ou de la minimisation) de fonctions de plusieurs variables.

En particulier, la *programmation mathématique* sera très utile au praticien de la recherche opérationnelle. En effet, il s'agit de la branche de l'optimisation qui s'occupe de maximiser ou minimiser un objectif sous certaines contraintes. Ceci se traduit mathématiquement de la façon suivante (dans le cas d'une minimisation)

$$\min f(x), \quad x \in D \subseteq \Re^n$$

où **x** est un vecteur de dimension n représentant les variables que l'on cherche à optimiser dans le problème considéré, f une fonction scalaire traduisant le critère selon lequel on va évaluer les diverses valeurs de **x** et D le domaine des valeurs admissible pour **x**, traduisant les contraintes du problèmes, c'est-à-dire les restrictions sur les valeurs de **x** que l'on peut accepter. Ce domaine se définit le plus souvent à l'aide de contraintes du type $g_i(\mathbf{x}) = 0$ et $h_j(\mathbf{x}) \geq 0$ pour différentes valeurs des indices i et j.

B.1.2 Applications

On peut citer les quelques exemples suivants d'applications de la programmation mathématique en recherche opérationnelle :

- Déterminer la meilleure façon de tracer les pistes conductrices sur un circuit imprimé (c'est-à-dire minimisant leur longueur et/ou le nombre de croisements, par exemple)
- Déterminer la politique optimale de gestion d'un stock, à savoir les instants et les quantités des commandes de réapprovisionnement
- Déterminer la taille optimale d'un central téléphonique en fonction de la distribution des appels qu'il sera amené à recevoir et du temps d'attente maximal qu'un client peut tolérer

Comme on peut le constater, l'ensemble des domaines abordés est très diversifié.

B.2 PROGRAMMATION LINEAIRE

B.2.1 Présentation et définitions

La *programmation linéaire* s'occupe d'un cas particulier du problème défini ci-dessus, celui d'un objectif et de contraintes linéaires. La fonction objectif f est alors une somme pondérée des composantes de **x** (certains coefficients pouvant bien sûr être nuls ou négatifs), et les contraintes $g_i(\mathbf{x}) = 0$ et $h_j(\mathbf{x}) \geq 0$ peuvent alors se représenter sous forme matricielle, pour donner le problème suivant (appelé programme linéaire)

$$\begin{gathered}\min c^T x, \quad x \in \Re^n \\ \text{avec } Ax = b \text{ et } Dx \geq e\end{gathered}$$

Les données **b**, **c**, **e** ainsi que **x** sont donc des vecteurs colonnes réels (respectivement de dimension l, n, m et n), alors que **A** et **D** sont des matrices de dimensions (l x n) et (m x n). Le signe ≥ est à prendre composante par composante. Dans la suite de ce document, on emploiera des lettres minuscules et majuscules pour désigner respectivement les vecteurs et les matrices. Par contre, on renoncera à l'emploi de caractères gras pour les différencier des quantités scalaires, le contexte indiquant clairement la nature des variables considérées.

On peut montrer que tout programme linéaire peut se ramener à l'une des deux formes suivantes, par adjonction de variables et/ou de contraintes

$$\begin{array}{ll}\min c^T x, \quad x \in \Re^n & \qquad \min c^T x, \quad x \in \Re^n \\ \text{avec } Ax \geq b \text{ et } x \geq 0 & \qquad \text{avec } Ax = b \text{ et } x \geq 0\end{array}$$

La première de ces deux formes se nomme *forme canonique*, la seconde *forme standard.* La matrice A est ici de taille (m x n), d'où l'appellation « programme à m contraintes et n variables ».

Notons que l'on suppose couramment que la matrice A est de plein rang. En effet, le cas contraire signifierait qu'une des contraintes (c'est-à-dire une ligne) est combinaison linéaire d'autres contraintes. Selon la valeur de b, cette contrainte est alors soit redondante (on peut alors la supprimer sans la modifier), soit contradictoire, ce qui implique que le problème est impossible. On peut donc se placer dans le cas où A est de rang maximal sans perdre aucune généralité.

B.2.2 Applications

Voici quelques exemples de problèmes qui se traitent par la résolution d'un programme linéaire

- La planification de production, où l'on dispose de plusieurs machines et d'une capacité finie de production à répartir entre plusieurs types de produits. Si l'on attribue un gain unitaire à chaque produit fabriqué, la programmation linéaire fournira la composition du plan de production idéal (entraînant un bénéfice maximum).
- Le problème de mélange : par exemple, une firme pétrolière dispose de plusieurs sortes de pétrole brut de différentes qualités. Etant données les contraintes de teneurs minimales et maximales en certains types d'hydrocarbures (dépendant de l'utilisation envisagée du pétrole), l'objectif du programme linéaire sera de maximiser le profit (c'est-à-dire en employant les sortes les moins coûteuses) tout en respectant les contraintes de composition et de disponibilité.
- Le problème de transport s'occupe par exemple de calculer la planification du transfert de biens entre plusieurs centres de production (ou dépôts) et plusieurs centres de consommation (ou de vente), étant donnés des coûts de transport spécifiques à chaque couple centre de production/centre de consommation (dépendant principalement de la distance entre les deux sites).
- Il faut également noter que la résolution d'un programme linéaire intervient également dans le traitement de problèmes non-linéaires :
    - Certains problèmes non-linéaires peuvent être approchés par un programme linéaire. On peut souvent améliorer la qualité de l'approximation au prix d'une augmentation du nombre de contraintes.

♦ Il existe des méthodes de résolution de certains problèmes (comme le *Branch & Bound* en programmation en nombre entiers, utilisé dans la résolution de nombreux problèmes de type combinatoire) qui nécessitent le calcul de l'optimum de nombreux sous-problèmes linéaires.

B.3 METHODE DU SIMPLEXE

La programmation linéaire est de loin la branche de la programmation mathématique la plus utilisée dans les applications pratiques. Outre le fait que ce soit une façon assez naturelle de modéliser un problème, cette popularité s'explique également par l'existence d'un algorithme de résolution des programmes linéaires très efficace : l'algorithme du simplexe, découvert en 1947 par Georges B. Dantzig *[DAN63]*.

Nous allons à présent esquisser le principe de base de cette méthode.

B.3.1 Principe de l'algorithme du simplexe

Partant de la forme canonique d'un programme linéaire,

$$\min c^T x, \quad x \in \Re^n$$
$$\text{avec } Ax \geq b \text{ et } x \geq 0$$

on constate que chacune des m contraintes définies par le couple (A, b) ampute le domaine de variation de x d'un demi espace bordé par un hyperplan. Il est dès lors clair que le domaine de x résultant de l'ensemble des contraintes est un polyèdre convexe (de plus, si ce domaine est borné et non vide, c'est un polytope).

L'objectif à minimiser étant linéaire, les ensembles de points prenant la même valeurs sont des hyperplans parallèles, et l'on conclut qu'on peut toujours trouver une solution optimale sur un des sommets du polyèdre (néanmoins, il se peut que l'ensemble des points d'une face du polyèdre - y compris ses sommets - soient optimums si cette face est parallèle aux hyperplans mentionnés). La Figure B-1 donne un aperçu de la situation dans le cas à deux dimensions.

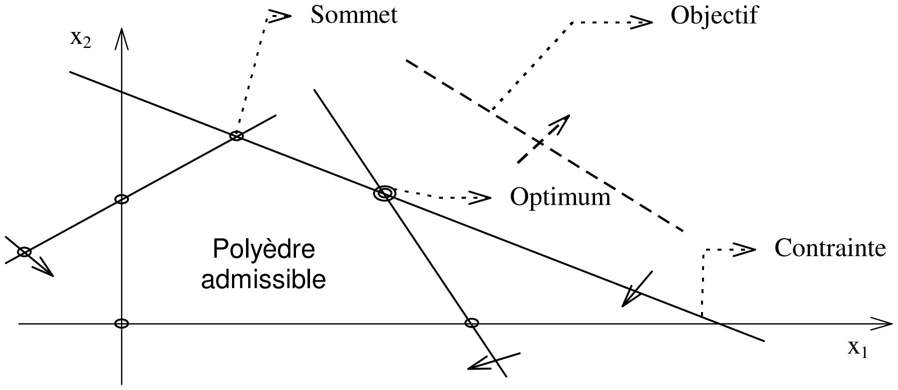


Figure B-1 - Exemple de polyèdre admissible (cas à deux dimensions)

Outre ces remarques d'ordre géométrique, l'idée principale de la méthode du simplexe est de caractériser les sommets du polyèdre de façon algébrique, en faisant correspondre à chacun de ces sommets ce qu'on appelle une base admissible. Sans rentrer dans les détails, l'algorithme consiste à partir d'un sommet initial (donc d'une base admissible) puis de passer successivement d'un sommet à un sommet adjacent, en améliorant constamment la valeur de l'objectif. Le changement de base se réalise à l'aide d'une procédure simple de calcul ne faisant intervenir que des opérations élémentaires.

Lorsqu'il n'est plus possible de passer à un sommet adjacent sans diminuer la valeur de l'objectif, on stoppe l'algorithme. Dantzig a démontré d'une part que cela se produit toujours après un nombre fini d'itérations, et d'autre part que l'on a alors forcément atteint un optimum, ce qui achève de prouver la validité de la méthode.

B.3.2 Caractéristiques de l'algorithme du simplexe

Cette méthode revient donc à une exploration combinatoire du polyèdre admissible des solutions. On peut dégager les remarques et propriétés suivantes :

- Cet algorithme nécessite un point de départ admissible. Cependant, il existe une méthode permettant, en deux phases, de résoudre les problèmes où l'on ne dispose pas d'un tel point de départ.
- Cet algorithme ne permet pas de traiter naturellement les variables libres, c'est-à-dire des variables sans contraintes de positivité. Il faut alors décomposer ces variables selon

  $$x \in \Re \Leftrightarrow \begin{cases} x = x_+ - x_- \\ x_+ \geq 0 \text{ et } x_- \geq 0 \end{cases}$$

  ce qui augmente la taille du vecteur x et de la matrice A (et donc du problème). Par contre, il existe une amélioration très simple de l'algorithme qui permet d'introduire des bornes inférieure et supérieure ($x_l \leq x \leq x_u$) sur les variables sans ajouter de contraintes explicites supplémentaires.
- Cet algorithme est sensible à la dégénérescence (le fait que certains des hyperplans correspondant aux contraintes soient concourants), qui diminue ses performances et peut entraîner des problèmes de cyclage. Dans les applications réelles, presque tous les problèmes sont dégénérés.
- Bien que très efficace en pratique (il est rare que le nombre d'itérations dépasse n + m), cette méthode ne satisfait pas les théoriciens de la complexité. En effet, il a été prouvé que cet algorithme a une complexité de pire cas de type exponentiel.

  Cela signifie qu'il existe des cas où le nombre d'itérations nécessaires pour atteindre la solution est exponentiel en fonction de la taille du problème (n). (V. Klee et G. Minty ont exhibé un tel problème en 1970, *[KLE72]*). En fait, au moment de la découverte de la méthode du simplexe, personne ne savait si un algorithme de type polynomial (donc théoriquement meilleur que ceux de type exponentiel) existait pour la programmation linéaire.

B.4 PREMIERE PRESENTATION DES METHODES DE POINT INTERIEUR

Nous sommes à présent en mesure de donner une première description des méthodes de point intérieur, en particulier d'expliciter leur dénomination. Plaçons nous dans le cas du programme linéaire dans sa forme standard

$$\begin{gathered} \min c^T x, \quad x \in \Re^n \\ \text{avec } Ax = b \text{ et } x \geq 0 \end{gathered}$$

A la différence de l'algorithme du simplexe, les méthodes de point intérieur partent d'un point situé *à l'intérieur* du polyèdre admissible. Pour être plus précis, ces méthodes nécessitent généralement un point de départ *strictement* intérieur, soit $x \in \Im_0$ avec les définitions suivantes

$$\Im = \{x \mid Ax = b \text{ et } x \geq 0\} \qquad \Im_0 = \{x \mid Ax = b \text{ et } x > 0\}$$

Le principe des méthodes de point intérieur consiste alors à faire évoluer *de façon itérative* ce point vers une solution optimale du problème, tout en restant à l'intérieur strict ($\Im_0$) du polyèdre admissible. En d'autres termes, on construit une suite x_0, x_1, x_2, ..., x_i, ... d'itérés intérieurs convergeant vers une solution, comme illustré sur la Figure B-2.

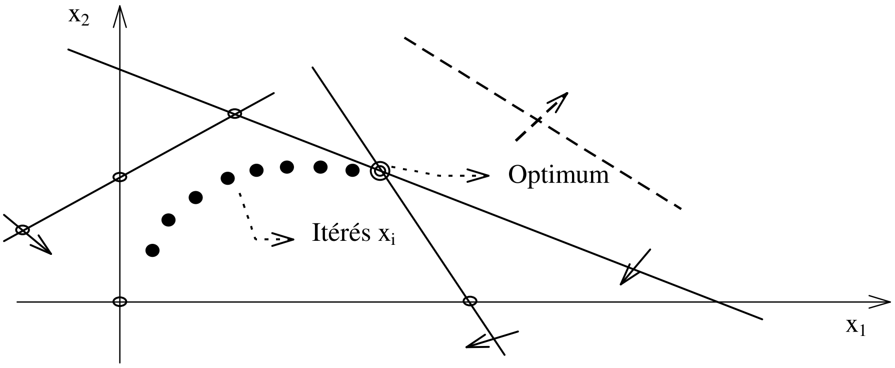


Figure B-2 - Convergence des itérés d'une méthode de point intérieur

Ce principe de construction d'itérés, que l'on retrouve dans de nombreuses méthodes d'optimisation ou de résolution d'équations, est fondamentalement différent de la méthode du simplexe. En effet, celle-ci finit par donner, en un nombre *fini* d'étapes, la valeur exacte du x optimum, alors que les méthodes de point intérieur se contentent de faire converger x. En fait, x ne peut pas atteindre exactement l'optimum puisque l'on veille à ce que les itérés restent à l'intérieur strict du polyèdre $\Im_0$, alors qu'on peut montrer que les x optimums contiennent des composantes nulles, donc appartiennent à $\Im \setminus \Im_0$.

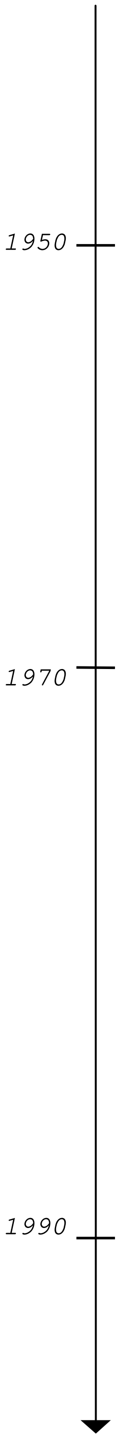
Cependant, il faut signaler que ce qui peut paraître comme un inconvénient n'en est pas un en réalité : d'une part, une solution approchée (par exemple avec une précision relative de 10^{-8}) est suffisante pour les applications pratiques et d'autre part, les calculs de l'algorithme du simplexe sont eux aussi de toutes façons réalisés avec une précision donnée. En fait, si l'on ne requiert qu'une solution grossière (par exemple avec une précision relative de 10^{-2}), les méthodes de point intérieur présentent même un certain intérêt puisqu'elles peuvent fournir avec moins d'itérations (donc plus rapidement) une approximation de la solution.

Il est également important de noter que ces méthodes ne fournissent pas le même type de solution que l'algorithme du simplexe. En effet, si la solution n'est pas unique (c'est-à-dire lorsqu'une face entière du polyèdre est solution), on peut montrer que les méthodes de point intérieur fourniront une solution située à l'intérieur de cette face, à l'inverse du simplexe qui donnera un de ses sommets.

La troisième différence importante se situe au niveau de la complexité algorithmique de ces méthodes (ou du moins de la plupart d'entre elles) : elles sont de type polynomial. Etant donné une précision ε, le nombre d'opérations à effectuer pour obtenir une solution approchée à ε près est borné par un polynôme fonction de la taille du problème. Cette dernière propriété est d'ailleurs à elle seule responsable du formidable engouement qu'ont témoigné les chercheurs en optimisation pour ces méthodes.

B.5 HISTORIQUE DU DOMAINE

Nous allons à présent donner un bref aperçu historique du domaine de la programmation linéaire, afin de situer la découverte des méthodes de point intérieur dans son contexte.



B.5.1 Débuts de la programmation linéaire - Algorithme du simplexe

1930–1940	Premières formulations de problèmes de programmation linéaire
1939–1945	Seconde guerre mondiale. La recherche opérationnelle fait ses débuts, applications militaires
1947	Georges B. Dantzig publie un article relatif à l'algorithme du simplexe de programmation linéaire *[DAN63]*
1956	A. J. Goldman et A. W. Tucker démontrent l'existence d'une solution strictement complémentaire au problème de programmation linéaire *[GOL56]*
1970	V. Klee et G. Minty démontrent le caractère exponentiel de la complexité algorithmique de la méthode du simplexe *[KLE72]*

B.5.2 Débuts des méthodes de point intérieur et complexité polynomiale

1955	K. R. Frisch invente une méthode de point intérieur pour résoudre un problème non-linéaire (méthode de potentiel logarithmique, utilisation d'une fonction barrière, cf. plus loin) *[FRI55]*
1968	A. V. Fiacco et G. P. McCormick développent l'utilisation des méthodes de point intérieur en programmation convexe non-linéaire *[FIA68]*
1978	L. G. Kachian applique la méthode de l'ellipsoïde (développée par N. Shor en 1970, *[SHO70]*) à la programmation linéaire et prouve qu'elle est de complexité polynomiale *[KHA79]*. Cette méthode, bien que faisant converger une suite d'itérés, n'est cependant pas une méthode de point intérieur au sens accepté actuellement.

Il faut remarquer qu'à cette l'époque, on utilise les méthodes de point intérieur dans le cadre non-linéaire uniquement. Bien que Fiacco et McCormick notent que leurs méthodes sont également applicables au cas linéaire, ils ne les considèrent pas sérieusement comme une alternative viable à l'algorithme du simplexe.

De plus, la méthode de Kachian, bien que théoriquement supérieure à l'algorithme du simplexe (du point de vue de la complexité de pire cas), est en pratique bien plus lente, principalement parce qu'elle atteint son pire cas pour la majorité des problèmes, alors que le simplexe n'exhibe qu'exceptionnellement son comportement de pire cas exponentiel *[BOR87]*.

B.5.3 Explosion des méthodes de point intérieur

1984	N. K. Karmarkar découvre une méthode de point intérieur polynomiale plus efficace que celle de Kachian, qu'il affirme également comme supérieure à l'algorithme du simplexe *[KAR84]*.
1994	Y. Nesterov et A. Nemirovsky publient leur étude sur les méthodes de point intérieur polynomiales appliquées à la programmation convexe (cf. plus loin) *[NES94]*
1997	Depuis l'annonce de Karmarkar, plus de 2000 articles de recherche portant sur les méthodes de point intérieur ont été publiés par la communauté scientifique *[FRE96]*. Les premiers ouvrages récapitulatifs paraissent. Les recherches se dirigent vers la programmation non-linéaire.

Avec le recul, il faut modérer l'affirmation de Karmarkar : sa méthode n'était en définitive pas compétitive avec les meilleures implémentations de l'algorithme du simplexe disponibles à l'époque, surtout pour des problèmes de petite taille. Néanmoins, elle a eu le mérite de susciter de nombreuses recherches dans ce domaine.

Signalons encore que la méthode de Kachian n'est pas à proprement parler la première méthode polynomiale de résolution du problème linéaire. En effet, il a été montré *a posteriori [ANS90]* que la méthode de Fiacco et McCormick était de complexité polynomiale lorsqu'elle était appliquée à la programmation linéaire. Ainsi, on disposait, dès 1968, - mais sans en être conscient - d'une méthode polynomiale pour la programmation linéaire.

PREMIERE PARTIE
LES METHODES DE POINT INTERIEUR POUR LA PROGRAMMATION LINEAIRE

C. DEVELOPPEMENTS THEORIQUES

C.1 DUALITE

C.1.1 Couple de problèmes primal-dual

La théorie de la programmation linéaire nous apprend qu'il est possible d'associer à tout programme linéaire (P) un programme (D), nommé programme dual par opposition à (P), le programme primal. Il existe en fait une série de règles simples transformant les variables de (P) en contraintes dans (D), ainsi que les contraintes de (P) en variables dans (D) (voir par exemple *[SCH86]*). De plus, lorsque le primal cherche à minimiser sa fonction objectif, le dual est un problème de maximisation. En fait, le dual du dual est identique au primal.

Dans le cas de la forme standard, la plus couramment utilisée dans le domaine des méthodes de point intérieur, le couple de problèmes primal-dual est le suivant

$$\begin{array}{ll}\min c^T x, \quad x \in \Re^n \\ \text{avec } Ax = b \text{ et } x \geq 0\end{array} \text{(P)} \qquad \begin{array}{ll}\max b^T y, \quad y \in \Re^m \\ \text{avec } A^T y \leq c\end{array} \text{(D)}$$

On peut donc ramener tout problème linéaire, au choix, à la forme (P) ou (D). Il est à noter que la variable y est libre dans le problème dual (pas de contrainte de non-négativité). Cependant, on transforme généralement le problème dual par l'introduction de variables supplémentaires appelées variables d'écart (*slacks* en anglais) afin de remplacer les contraintes d'inégalité par des contraintes d'égalité.

$$\begin{array}{l}\max b^T y, \quad y \in \Re^m \text{ et } s \in \Re^n \\ \text{avec } A^T y + s = c \text{ et } s \geq 0\end{array}$$

Bien que ce problème puisse sembler plus compliqué que le précédent, la présence de la variable s simplifie l'expression de nombreux résultats théoriques. Etant donné que ces deux problèmes sont extrêmement liés, on les considérera simultanément la plupart du temps, ce qui nous amène à redéfinir nos concepts de point intérieur et strictement intérieur, pour englober les deux problèmes

$$\Im = \left\{(x,y,s) \mid Ax = b, A^T y + s = c \text{ et } x \geq 0, s \geq 0\right\} \qquad \Im_0 = \left\{(x,y,s) \mid Ax = b, A^T y + s = c \text{ et } x > 0, s > 0\right\}$$

C.1.2 Propriétés de la dualité

Citons quelques propriétés élémentaires du couple primal-dual *[SCH86]*

- Pour tout vecteur (x,y,s) $\in \Im$ (c'est-à-dire vérifiant les contraintes du couple (P) & (D), on parle alors de solution admissible mais pas nécessairement optimale), on a l'inégalité $c^T x \geq b^T y$, ce qui signifie que toute solution admissible du dual (respectivement du primal) fournit une borne inférieure (respectivement supérieure) à l'objectif du problème primal (respectivement dual).
- (x,y,s) $\in \Im$ et $c^T x = b^T y \Leftrightarrow$ x et y sont solutions optimales de leurs problèmes respectifs. Cela entraîne que les problèmes (P) et (D) ont les mêmes valeurs optimales.
- La quantité $c^T x - b^T y$, toujours positive, est appelée le saut de dualité (*duality gap*). On montre aisément, par de simples manipulations algébriques, que $c^T x - b^T y = x^T s$. Pour tout point admissible, le saut de dualité est nul si et seulement si la solution est optimale (corollaire du point précédent)
- Si l'un des deux problèmes est non borné (de valeur optimale infinie), l'autre est impossible ($\Im = \emptyset$). Si l'un des deux problèmes a une solution optimale finie, il en est de même pour l'autre (et leurs valeurs optimales sont égales)

- Pour toute solution optimale (x,y,s), on a $x_i s_i = 0$ (c'est-à-dire au plus une des deux variables x_i et s_i est non nulle) pour tout i. Ce résultat est connu sous le nom de théorème des écarts complémentaires.
- Si $\Im \neq \emptyset$, il existe au moins une solution (x,y,s) telle que $x + s > 0$. Une telle solution est dite strictement complémentaire, puisqu'elle interdit la présence d'un indice i pour lequel x_i et s_i seraient simultanément nuls. Tenant compte de la propriété précédente, on en déduit que pour une telle solution, on a à chaque indice i l'alternative ($x_i = 0$ et $s_i > 0$) ou ($x_i > 0$ et $s_i = 0$). Les zéros des deux vecteurs x et s sont donc en quelque sorte complémentaires. Ce résultat est connu sous le nom de théorème de complémentarité de Goldman-Tucker (ou théorème fort des écarts complémentaires) *[GOL56]*.

C.1.3 Dualité et conditions KKT

Les conditions de Karush-Kuhn-Tucker (KKT) sont des conditions *nécessaires* d'optimalité valables dans le cadre très général de l'optimisation non-linéaire sous contraintes avec objectif différentiable. Elles s'appliquent donc bien sûr à la programmation linéaire, et sont même dans ce cas *suffisantes* (ceci étant dû à la propriété de convexité de ce problème particulier).

Nous allons à présent réexaminer la dualité à la lumière des conditions KKT. Appliquons-les au problème primal (P) : on obtient alors les conditions nécessaires et suffisantes d'optimalité suivantes

$$\begin{cases} A^T z + t = c & \text{(KKT 1)} \\ Ax = b & \text{(KKT 2)} \\ x_i t_i = 0 \ \forall i & \text{(KKT 3)} \\ x \geq 0 \text{ et } t \geq 0 & \text{(KKT 4)} \end{cases}$$

Ces conditions introduisent de nouvelles variables z et t. Ceci est à prendre au sens « x est optimum si et seulement si il existe des variables z et t telles que (x,z,t) vérifient les conditions KKT ». En fait, on peut montrer que ces variables sont en réalité les multiplicateurs de Lagrange des contraintes (d'égalité et de non-négativité) du problème primal.

Si l'on regarde ces conditions de plus près, on retrouve les conditions d'admissibilité de x pour le problème primal (KKT 2 et la première partie de KKT 4), une égalité de type linéaire (KKT 1), une contrainte de non-négativité (seconde partie de KKT 4) ainsi qu'une équation non-linéaire (KKT 3). Une analyse un peu plus détaillée de la forme de ces équations permet de découvrir que (KKT1) et la seconde inégalité de (KKT4) définissent un problème identique au problème dual décrit plus haut (p. 13) : il suffit en effet de remplacer z par y et t par s. On obtient alors le système

$$\begin{cases} A^T y + s = c & \text{(KKT 1)} \\ Ax = b & \text{(KKT 2)} \\ x_i s_i = 0 \ \forall i & \text{(KKT 3)} \\ x \geq 0 \text{ et } s \geq 0 & \text{(KKT 4)} \end{cases}$$

Cela explicite donc clairement les liens très forts qui unissent le problème primal à son dual : les conditions d'optimalité du primal sont simplement formées de ses propres conditions d'admissibilité, de celles de son problème dual et de l'égalité non-linéaire (KKT 3). De même, si l'on appliquait les conditions KKT au problème dual, on retrouverait l'égalité Ax=b, la condition de non-négativité $x \geq 0$ ainsi que l'équation non-linéaire (KKT 3).

Ceci nous permet de conclure que les problèmes primal et dual sont en fait deux versions du même problème fondamental (ou « les deux faces de la même pièce de monnaie », *[WRI96]*), puisqu'ils admettent exactement les mêmes conditions d'optimalité. Nous verrons plus loin que les méthodes de point intérieur de type primal-dual, conformément à cette réalité mathématique profonde, traitent simultanément et de façon complètement symétrique les problèmes primal et dual.

Remarquons qu'il existe une façon simple de démontrer la nécessité de la condition (KKT 3). En effet, on a vu que pour un point admissible ($\in \Im$), $c^T x = b^T y$ est une condition nécessaire et suffisante d'optimalité (nullité du saut de dualité). Or ceci est équivalent à $x^T s = 0$, c'est-à-dire à

$\sum_{i=1}^{n} x_i s_i = 0$. Puisque les variables x et s sont non-négatives, on déduit immédiatement que la seule possibilité de vérifier cette dernière équation est $x_i s_i = 0 \ \forall i$, ce qui est bien la condition (KKT 3). Celle-ci avait également été énoncée dans la liste des propriétés de la dualité.

C.2 DEMARCHE DE CONCEPTION DES METHODES DE POINT INTERIEUR

L'objectif de ce paragraphe est d'établir un cheminement logique aboutissant au schéma suivi par les méthodes de point intérieur actuelles. Il faut bien entendu être conscient que le véritable processus de découverte ne s'est pas fait de façon aussi claire et lucide, l'intuition et l'expérimentation y ayant joué un rôle important. On peut cependant, avec le recul dont on dispose à présent, établir un raisonnement fictif amenant de façon naturelle les concepts-clefs de la théorie des méthodes de point intérieur.

Comme on l'a annoncé plus haut, les méthodes de point intérieur sont des méthodes itératives d'optimisation. Dès lors, on peut tenter d'adapter une méthode itérative d'optimisation classique au cas qui nous intéresse, celui de la programmation linéaire. Parmi ces algorithmes classiques, on trouve l'algorithme du gradient, qu'on applique généralement à l'optimisation sans contraintes et qui consiste simplement à suivre la direction de la plus forte pente.

C.2.1 Optimisation à l'aide de la méthode du gradient

Ainsi, on peut imaginer partir d'un point initial et suivre la direction de la plus forte pente (cette suggestion provient de *[ARB91]*). Pour rappel, il s'agit d'évaluer cette direction à l'endroit de l'itéré courant, puis de faire un pas dans cette direction pour déterminer la valeur de l'itéré suivant..

Cependant, il faut noter trois différences importantes entre la situation où l'on applique généralement une méthode de gradient (optimisation sans contraintes) et le cas de la programmation linéaire

1) Nous sommes dans un cas d'optimisation sous contraintes. Il faut donc tenir compte de Ax = b au moment du calcul de l'itéré suivant pour ne pas aboutir à un point qui ne satisfasse plus ces contraintes.
2) La contrainte $x \geq 0$ doit également être prise en compte.
3) Le gradient de la fonction objectif est constant : il ne dépend pas de x et vaut toujours c. Dans ces conditions, si on suit à la lettre la méthode du gradient sans ternir compte des contraintes, on va forcément évoluer constamment dans la même direction.

Une façon de se débarrasser de la première remarque est de ne pas utiliser la direction du gradient mais celle de sa projection sur le sous-espace orthogonal à celui engendré par les colonnes A, noté Null(A). De cette façon, si l'itéré courant vérifie Ax = b, l'itéré suivant, calculé par $x_{n+1} = x_n + k.d$ où d est la direction projetée du gradient, vérifiera bien encore

$$Ax_{n+1} = A(x_n + kd) = Ax_n + k(Ad) = Ax_n = b$$

(Ad = 0 puisque d appartient à Null(A)). Pour vérifier la contrainte du point 2), celle de non-négativité, il suffit de choisir la longueur du pas k de façon à rester dans les x à composantes positives.

Le troisième problème - la constance de la direction du pas - est plus difficile à résoudre. Avant de s'y attaquer, remarquons encore un autre inconvénient : la direction du gradient est plus ou moins efficace selon la *centralité* (notion assez vague qui sera définie plus précisément plus loin) de l'itéré courant. Par centralité, on entend ici une notion proche de « éloignement par rapport aux contraintes » ou « équidistance par rapport aux faces du polyèdre admissible ».

Ainsi, on remarque sur la Figure C-1 (où l'on a représenté la direction du gradient) que l'itéré (a) progressera bien plus vers la solution en suivant la direction du gradient que l'itéré (b), parce qu'il est plus central.

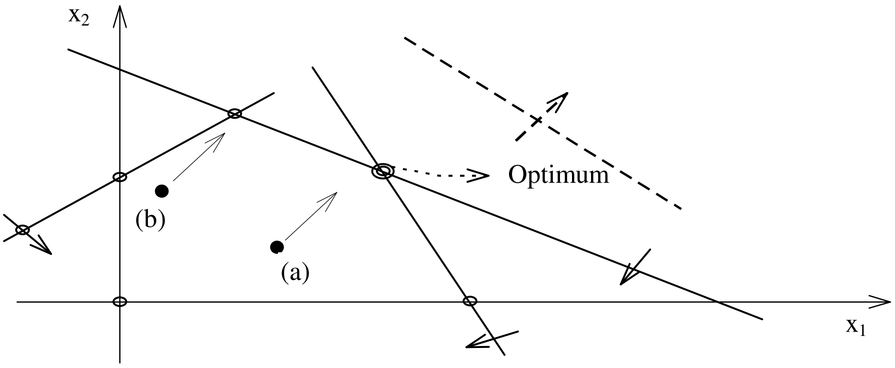


Figure C-1 - Progression potentielle d'un itéré central et d'un itéré peu central

Partant de cette constatation, on a mis au point un stratagème résolvant d'un coup le problème de la constance de la direction du pas et celui de la sensibilité à la centralité : il s'agit de la mise à l'échelle par l'utilisation d'une transformation affine.

C.2.2 Utilisation de la mise à l'échelle affine

Plaçons nous dans la cas simple où l'on cherche à optimiser x_1+x_2 avec pour seules contraintes $x_1 \geq 0$ et $x_2 \geq 0$. Ceci correspond à la Figure C-2 :

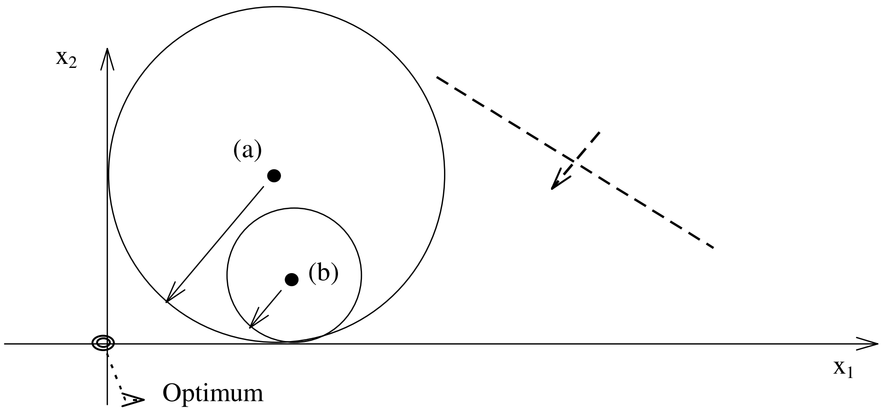


Figure C-2 - Un cas simple d'optimisation

On a porté la direction du gradient sur un cercle centré sur les itérés. Encore une fois, on constate que l'itéré (a) progressera plus du fait de sa position centrale (plus proche de la bissectrice principale).

Imaginons que l'on dilate la figure précédente de façon à faire occuper à l'itéré (b) une position centrale du type de celle de (a). Ceci peut se faire par une simple transformation affine (dilatation selon les deux axes). Dans le nouvel espace transformé, on peut à présent effectuer un pas de gradient beaucoup plus efficace vers la solution.

Si l'on interprète ce pas dans l'espace initial, cela revient à chercher la direction du pas non plus autour d'un cercle mais d'une ellipse « centrée » à l'itéré courant. Cette direction résout correctement nos deux problèmes : en effet, celle-ci évolue bien en fonction de la position de l'itéré courant, et permet de progresser plus rapidement vers la solution lorsque l'itéré est peu central, comme la Figure C-3 l'explicite :

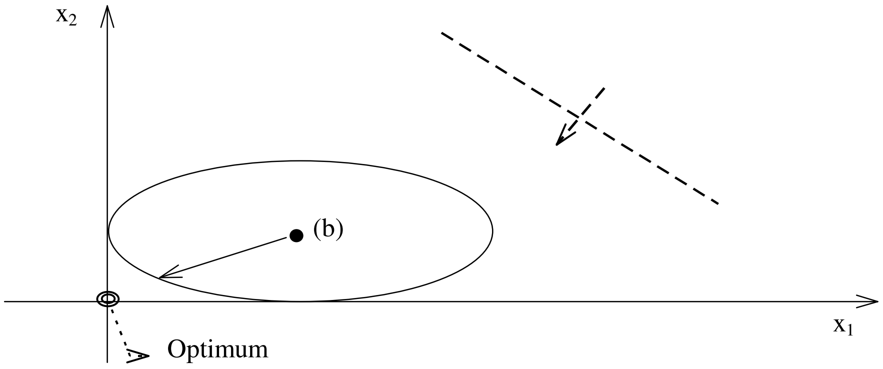


Figure C-3 - Progression avec mise à l'échelle affine

Ceci termine une description sommaire d'une des premières méthodes de point intérieur à avoir été mise en œuvre : il s'agit de l'algorithme de mise à l'échelle affine (*affine scaling*). Cependant, nous n'entamerons pas de plus amples développements dans cette direction - bien qu'elle ait été examinée en son temps par de nombreux chercheurs - parce qu'elle se ramènera à un cas particulier des méthodes que nous allons développer plus loin.

C.2.3 Une autre façon d'optimiser

Une autre façon d'aborder le problème consiste non pas à tenter d'adapter une méthode d'optimisation itérative à la programmation linéaire mais à tenter de résoudre directement les équations formées par les conditions d'optimalité (provenant soit directement des conditions KKT, soit de l'application de la théorie des multiplicateurs de Lagrange).

En effet, ces conditions étant dans le cas qui nous intéresse à la fois nécessaires et suffisantes, la connaissance de leur solution nous donnera directement la solution de notre problème d'optimisation.

Cependant, deux problèmes distincts vont ici aussi nous compliquer la tâche

1) L'équation (KKT 3) est non-linéaire. Toutefois, cette non-linéarité peut être qualifiée de légère, puisqu'il s'agit d'un simple et unique produit de deux variables, alors que les équations (KKT 1) et (KKT 2) sont parfaitement linéaires.
2) Les variables x et s sont soumises à des contraintes de non-négativité (KKT 4).

Si l'on fait momentanément abstraction du problème 2), on peut se tourner vers n'importe quelle technique classique de résolution itérative d'un système d'équations non-linéaires. On pense évidemment en priorité à la méthode de Newton (ou Newton-Raphson).

Pour rappel, soit le problème suivant à résoudre : trouver $\mathbf{x} \in \Re^n$ tel que $\mathbf{f}(\mathbf{x}) = 0$ où $\mathbf{f}$ est une fonction de $\Re^n \to \Re^m$. La méthode de Newton consiste alors à passer de l'itéré $\mathbf{x}_k$ à $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k$ avec $\Delta\mathbf{x}_k = - \mathbf{J}(\mathbf{x}_k)^{-1}\, \mathbf{f}(\mathbf{x}_k)$. Plutôt qu'inverser directement le jacobien $\mathbf{J}$ de la fonction f, on préfère généralement calculer $\Delta\mathbf{x}_k$ comme la solution du système linéaire $\mathbf{J}(\mathbf{x}_k)\, \Delta\mathbf{x}_k = - \mathbf{f}(\mathbf{x}_k)$.

Géométriquement, cette méthode fournit comme itéré suivant le zéro de l'approximation locale du premier ordre (c'est-à-dire l'hyperplan tangent) de la fonction f autour de l'itéré courant. La Figure C-4 explicite la situation dans le cas unidimensionnel.

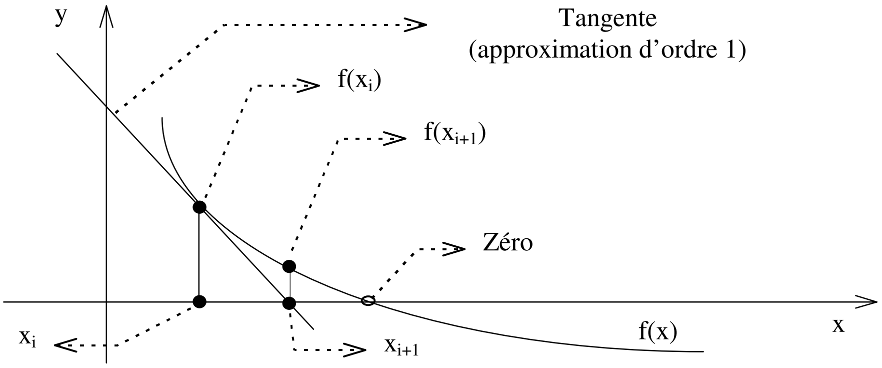


Figure C-4 - Une itération de la méthode de Newton pour une fonction scalaire

Nous pouvons donc appliquer la méthode de Newton à la résolution du système (KKT) si nous trouvons un moyen de satisfaire à l'exigence 2). A cet effet, nous nous tournons vers une autre ressource classique de l'optimisation : l'utilisation d'une fonction barrière (la première utilisation de cette idée en programmation non-linéaire est généralement attribuée à Frisch, en 1955 *[FRI55]*).

C.2.4 Utilisation d'une fonction barrière

Lorsqu'on désire appliquer une méthode d'optimisation sans contrainte à un problème sous contraintes, il existe un moyen de retirer les contraintes, moyennant leur incorporation dans la fonction objectif : ceci se fait à l'aide d'une fonction barrière, répondant à la définition suivante.

Soit une fonction Φ de $\Re \rightarrow \Re$. On appelle Φ une *fonction barrière* si

$$\lim_{x\to 0} \Phi(x) = +\infty$$

Dans ces conditions, on peut remplacer le problème avec contraintes

$$\begin{array}{l} \min f(x) \\ \text{sous } g_i(x) \geq 0 \text{ pour } i = 1,2,\ldots,m \end{array}$$

par le problème non contraint paramétré

$$\begin{array}{l} \min f(x) + \mu \sum_{i=1}^{m} \Phi(g_i(x)) \\ \text{avec } \mu \in \Re^{+} \end{array}$$

En effet, plus on se rapproche de la limite de validité d'une des contraintes, c'est-à-dire plus $g_i(x)$ tend vers zéro, plus le terme $\Phi(g_i(x))$ correspondant augmente (puisqu'il tend vers $+\infty$), ce qui augmente à son tour la valeur du nouvel objectif et va obliger la minimisation à s'éloigner de la zone « dangereuse ».

Néanmoins, il est clair que le x optimum de ce problème n'a aucune raison de coïncider avec l'optimum du problème avec contrainte. En fait, on a défini toute une famille paramétrée de problèmes, chacun d'entre eux possédant son propre optimum, différent du véritable optimum recherché.

Imaginons cependant que l'on résolve une série de problèmes pour une suite de paramètres μ décroissante et tendant vers zéro. On peut dès lors s'attendre à ce que l'optimum des problèmes non contraints tende vers l'optimum du problème avec contraintes, puisqu'on diminue peu à peu l'influence perturbatrice de la fonction barrière. L'avantage que l'on obtient à procéder de cette façon est que cette suite d'optima est constituée de points satisfaisant (strictement) les contraintes.

C'est ce principe qui va servir de complément à la méthode de Newton. Dans notre cas, les fonctions $g_i(x)$ sont simplement les projections des diverses composantes de x (c'est-à-dire $g_i(x) = x_i$). Les pionniers des méthodes de point intérieur ont ensuite choisi une des fonctions barrières les plus courantes : la fonction logarithme, plus précisément : $\Phi(x) = -\ln(x)$. (en effet, on a bien $\lim_{x\to 0^+} -\ln(x) = +\infty$).

C.2.5 Combinaison de la méthode de Newton et de la fonction barrière

Voici nos deux problèmes primal et dual auxquels on a adjoint une fonction barrière, paramétrée par τ.

$$\begin{array}{ll} \min c^T x - \tau \sum_{i=1}^{n} \ln(x_i), & \max b^T y + \tau \sum_{i=1}^{n} \ln(s_i), \\ \text{avec } Ax = b \text{ et } x \in \Re^n & \text{avec } A^T y + s = c \text{ et } y \in \Re^m, s \in \Re^n \end{array}$$

Sous certaines conditions ($\Im_0 \neq \emptyset$), chacun de ces problèmes admet pour tout τ un optimum unique : appelons-les respectivement $x(\tau)$ et $(y(\tau),s(\tau))$. Pour appliquer la méthode de Newton à ces deux problèmes, il faut d'abord déterminer leurs conditions d'optimalité. Comme on peut s'en douter au vu de ce qui précède, ces deux problèmes admettent exactement les mêmes conditions d'optimalité (ce qui signifie qu'ils sont équivalents), à savoir

$$\begin{cases} A^T y + s = c & \text{(KKT 1)} \\ Ax = b & \text{(KKT 2)} \\ x_i s_i = \tau \ \forall i & \text{(KKT 3}\tau\text{)} \\ x > 0 \text{ et } s > 0 & \text{(KKT 4')} \end{cases}$$

Un examen rapide nous apprend que ces conditions sont les conditions KKT des problèmes originaux à deux petites différences près : le membre de droite de l'équation (KKT 3) n'est plus 0, il vaut à présent τ et les inégalités (KKT 4') sont à présent strictes, puisque les fonctions barrières restreignent implicitement les domaines de variation de x et s aux valeurs strictement positives.

On va donc finalement appliquer la méthode de Newton au système non-linéaire formé par (KKT 1), (KKT 2) et (KKT 3τ), ce qui nous donne - après quelques calculs simples - la définition du pas de Newton $(\Delta x,\Delta y,\Delta s)$ suivante

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} . \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \tau e - XSe \end{bmatrix} \qquad \text{(NWT)}$$

Notons qu'on a utilisé certaines notations classiques dans le domaine des méthodes de point intérieur afin de rendre l'expression (NWT) plus compacte, à savoir d'une part *X* en tant qu'abréviation pour *diag(x)*, c'est-à-dire la matrice diagonale telle que $X_{ii} = x_i$ (idem pour S) et d'autre part *e* pour le *vecteur unitaire* ($e_i = 1 \ \forall i$).

Nous sommes à présent très avancés dans le processus de conception des méthodes de point intérieur. Cependant, avant d'esquisser un premier modèle d'algorithme, nous avons besoin d'une dernière notion, qui va nous éclairer sur le rôle du paramètre de la barrière τ : il s'agit du chemin central

C.2.6 Chemin central

Soit le système d'équations suivant

$$\begin{cases} A^T y + s = c \\ Ax = b \\ x_i s_i = \tau \ \forall i \\ x > 0 \text{ et } s > 0 \end{cases}$$

Comme on l'a vu plus haut, ce système forme les conditions d'optimalité des problèmes primal et dual auxquels on a ajouté une fonction barrière. Cependant, il est possible de l'interpréter indépendamment des conditions (KKT), afin d'étudier ses propriétés intrinsèques.

Sous l'hypothèse $\Im_0 \neq \emptyset$, on peut montrer que ce système associe à chaque valeur strictement positive de τ un point unique $(x_\tau,y_\tau,s_\tau) \in \Im_0$. Dans ces conditions, on peut définir de façon univoque une courbe C par $\{ (x_\tau,y_\tau,s_\tau) \mid \tau > 0 \}$. On appelle C le chemin central du couple de problèmes primal et dual. Il possède les propriétés suivantes *[JAN95]*

- Ses équations de définition sont une approximation des conditions d'optimalité (KKT 1-4) d'autant meilleure que τ est proche de 0.
- Le saut de dualité pour le point (x_τ,y_τ,s_τ) vaut $n\tau$. Il tend donc vers 0 lorsque τ tend vers 0.

- Les points de C sont centrés (c'est-à-dire éloignés des contraintes de non-négativité) puisqu'ils dérivent des problèmes munis d'une fonction barrière.
- Le point (x_τ,y_τ,s_τ) converge vers un point (x^*,y^*,s^*) lorsque τ tend vers 0. De plus, ce point est une solution optimale strictement complémentaire du couple de problèmes primal et dual.

On déduit de ces propriétés que les points de C fournissent un chemin le long duquel le saut de dualité décroît régulièrement au fur et à mesure que τ approche de 0, tendant vers une solution optimale tout en restant éloigné des contraintes (centré). C'est cette interprétation des équations qui va expliquer le rôle de τ dans les algorithmes de point intérieur. La Figure C-5 fournit un exemple de chemin central :

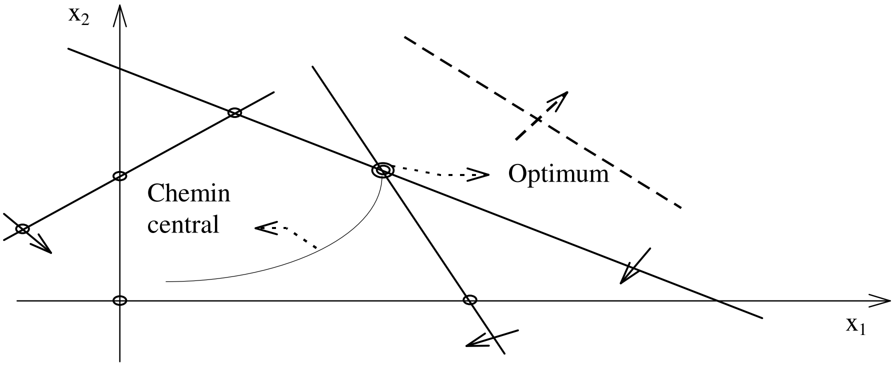


Figure C-5 - Exemple de chemin central à deux dimensions

C.2.7 Modèle d'algorithme de point intérieur

Comme on l'a vu plus haut (p. 19), l'équation de définition du pas de Newton dépend d'un paramètre τ, provenant des problèmes avec fonction barrière. Ce paramètre définissait l'importance relative du terme barrière (destiné à éviter les contraintes) par rapport au véritable objectif à minimiser. On avait d'ailleurs remarqué que l'on pouvait s'attendre à obtenir un optimum de plus en plus proche du véritable minimum au fur et à mesure que τ tend vers 0, tout en restant dans la zone admissible.

Ceci nous conduit naturellement à la méthode de résolution naïve suivante

```
Soit une suite $\tau_1$, $\tau_2$, $\tau_3$, ... décroissante et tendant vers zéro et $(x_0,y_0,s_0)$ un itéré initial
Pour k = 1,2,3,... faire
    ■ Appliquer la méthode de Newton à la résolution des problèmes primal et dual avec le paramètre $\tau_k$ pour le terme barrière. Ceci se fait par une succession de pas définis par (NWT), en partant de la solution du k précédent $(x_{k-1},y_{k-1},s_{k-1})$
    ■ $(x_k,y_k,s_k)$ est la solution fournie par la méthode de Newton
Fin du pour
```

Algorithme 1 - Méthode de point intérieur naïve

Cet algorithme est en fait très semblable à celui conçu par Frisch pour sa première utilisation des fonctions barrières. On peut raisonnablement s'attendre à ce que l'itéré (x_k,y_k,s_k) converge vers l'optimum cherché.

Cependant, il faut se rendre compte que cette méthode implique de nombreux calculs : en particulier, il faut à chaque itération (donc pour chaque k) procéder à une résolution complète par la méthode de Newton, ce qui entraîne le calcul de nombreux pas de Newton. Cet algorithme contient donc en réalité deux boucles imbriquées, ce qui le rend peu intéressant d'un point de vue rapidité de calcul.

Pour pallier cet inconvénient, il faut s'éloigner de l'idée originale de Frisch et introduire la dernière caractéristique des méthodes de point intérieur : on ne va plus chercher à résoudre exactement et complètement les problèmes avec la fonction barrière pour chaque valeur du paramètre τ, mais on va se contenter d'une seule itération de Newton. Cela va donc considérablement accélérer la résolution, puisqu'un seul pas de Newton est dorénavant calculé à chaque itération.

Bien sûr, on ne peut affirmer *a priori* que cette modification va conserver la convergence vers une solution. Une grande partie de la littérature consacrée aux méthodes de point intérieur consiste d'ailleurs à prouver la convergence de diverses versions de cette méthode. Cependant, on peut à présent donner le modèle commun à tous les algorithmes utilisés

```
Soit un itéré initial $(x_0,y_0,s_0)$
Pour k = 0,1,2,... faire
    ■ Choisir une valeur de $\tau_k$
    ■ Résoudre le système (NWT) avec $(x_k,y_k,s_k)$ et $\tau_k$
    ■ Calculer $(x_{k+1},y_{k+1},s_{k+1})$ par $(x_k,y_k,s_k) + \alpha_k (\Delta x_k,\Delta y_k,\Delta s_k)$
Fin du pour
```

Algorithme 2 - Modèle commun aux méthodes de point intérieur

On constate donc qu'on se contente, à chaque itération, d'ajouter à l'itéré courant un incrément proportionnel (terme α_k) au pas fourni par le système (NWT).

On peut s'interroger sur la présence d'un coefficient α_k devant le pas de Newton. En effet, la méthode de Newton classique préconise d'ajouter simplement l'incrément fourni par (NWT). Cependant, il ne faut pas oublier la contrainte de non-négativité (KKT 4) qui, bien que prise en compte sous la forme de fonction barrière qui influence le calcul du pas, ne peut en aucun cas être violée. Or on constate bien souvent qu'un pas de Newton complet amène l'itéré en dehors de la zone valide. Il faut donc modérer l'incrément par un coefficient plus petit que 1 et prendre ce qu'on appelle un pas de Newton partiel garantissant $(x_{k+1},s_{k+1}) > 0$. Le choix de la valeur de α_k sera détaillé plus loin.

Il reste de nombreuses choses à expliciter dans ce modèle, comme par exemple comment trouver (x_0,y_0,s_0), quand arrêter l'algorithme, comment choisir τ_k et α_k, etc. Les réponses à ces questions dépendent du type de méthode de point intérieur que l'on veut étudier, et seront explicitées par la suite.

C.2.8 Signification du paramètre τ en termes de chemin central

Ce paragraphe va tenter de donner une explication intuitive de la signification du paramètre τ, afin de justifier la décision qui a été prise de ne plus effectuer qu'un seul pas de Newton à chaque itération, au lieu d'une optimisation complète.

Supposons tout d'abord que τ soit nul. Dans ce cas, le système que l'on cherche à résoudre par la méthode de Newton se réduit aux véritables conditions d'optimalité des problèmes primal et dual initiaux, sans terme barrière. C'est comme si l'on demandait à la méthode de Newton de résoudre directement le problème initial, sans se préoccuper des contraintes. On peut d'une façon imagée résumer en disant que le pas de Newton vise alors directement la solution optimale, à savoir la limite du chemin central quand τ tend vers 0. Bien sûr, la cible ne sera pas atteinte (puisqu'il faut normalement de nombreux pas de Newton pour atteindre l'objectif) mais l'itéré obtenu en est normalement rapproché.

Par contre, lorsque τ est différent de zéro, le système auquel on applique la méthode de Newton est constitué par les véritables conditions d'optimalité des problèmes primal et dual avec une fonction barrière (paramétrée par τ). Or, on a vu plus haut que ces équations définissent univoquement un point du chemin central, (x_τ,y_τ,s_τ). Toujours de façon imagée, on peut dire que le pas de Newton vise dans ce cas un point du chemin central repéré par la valeur de τ.

On comprend à présent intuitivement la validité de l'algorithme proposé. Partant d'un point initial, on se fixe (par le biais de τ_1) un objectif sur le chemin central et l'on calcule le pas de Newton destiné à atteindre cette cible. La cible ne sera pas non plus atteinte dans ce cas. Dès lors, on se fixe un nouvel objectif, un peu plus proche du véritable objectif (l'extrémité du chemin central), donc un τ

plus proche de zéro, et on recommence la procédure de « visée - pas de Newton - mise à jour de l'objectif ».

Ce procédé peut encore être schématisé par la Figure C-6, qui montre deux itérations successives de l'algorithme

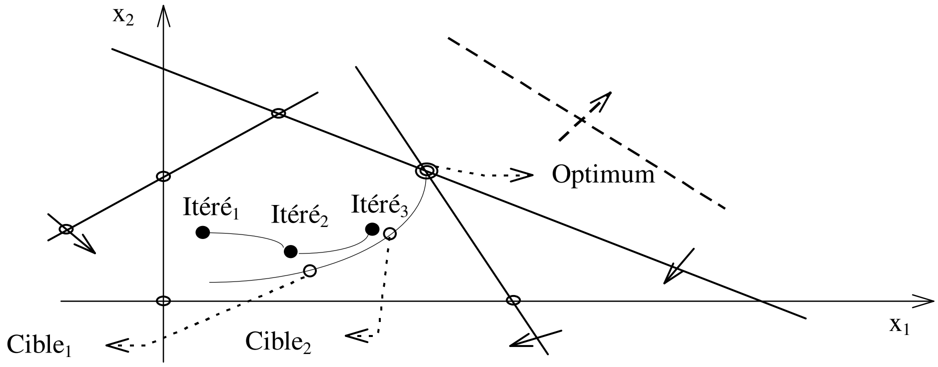


Figure C-6 - Deux pas de Newton vers deux cibles du chemin central

C.2.9 Importance de la centralité

Au vu de l'explication précédente, on peut se demander pourquoi il est nécessaire d'effectuer ce processus de « visée - pas de Newton - mise à jour de l'objectif », plutôt que d'effectuer simplement la série de pas de Newton correspondant à l'application de la méthode normale ayant l'extrémité du chemin central pour objectif.

La raison en est simple : la méthode de Newton n'est pas une méthode d'optimisation sous contraintes et, comme déjà mentionné plus haut, rien ne garantit que le pas de Newton ne nous emmène pas en dehors de la zone admissible (à savoir la zone où $x \geq 0$ et $s \geq 0$). C'est d'ailleurs à cause de cela qu'il a fallu introduire le coefficient α_k dans l'algorithme. On constate alors que l'application directe de la méthode de Newton (avec τ nul) a tendance à diriger les itérés vers le bord de la zone admissible. A cause des coefficients α_k, les pas suivants sont très réduits, ce qui ralentit extrêmement la convergence.

A l'inverse, si l'on se fixe un objectif plus modeste, sous la forme d'un point du chemin central, on évite les bords de la zone admissible, et on s'autorise ainsi des pas de Newton plus longs, d'où une convergence plus rapide. Ces deux situations sont explicitées sur la Figure C-7.

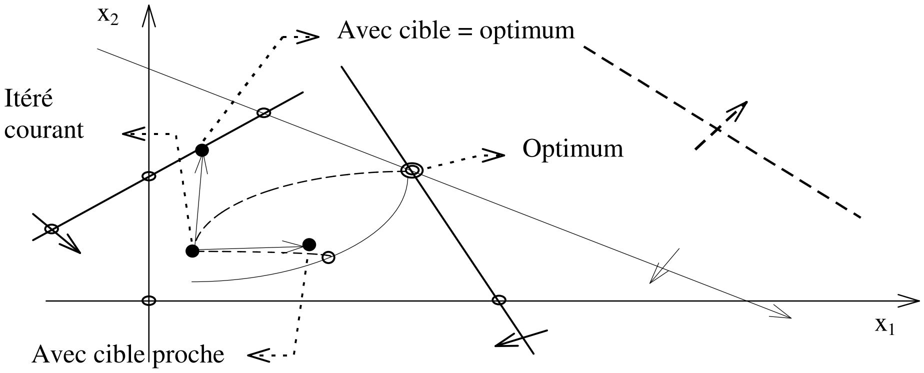


Figure C-7 - Un pas visant l'optimum et un pas visant une cible sur le chemin central

En conclusion, on peut voir le chemin central comme un guide vers la solution optimale, qui permet d'éviter les bords de la zone admissible (c'est-à-dire les contraintes de non-négativité) pour garantir une convergence plus rapide.

C.2.10 Lien avec la mise à l'échelle affine

On a abandonné plus haut l'étude des méthodes de type mise à l'échelle affine après avoir affirmé qu'elles constituaient un cas particulier des méthodes que l'on allait décrire par la suite. En effet, on peut montrer que les formules de mise à jour de l'itéré courant par la méthode de mise à l'échelle affine sont identiques à celles de l'algorithme décrit ci-dessus dans le cas où τ est nul *[ARB91]*. Ceci fournit donc une interprétation géométrique supplémentaire de cet algorithme, du moins dans une de ses versions.

On pressent d'ailleurs que cette variante ne sera pas la plus efficace, au vu des raisons invoquées pratiques ont prouve au paragraphe précédent, ce qui explique son relatif abandon dans le courant actuel des recherches. Bien que sa convergence ait été prouvée, aucune résultat n'est disponible au sujet de sa complexité, et on suspecte celle-ci d'être de type exponentiel *[FRE96,* p. 2*]*.

C.3 CLASSIFICATION DES METHODES DE POINT INTERIEUR

Nous sommes à présent en possession de tous les outils nécessaires à l'établissement d'une classification des méthodes de point intérieur.

C.3.1 Méthodes primales, duales et primales-duales

Une *méthode primale* est une méthode définie, décrite, motivée et implémentée uniquement en termes du problème primal. L'algorithme développé par Karmarkar en 1984 est un exemple de méthode primale. La définition d'une méthode duale s'obtient aisément par symétrie.

Cependant, certaines méthodes primales utilisent le problème dual. Ainsi, il existe des algorithmes qui calculent une *estimation* des variables duales, dont ils se servent pour établir une borne inférieure de la valeur de la fonction objectif (cf. théorie de la dualité). Bien qu'elles fassent référence aux variables duales, ces méthodes ne sont pas à proprement parler des méthodes primales-duales, en raison de la définition suivante.

Une *méthode primale-duale* est une méthode définie, décrite, motivée et implémentée en termes du problème primal et de son dual *de façon symétrique*. Ainsi, on doit non seulement y retrouver à la fois les variable primales (x) et duales (y et s), mais également leur faire subir un traitement totalement symétrique (par exemple, si l'on applique un pas de Newton aux variables primales, il faut le faire également pour les variables duales).

La démarche de conception que nous avons présentée plus haut a en fait abouti à un modèle d'algorithme primal-dual. Ce modèle n'est donc pas le plus général possible, puisqu'il est possible de concevoir des algorithmes purement primals ou duals. Cependant, nous concentrerons notre étude sur les méthodes primales-duales pour les raisons suivantes :

- La théorie des méthodes primales-duales est plus simple et plus élégante que celle des méthodes purement primales ou duales. En fait, comme on l'a constaté tout au long de la présentation de la démarche conceptuelle de création de ces méthodes, l'idée de trst nul

  efficaces que leurs contreparties primales ou duales (en fait, ce sont ces méthodes qui sont compétitives avec l'algorithme du simplexe, voir plus loin)
- Cependant, les méthodes primales restent d'un grand intérêt théorique, et continuent à être étudiées, notamment en raison des idées qu'elles engendrent qui peuvent parfois être transposées au cas primal-dual.

C.3.2 Méthodes à départ admissible et non-admissible

Le schéma général de méthode de point intérieur primale-duale que nous avons présenté plus haut nécessite la connaissance d'un point de départ strictement admissible (appartenant à $\Im_0$). Les méthodes requérant ce type de point de départ sont qualifiées de méthodes à départ admissible (*feasible methods*, adjectif qualifiant également les vecteurs admissibles).

Cependant, il faut noter qu'il existe également des méthodes dites à départ non-admissible (*infeasible methods*), c'est-à-dire qui ne nécessitent pas la connaissance d'un premier itéré strictement admissible. Ces méthodes sont utiles dans deux cas :

- On ne connaît pas un tel point de départ. En fait, pour certains problèmes, le recherche d'un point admissible est presque aussi difficile (et importante) que le calcul de l'optimum.
- Il n'existe pas de point strictement admissible. En effet, il existe des problèmes pour lesquels $\mathfrak{I}_0$ est vide tout en admettant une solution optimale finie. En fait, on peut montrer *[WRI96,* p. 27*]* que cette situation survient uniquement lorsque l'ensemble des points optimaux n'est pas borné, c'est-à-dire lorsqu'il existe une infinité de solutions optimales dont la norme tend vers l'infini. C'est par exemple le cas lorsqu'une variable non bornée supérieurement par les contraintes n'intervient pas dans l'objectif.

Nous parlerons par la suite principalement des méthodes à départ admissible, bien que le paragraphe C-C.5 sera consacré aux différentes façons de se passer d'un tel point de départ.

D'un point de vue historique, il faut noter que pendant un certain temps, les méthodes à départ non-admissible étaient utilisées en pratique et fonctionnaient très bien, sans que l'on ait la moindre justification théorique de leur bon comportement (en effet, on peut transformer une méthode à départ admissible en méthode à départ non-admissible par une modification assez simple). En fait, on ne disposait même pas de la preuve de la convergence de ces algorithmes. Cette lacune a été comblée depuis, et les méthodes à départ non-admissible jouissent à l'heure actuelle de résultats de complexité algorithmique comparables à leurs contreparties à départ admissible.

C.3.3 Trois grandes catégories de méthodes

Enfin, on peut encore classer les méthodes de point intérieur selon le type d'algorithme qu'elles emploient. A cet effet, on distingue trois catégories : les méthodes de mise à l'échelle affine, les méthodes de réduction de potentiel et les méthodes de suivi de chemin. Nous allons à présent passer en revue ces trois méthodes, en détaillant leurs particularités par rapport au schéma général présenté plus haut.

a) Méthodes de mise à l'échelle affine

Comme on l'a déjà mentionné, ces méthodes (*affine scaling methods*), qui possèdent une interprétation géométrique en tant qu'algorithme du gradient dans un espace mis à l'échelle, correspondent au cas où l'on fixe le paramètre τ à zéro pour chaque itération. Nous allons à présent décrire comment s'effectue le choix du coefficient α_k.

Soit (x_k,y_k,s_k) l'itéré courant et $(\Delta x_k,\Delta y_k,\Delta s_k)$ le pas de Newton fourni par le système (NWT). Le nouvel itéré sera calculé par la formule $(x_k,y_k,s_k) + \alpha_k (\Delta x_k,\Delta y_k,\Delta s_k)$.

Il n'est généralement pas possible d'ajouter un pas de Newton complet (c'est-à-dire choisir α_k égal à un) en raison des contraintes de non-négativité $x \geq 0$ et $s \geq 0$. La solution idéale serait évidemment d'effectuer une recherche dans la direction du pas de Newton et de choisir la valeur de α_k qui donnerait le meilleur itéré suivant (c'est-à-dire avec le plus petit saut de dualité). Cependant, cette solution est généralement trop coûteuse en temps de calcul, et on lui préfère souvent la solution suivante : $\alpha_k = \rho\ \alpha_{max,k}$ où

- $\alpha_{max,k}$ est la valeur maximale que peut prendre α_k tout en laissant x_{k+1} et s_{k+1} positifs. Cette valeur peut être déterminée très aisément en quelques opérations arithmétiques élémentaires (par le calcul du minimum des rapports positifs entre les composantes de (x_k,y_k,s_k) et de $-(\Delta x_k,\Delta y_k,\Delta s_k)$, appelé *ratio test*)
- ρ est un coefficient fixe proche de 1 (tel que 0,95 ou 0,99)

Ces méthodes sont quelque peu délaissées à l'heure actuelle, nous n'en reparlerons plus par la suite.

b) Méthodes de suivi de chemin

Les méthodes de suivi de chemin (*path-following methods*) sont articulées autour du chemin central décrit plus haut et se caractérisent par un choix du paramètre τ différent de zéro. Leur principe revient à définir un certain voisinage autour du chemin central, et à faire évoluer les itérés à l'intérieur de ce voisinage tout en progressant vers la solution, comme le montre la Figure C-8.

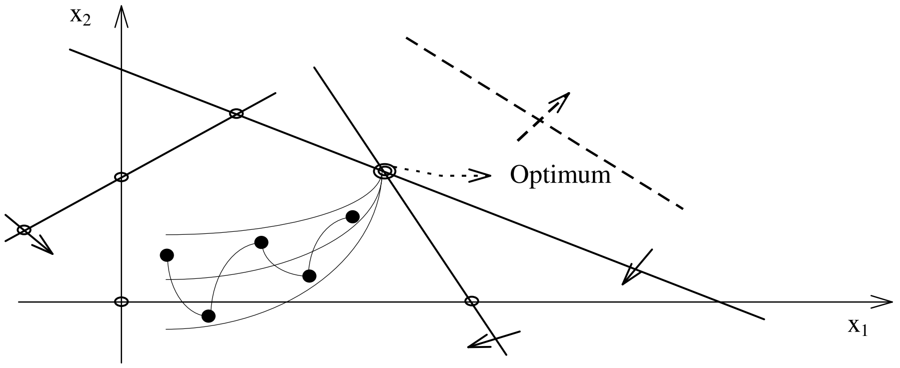


Figure C-8 - Quelques itérés d'une méthode de suivi de chemin

On peut se demander comment ces algorithmes choisissent les valeurs de τ_k. Comme on l'a montré plus haut (p. 21), chaque choix de τ_k revient à choisir une cible appartenant au chemin central. Le principe général est de choisir cette cible en fonction de la position actuelle de l'itéré suivant par rapport au chemin central.

En général, l'itéré courant ne sera pas sur le chemin central, mais dans son voisinage. La caractéristique des points du chemin central, outre leur admissibilité, étant l'égalité des produits x_is_i (on a $x_is_i = \tau$ pour le point repéré par τ), définissons la *mesure de dualité* suivante :

$$\forall(x,y,s)\ \mu(x,y,s) = \frac{\sum_{i=1}^{n} x_i s_i}{n}$$

Cette mesure, qui équivaut au produit moyen des x_is_i, vaut donc τ pour le point du chemin central repéré par τ. Pour les points situés hors de ce chemin, elle fournit une valeur définissant une sorte de projection sur le chemin central (le point le plus proche, dans un certain sens). La dénomination « mesure de dualité » provient du fait, déjà mentionné plus haut, que le saut de dualité de l'itéré (x,y,s), mesurant sa proximité à l'optimum, est égal à $c^Tx - b^Ty = x^Ts = n.\mu(x,y,s)$.

Le principe des méthodes de suivi de chemin sera donc de calculer la mesure de dualité de l'itéré courant, soit μ_k, et à prendre pour cible un τ_k inférieur (plus proche de l'optimum) en appliquant un coefficient de proportionnalité σ_k, selon la formule $\tau_k = \sigma_k\ \mu_k$. Le choix de σ_k dépend de la méthode considérée.

Il existe de nombreux types de méthodes de suivi de chemin, nous en détaillerons trois au cours du chapitre suivant : les méthodes dites à pas court, les méthodes prédiction-correction (à pas alterné) et les méthodes à pas long. Il existe également plusieurs définitions possibles du voisinage du chemin central. Les deux versions les plus intéressantes sont celles du voisinage $N_2(\theta)$ et du voisinage $N_{-\infty}(\gamma)$ (avec θ et γ compris entre 0 et 1)

$$N_2(\theta)=\left\{(x,y,s)\in\Im_0 \,\middle|\, \|XSe-\mu e\|_2 \le \theta\mu\right\} \quad N_{-\infty}(\gamma)=\left\{(x,y,s)\in\Im_0 \,\middle|\, x_is_i \ge \gamma\mu\ \forall i\right\}$$

(0,5 et 10^{-3} sont des valeurs typiques respectivement pour les paramètres θ et γ).

Le voisinage $N_2(\theta)$ exprime que la distance Euclidienne entre $(x_1s_1, x_2s_2, ..., x_ns_n)$ et $(\mu, \mu, ..., \mu)$ ne peut excéder un certain pourcentage de μ (où, rappelons-le, μ est la valeur moyenne de ces produits x_is_i). Quant à $N_{-\infty}(\gamma)$, il exprime simplement que chaque produit x_is_i ne peut être inférieur à un certain pourcentage de leur valeur moyenne μ.

Le voisinage $N_{-\infty}(\gamma)$ est assez lâche :)

.

En gardant tous leurs itérés dans l'un de ces voisinages, les méthodes de suivi de chemin veulent garder leurs itérés centraux, et tentent de réduire tous les produits x_is_i à zéro *à la même vitesse*, c'est-à-dire plus ou moins en même temps.

c) Méthodes de réduction de potentiel

Les méthodes de réduction de potentiel (*potential reduction methods*) prennent des pas de Newton de la même forme que ceux des méthodes de suivi de chemin. Cependant, ces méthodes ne suivent pas explicitement le chemin central, et se justifient indépendamment de lui. Elles utilisent une *fonction potentiel* logarithmique, que nous appellerons Φ (à ne pas confondre avec une fonction barrière) répondant aux deux conditions suivantes

$$\Phi \to +\infty \text{ si } \exists i \mid x_i s_i \to 0 \text{ mais } \mu \not\to 0$$

$$\Phi \to -\infty \iff (x, y, s) \to \Omega \text{ où } \Omega \text{ est l'ensemble des solutions optimales}$$

Ces conditions signifient que

- d'une part, la fonction potentiel tend vers -∞ si et seulement si l'itéré tend vers une solution optimale
- d'autre part, la fonction potentiel tend vers +∞ lorsque l'un des produits $x_i s_i$ tend vers zéro sans que la mesure de dualité tende également vers cette valeur, c'est-à-dire lorsqu'on se rapproche des contraintes de non-négativité (un x_i ou un s_i tend vers 0) sans se rapprocher simultanément de l'optimalité (μ ne tendant pas vers 0).

L'objectif de ces algorithmes sera donc de faire tendre Φ vers -∞ (d'où leur dénomination), la première condition assurant la convergence vers un optimum tandis que la seconde permet de s'éloigner des contraintes et rester central. Comme dans le cas des méthodes de suivi de chemin, on utilise la mesure de dualité μ, et il faut choisir un σ_k et un α_k à chaque itération, le choix des paramètres dépendant de la méthode (et de la fonction potentiel Φ) choisie. Nous décrirons au chapitre suivant une de ces méthodes.

C.4 DESCRIPTION DETAILLEE DE QUATRE METHODES

Nous allons à présent décrire plus en détail quatre méthodes primales-duales à départ admissible.

C.4.1 Méthode de suivi de chemin à pas court

Le principe de cette méthode consiste à choisir un voisinage $N_2(\theta)$ pour une valeur de θ donnée, et de fixer les paramètres σ_k et α_k à deux valeurs constantes : $\sigma_k = \sigma$ et $\alpha_k = 1$ $\forall k$. Un choix judicieux du couple (θ,σ) permet de démontrer que les itérés successifs seront tous contenus dans le voisinage choisi et que la mesure de dualité μ_k converge linéairement vers zéro (à la vitesse constante $1-\sigma$). A titre d'exemple, la convergence a ainsi été prouvée pour les valeurs *[Wri96, p. 86]*

$$\theta = 0{,}4 \text{ et } \sigma = 1 - \frac{0{,}4}{\sqrt{n}}.$$

Voici la description complète de la méthode

```
Soit $0 < \theta < 1$, $0 < \sigma < 1$ et un itéré initial $(x_0,y_0,s_0) \in N_2(\theta)$
Pour $k = 0,1,2,...$ faire
    ■ Calculer $\mu_k$
    ■ Résoudre le système (NWT) avec $(x_k,y_k,s_k)$ et $\tau_k = \sigma \mu_k$ pour obtenir $(\Delta x_k,\Delta y_k,\Delta s_k)$
    ■ $(x_{k+1},y_{k+1},s_{k+1}) \leftarrow (x_k,y_k,s_k) + (\Delta x_k,\Delta y_k,\Delta s_k)$
Fin du pour
```

Algorithme 3 - Méthode de suivi de chemin à pas court

On qualifie cette méthode de suivi de chemin à pas court à cause du choix de σ relativement conservateur. En effet, les valeurs choisies sont souvent très proches de 1 à cause du caractère restrictif du voisinage N_2. Cela implique que le τ_k visé sera très proche du μ_k courant : on demande à la méthode de Newton d'atteindre un point du chemin central relativement proche de l'itéré courant, ce qui entraîne le calcul d'un pas relativement court et une progression assez lente vers la solution (l'extrémité du chemin). Notons que l'on prend ici un pas de Newton complet à chaque itération ($\alpha_k = 1$).

On peut visualiser les itérés fournis par cette méthode à l'aide de la Figure C-9, dans le cas où n=2

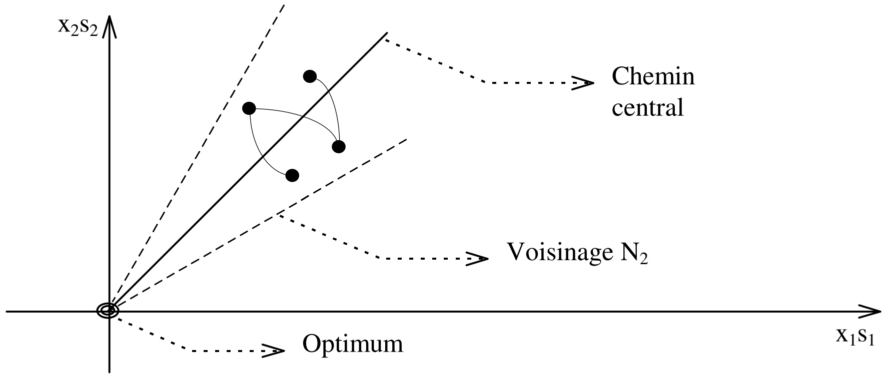


Figure C-9 - Méthode de suivi de chemin à pas court en axes xs

Cette représentation un peu particulière diffère de la représentation classique de l'espace des itérés. En effet, on a ici représenté les produits x_1s_1 et x_2s_2 le long des axes, ce qui entraîne les remarques suivantes

- La solution optimale se trouve à $x_1s_1 = x_2s_2 = 0$, c'est-à-dire à l'origine
- Le chemin central, caractérisé par des produits x_1s_1 et x_2s_2 égaux, correspond à la bissectrice principale
- On peut montrer que les limites du voisinage $N_2(\theta)$ correspondent à deux droites passant par l'origine, symétriques par rapport à la bissectrice, d'autant plus proches des axes que θ est proche de 1.
- Cette représentation simplifie donc la représentation des itérés, son seul inconvénient étant sa non-linéarité : elle déforme l'espace original de telle façon que la direction du pas de Newton n'est plus représentée par une droite mais par une courbe.

C.4.2 Méthode de suivi de chemin à prédiction-correction

La méthode de suivi de chemin à pas court avait pour caractéristique un σ constant, compris entre 0 et 1. Ce choix σ assurait deux objectifs simultanés : rester central (en visant une cible sur le chemin central) et se rapprocher de l'optimum (en réduisant la mesure de dualité). La méthode à prédiction-correction va également chercher à atteindre ces deux objectifs, mais à l'aide de deux types d'itérations différentes.

Avant de décrire ces deux itérations, nous allons examiner les conséquences de deux choix particuliers pour le paramètre σ : $\sigma = 0$ et $\sigma = 1$.

a) Cas $\sigma = 0$

Dans ces conditions, on a $\tau_k = 0$ à chaque itération. On a déjà rencontré cette situation : c'est celle de la mise à l'échelle affine. On cherche ici uniquement à *réduire la mesure de dualité* μ (la faire tendre vers 0) pour atteindre l'optimum, sans se préoccuper de la centralité des itérés. L'inconvénient d'une méthode exclusivement basée sur ce choix de σ est que l'on arrive très vite aux bords de la zone admissible, ce qui réduit ensuite la taille des pas de Newton que l'on peut effectuer.

b) Cas $\sigma = 1$

Cette situation est à l'opposé de la précédente : on vise ici un point du chemin central repéré par un τ_k égal à μ_k, la mesure de dualité de l'itéré courant. Cela signifie donc que l'on ne cherche pas à progresser vers l'optimum (on vise un saut de dualité identique à celui que l'on a déjà), mais plutôt à se *recentrer*, c'est-à-dire à se rapprocher du chemin central (plus précisément, du point repéré par μ_k). Une méthode basée exclusivement sur ce choix de σ n'a bien sûr aucune chance de converger vers un optimum.

c) Principe de la prédiction correction

On aura sans doute compris que les deux types d'itération que propose la méthode de suivi de chemin à prédiction-correction correspondent exactement aux deux choix de σ présentés plus haut.

Cette méthode comporte un second ingrédient caractéristique : elle se base sur deux voisinages emboîtés de type N_2 : le voisinage extérieur N_2^{ext} et le voisinage intérieur N_2^{int}.

Le déroulement de l'algorithme est alors le suivant :

- Itérations de prédiction paires (k = 0,2,4,6,...) : partant de l'itéré courant situé à l'intérieur de N_2^{int}, on effectue un pas de type $\sigma = 0$. Ce pas va naturellement s'éloigner du chemin central. On choisira une valeur de α_k la plus grande possible de façon à rester dans le voisinage N_2^{ext} (pas de Newton partiel)
- Itérations impaires (k = 1,3,5,...) de correction : partant de l'itéré courant situé à l'intérieur de N_2^{ext}, on effectue un pas de recentrage ($\sigma = 1$) qui nous ramène à l'intérieur de N_2^{int}. On peut dans ce cas toujours prendre α_k égal à un.

Ce déroulement est représenté sur la Figure C-10 (toujours en axes x_is_i)

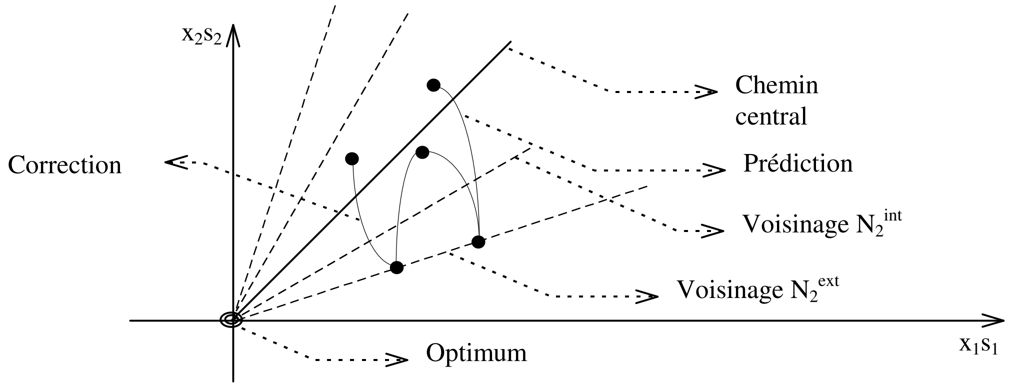


Figure C-10 - Méthode de suivi de chemin à prédiction-correction en axes xs

Un choix judicieux des voisinages permet de garantir la validité des pas de correction, à savoir le retour à l'intérieur de N_2^{int}. A titre d'exemple, la convergence a été prouvée pour les voisinages $N_2^{ext}(0{,}5)$ et $N_2^{int}(0{,}25)$ *[WRI96*, p. 91*]*. Voici la description complète de l'algorithme

```
Soit $0 < \theta^{int} < \theta^{ext} < 1$ et un itéré initial $(x_0,y_0,s_0) \in N_2(\theta^{int})$
Pour k = 0,1,2,... faire
■ Si k est pair
    • Résoudre le système (NWT) avec $(x_k,y_k,s_k)$ et $\tau_k = 0$ pour obtenir $(\Delta x_k,\Delta y_k,\Delta s_k)$
    • Calculer le $\alpha_k$ maximum tel que $(x_k,y_k,s_k) + \alpha_k(\Delta x_k,\Delta y_k,\Delta s_k) \in N_2(\theta^{ext})$
    • $(x_{k+1},y_{k+1},s_{k+1}) \leftarrow (x_k,y_k,s_k) + \alpha_k(\Delta x_k,\Delta y_k,\Delta s_k)$
■ Si k est impair
    • Calculer $\mu_k$
    • Résoudre le système (NWT) avec $(x_k,y_k,s_k)$ et $\tau_k = \mu_k$ pour obtenir $(\Delta x_k,\Delta_{yk},\Delta s_k)$
    • $(x_{k+1},y_{k+1},s_{k+1}) \leftarrow (x_k,y_k,s_k) + (\Delta x_k,\Delta y_k,\Delta s_k$
Fin du pour
```

Algorithme 4 - Méthode de suivi de chemin à prédiction-correction

La dénomination prédiction-correction provient du domaine des algorithmes de résolution des équations différentielles : on y trouve également un pas de prédiction (qui se meut le long de la tangente à la trajectoire recherchée) et un pas de correction (qui ramène l'itéré prédit plus près de la trajectoire recherchée).

Ce type de méthode est une amélioration certaine par rapport aux méthodes à pas court ou de mise à l'échelle affine : en effet, le recentrage qui survient à l'itération de correction permet des pas de Newton beaucoup plus longs lors de l'itération de prédiction suivante. Cependant, cet algorithme souffre encore du fait que les voisinages N_2 sont assez restrictifs, en particulier au début de l'algorithme, lorsqu'on est encore assez éloigné de l'optimum. On constate ensuite une amélioration progressive au fur et à mesure de l'avancement de la résolution, les pas de prédiction devenant de plus en plus longs. On peut en fait montrer que la convergence de la mesure de dualité est superlinéaire en phase finale de l'algorithme (voir le paragraphe E.3 des compléments).

C.4.3 Méthode de suivi de chemin à pas long

Le principe de cette méthode est encore une fois de confiner les itérés dans un voisinage du chemin central, défini cette fois-ci par $N_{-\infty}(\gamma)$. Cependant, on va cette fois s'autoriser un choix moins conservateur du paramètre σ_k : on le prendra à l'intérieur de l'intervalle $[\sigma_{min}, \sigma_{max}]$. Le paramètre α_k sera quant à lui le plus grand possible tout en conservant l'itéré à l'intérieur du voisinage $N_{-\infty}(\gamma)$ (ce qui entraîne donc que l'on prendra la plupart du temps un pas de Newton partiel, à l'inverse de la méthode à pas court).

Le comportement typique de ces méthodes est illustré sur la Figure C-11.

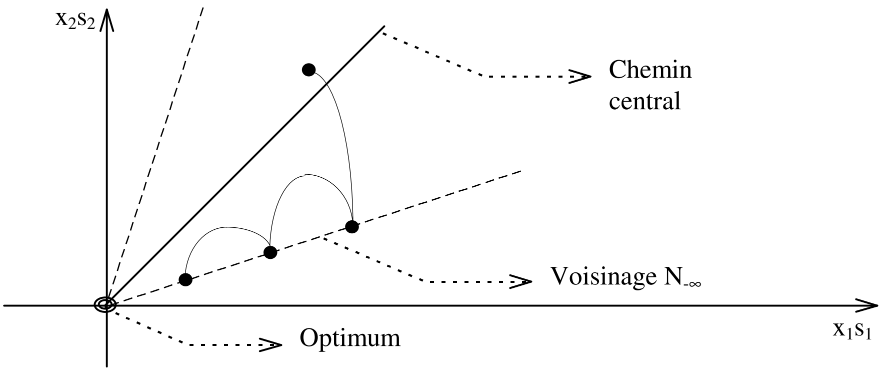


Figure C-11 - Méthode de suivi de chemin à pas long en axes xs

On comprend**8**

alors aisément le rôle des deux bornes de l'intervalle $[\sigma_{min}, \sigma_{max}]$:

- σ_{min} garantit un certain recentrage : le pas de Newton va donc commencer par s'éloigner de la frontière du voisinage. Cependant, au fur et à mesure que l'on avance dans cette direction, on commence à s'écarter à nouveau du chemin central. Cela est dû au fait que les équations de Newton sont basées sur une approximation linéaire d'équations en réalité non-linéaires. Plus on s'avance dans la direction préconisée, plus l'approximation linéaire s'éloigne de la réalité et plus on s'éloigne à nouveau du point central visé.
- σ_{max} quant à lui garantit une certaine progression minimale vers la solution à chaque itération. En effet, on a déjà vu que des valeurs élevées de σ (proches de 1) entraînent de petits pas de Newton et une convergence assez lente.

Voici la description complète de l'algorithme

```
Soit $0 < \gamma < 1$, $0 < \sigma_{min} < \sigma_{max} < 1$ et un itéré initial $(x_0,y_0,s_0) \in N_{-\infty}(\gamma)$
**Pour** $k = 0,1,2,...$ **faire**
    ■ Choisir $\sigma_k \in [\sigma_{min}\ \sigma_{max}]$
    ■ Résoudre le système (NWT) avec $(x_k,y_k,s_k)$ et $\tau_k = \sigma_k \mu_k$ pour obtenir $(\Delta x_k,\Delta y_k,\Delta s_k)$
    ■ Calculer le $\alpha_k$ maximum tel que $(x_k,y_k,s_k) + \alpha_k (\Delta x_k,\Delta y_k,\Delta s_k) \in N_{-\infty}(\gamma)$
    ■ $(x_{k+1},y_{k+1},s_{k+1}) \leftarrow (x_k,y_k,s_k) + \alpha_k (\Delta x_k,\Delta y_k,\Delta s_k)$
**Fin du pour**
```

Algorithme 5 - Méthode de suivi de chemin à pas long

Le choix exact de σ_k à l'intérieur de l'intervalle dépend alors de l'itéré courant, et se fait généralement de façon heuristique (un exemple sera donné plus loin). On démontre alors la convergence et la polynomialité de ces méthodes en fonction des valeurs de σ_{min}, σ_{max} et γ. Celles-ci sont en général plus efficaces que les méthodes à pas court, à cause du choix plus agressif du paramètre σ_k.

Notons encore que les limites du voisinage $N_{-\infty}(\gamma)$ dans la représentation en axes $x_i s_i$ sont également deux droites passant par l'origine, symétriques par rapport à la bissectrice, d'autant plus proches des axes que γ est proche de zéro. Il y a donc équivalence entre les deux types de voisinage N_2 et $N_{-\infty}$ pour le cas à deux dimensions, mais on peut montrer que cette équivalence n'est plus valable pour les dimensions supérieures ($n \geq 3$).

C.4.4 Méthode de réduction de potentiel

A l'inverse des trois algorithmes précédents, cette méthode ne se base pas sur un ou plusieurs voisinages du chemin central, mais sur la minimisation d'un fonction potentiel. Nous allons utiliser la fonction potentiel symétrique primale-duale de Tanabe, Todd et Ye *[TAN87, TOD90]* (celle-ci vérifie les propriétés énoncées plus haut, p. 25) :

$$\Phi_\rho = \rho \ln x^T s - \sum_{i=1}^{n} \ln x_i s_i$$

L'algorithme que nous allons choisir est celui de Kojima, Mizuno et Yoshise *[KOJ91]*. Etant donné un paramètre fixé $\rho > n$, on fixe σ_k égal à n / ρ afin de calculer le pas de Newton par (NWT). La fonction potentiel Φ_ρ est alors utilisée pour le choix du paramètre α_k : on va prendre la valeur qui minimise Φ_ρ selon

$$\alpha_k = \arg \min_{\alpha \in (0,\alpha_{\max,k})} \Phi_\rho(x_k + \alpha \Delta x_k, s_k + \alpha \Delta s_k)$$

Une analyse théorique permet alors de montrer que la fonction potentiel Φ_ρ sera réduite d'au moins une certaine quantité constante à chaque itération (0,15 pour être précis), ce qui entraîne la convergence globale de l'algorithme en vertu des propriétés de Φ_ρ. Voici la description complète de l'algorithme

```
Soit $\rho > n$ et un itéré initial $(x_0,y_0,s_0) \in \Im_0$
Pour $k = 0,1,2,...$ faire
    ■ Calculer $\mu_k$
    ■ Résoudre le système (NWT) avec $(x_k,y_k,s_k)$ et $\tau_k = n \mu_k / \rho$ pour obtenir $(\Delta x_k,\Delta y_k,\Delta s_k)$
    ■ Calculer $\alpha_{max,k}$ et choisir $\alpha_k$ minimisant $\Phi_\rho(x_k + \alpha\Delta x_k, s_k + \alpha\Delta s_k)$ sur $(0, \alpha_{max,k})$
    ■ $(x_{k+1},y_{k+1},s_{k+1}) \leftarrow (x_k,y_k,s_k) + \alpha_k (\Delta x_k,\Delta y_k,\Delta s_k)$
Fin du pour
```

Algorithme 6 - Méthode de réduction de potentiel

D'un point de vue pratique, il est assez difficile d'obtenir la valeur exacte de α_k puisque cela requiert la minimisation d'une fonction hautement non-linéaire. On dispose alors des alternatives suivantes :

- Utiliser un minimum approché de Φ_ρ, que l'on trouve par exemple avec une méthode de recherche linéaire (*line search*). Cependant, la valeur de α_k résultante est souvent trop petite pour donner en pratique une convergence rapide. On se tourne alors vers les deux autres possibilités
- Utiliser une approximation quadratique de Φ_ρ le long de la direction du pas de Newton. On peut montrer que cette approximation présente toujours un minimum compris entre 0 et $\alpha_{max,k}$, calculable aisément, et l'on choisit alors un α_k égal à ce minimum
- Une troisième possibilité est de prendre le plus long pas possible, c'est-à-dire $\alpha_k = 0,95\ \alpha_{max,k}$ ou $\alpha_k = 0,99\ \alpha_{max,k}$.

- La meilleure solution consiste toutefois à choisir une heuristique réalisant un compromis entre théorie et pratique. Ainsi, on peut par exemple essayer quelques grandes valeurs pour α_k, et effectuer le pas correspondant s'il réalise bien la décroissance escomptée de Φ_p. Dans le cas contraire, on prend alors le minimum de l'approximation quadratique ou même le minimum approché trouvé par recherche linéaire.

Il faut toutefois noter que la convergence ou la polynomialité de la méthode ne sont plus garanties de façon théorique si l'on choisit une de ces solutions.

C.5 SE PASSER D'UN POINT DE DEPART ADMISSIBLE

Comme on a pu le constater, les algorithmes décrits plus haut nécessitent tous la connaissance d'un itéré initial strictement admissible. Lorsqu'on ne dispose pas d'une telle information, on a à sa disposition trois possibilités : transformer le problème pour rendre le point de départ admissible, modifier la méthode utilisée ou opter pour la technique du problème homogène auto-dual.

C.5.1 Rendre le point de départ admissible dans un nouveau problème

Cette technique est dérivée de celle que l'on utilise dans des conditions similaires avec l'algorithme du simplexe (méthode connue sous le nom de *big M*). Nous allons ici la présenter dans le cas primal.

Soit un point de départ quelconque x_0 tel que $x_0 > 0$. Calculons l'inadmissibilité (l'erreur résiduelle) de ce point de départ, c'est-à-dire la différence entre Ax_0 et b :

$$r_b = Ax_0 - b$$

Il suffit alors de créer un nouveau problème, comportant une variable et une colonne de plus, et possédant un point de départ admissible connu, soit

$A' = [A \mid r_b]$, $x_0' = (x_0,\ -1)$, $c' = (c,\ M)$, le problème primal devenant $\begin{array}{l} \min c'^T x', \quad x' \in \Re^{n+1} \\ \text{avec } A'x' = b \text{ et } x' \geq 0 \end{array}$

On vérifie aisément que le point x_0' vérifie la contrainte $A'\,x_0' = b$, et peut donc convenir comme point de départ strictement admissible. La constante M sera choisie de façon à être très largement supérieure à tous les autres coefficients du problème : ainsi, on peut s'attendre à ce que l'algorithme tente d'annuler en priorité la variable supplémentaire que l'on a ajoutée x_{n+1}. S'il s'avère que cette dernière est toujours significativement différente de zéro à la fin de l'optimisation, on peut en conclure que le problème initial est impossible.

Cependant, il faut noter que cette façon de procéder n'est pas sans présenter quelques inconvénients majeurs : outre l'instabilité numérique que peut provoquer l'introduction d'un coefficient M largement plus grand que tous les autres, on a ajouté à la matrice A une colonne entièrement remplie de valeurs non-nulles, autrement dit une colonne dense. Comme on le verra dans la partie consacrée à l'implémentation des méthodes, il s'agit d'un inconvénient majeur dans le cas où le problème de départ est creux.

C.5.2 Modifier l'algorithme

L'alternative que nous allons présenter ici est spécifique aux méthodes de point intérieur. Cependant, son principe est assez général, ce qui permet de l'appliquer à la plupart des méthodes à départ admissible connues.

Il s'agit en fait de confier à la méthode de Newton le soin de réduire l'inadmissibilité. Un examen rapide de la définition du pas de Newton nous amène à réécrire le système (NWT) sous la forme

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} . \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ \tau e - XSe \end{bmatrix} \qquad \text{(NWT - INF)}$$

où r_b et r_c sont les erreurs résiduelles de primal et du dual, définies par

$$r_b = Ax - b \qquad r_c = A^T y + s - c$$

La seule différence par rapport à (NWT) est le remplacement des zéros du terme de droite par $-r_c$ et $-r_b$. Cette nouvelle définition donne toujours un pas de Newton visant le point (x_τ, y_τ, s_τ) du chemin central. Cependant, il essaie à présent également d'enlever l'inadmissibilité de l'itéré courant. Comme les contraintes sont linéaires, il est facile de voir qu'il suffit d'un pas de Newton complet (c'est-à-dire une situation où $\alpha_k = 1$ dans les algorithmes décrits plus haut) pour rendre l'itéré suivant admissible d'un seul coup (les résidus r_b et r_c devenant nuls), les itérés suivants continuant bien entendu à vérifier les contraintes.

Cette situation idéale (pas de Newton complet) ne se produit malheureusement que très rarement dans les cas pratiques, ce qui nous oblige à ajouter un garde-fou supplémentaire dans nos algorithmes : puisque la méthode de Newton a désormais deux objectifs différents à atteindre (l'optimalité et l'admissibilité), il faut veiller à ce que le progrès vers ces deux objectifs se fasse plus ou moins simultanément, sous peine d'arriver en fin de résolution à une solution « optimale » qui violerait encore les contraintes !

La façon la plus courante d'imposer cette progression harmonieuse vers les deux objectifs est de redéfinir la notion de voisinage du chemin central pour incorporer une mesure d'admissibilité. Ainsi, mentionnons à titre d'exemple une des possibilités d'extension du voisinage $N_{-\infty}(\gamma)$, valable dans le cadre de la méthode de suivi de chemin à pas long :

$$N_{-\infty}(\gamma,\beta) = \left\{ (x,y,s) \mid x > 0,\ s > 0,\ x_i s_i \geq \gamma\mu\ \forall i,\ \left\| (r_b, r_c) \right\| \leq I_0 \beta\mu \right\}$$

β est un paramètre complémentaire supérieur ou égal à un qui va permettre de quantifier la rapidité avec laquelle on veut réduire l'inadmissibilité (comparativement à la réduction du saut de dualité μ) et I_0 est une mesure de l'inadmissibilité initiale définie par

$$\frac{\left\| (r_{b,0}, r_{c,0}) \right\|}{\mu_0}.$$

Intuitivement, cette définition impose aux points du voisinage $N_{-\infty}(\gamma,\beta)$ d'avoir la norme de leurs résidus bornée par un multiple de la mesure de dualité μ. Ainsi, à mesure que l'on progresse vers l'optimalité ($\mu=0$), les itérés seront contraints de réduire leur inadmissibilité de façon au moins proportionnelle à la réduction de μ.

Bien sûr, la démonstration de la convergence de l'algorithme initial n'est plus valable dans ce cas. On remarque d'ailleurs que les démonstrations de convergence des méthodes à départ non-admissible présentent généralement plus de difficultés techniques que celles de leurs équivalents à départ admissible.

C.5.3 Utiliser la technique du problème homogène auto-dual.

Cette technique, d'une grande élégance mathématique, sera présentée dans les compléments. Tout comme la méthode du *big M*, elle consiste à incorporer l'inadmissibilité dans un problème plus grand qui possède en sus la propriété d'auto-dualité, ce qui lui confère entre autres l'avantage de posséder un point de départ simple et déterminé qui appartient au chemin central.

C.6 COMPLEXITE

Au cours de l'introduction, nous avons simplement mentionné que la majorité des méthodes de point intérieur étaient de complexité polynomiale (nous emploierons ici le mot complexité au sens complexité de pire cas). Nous allons donner ici quelques informations complémentaires sur les meilleurs résultats de complexité disponibles actuellement.

Il existe de nombreuses définitions possibles (et couramment utilisées) de la complexité algorithmique d'une méthode de résolution *[GAR79]*. Celles-ci se résument toutes à donner une borne du *temps d'exécution* de l'algorithme sous la forme d'une fonction de la *taille* des données, les définitions de *temps d'exécution* et *taille* dépendant de la définition considérée. Par exemple, lorsque cette fonction est un polynôme, on parle de complexité polynomiale.

C.6.1 Complexité de pire cas ou complexité moyenne.

Il est tout d'abord possible de s'intéresser au pire cas ou au cas moyen. Dans le premier cas, on cherchera à évaluer le temps maximal que peut mettre un algorithme, c'est-à-dire lorsque les données lui sont très défavorables. Dans le second cas, on ne s'intéresse qu'au temps moyen d'exécution. Notons que cela nécessite une idée *a priori* de la distribution des données du problème, et que la complexité résultante peut dépendre de cette distribution. Nous adopterons l'optique du pire cas par la suite, la plus couramment utilisée, parce que le calcul de la complexité moyenne est difficile à réaliser de façon analytique (on a alors plutôt recours à des mesures réalisées sur ordinateur, donnant une complexité empirique).

Gardons toutefois à l'esprit que c'est la complexité moyenne qui détermine les performances réelles de l'algorithme en pratique. Ainsi, la méthode du simplexe exhibe en pratique un comportement moyen de type polynomial *[BOR87]*, l'optimisation nécessitant rarement plus de n+m itérations, bien qu'étant de pire cas exponentiel (l'exemple de V. Klee et G. Minty nécessite 2^n itérations *[KLE72]*).

C.6.2 Complexité en arithmétique rationnelle

Avec ce type de complexité (aussi appelé *bit complexity*, complexité en bits), on modélise le temps d'exécution par le nombre d'opérations arithmétiques élémentaires nécessaires à l'accomplissement de l'algorithme (de type additions et multiplications). Si l'on suppose les donnés rationnelles, tous les nombres manipulés par l'algorithme seront eux aussi rationnels (à condition de se limiter aux opération élémentaires classiques, c'est-à-dire de ne pas utiliser de racine carrée ou de fonction transcendante). La plupart des analyses de complexité font appel à ce modèle.

Notons encore que toutes les méthodes de point intérieur sont basées sur le calcul d'un pas de Newton à chaque itération, opération prenant à elle seule la quasi-totalité du temps d'exécution. Si l'on désire comparer différentes méthodes, il suffit donc de comparer le nombre d'itérations nécessaires, le nombre d'opérations par itération étant grosso modo constant (de l'ordre de n^3).

C.6.3 Taille du problème

On peut simplement mesurer la taille du problème par n, la dimension du vecteur à optimiser ou m, le nombre de contraintes. Cependant, on peut aussi faire intervenir le nombre de bits nécessaires au stockage des données, à savoir la matrice A et les vecteurs b et c dans le cas de la programmation linéaire. Dans le modèle rationnel, on peut supposer les éléments de A, b et c entiers. Chaque entier k occupe un nombre de bits égal au plus petit entier supérieur à $\log_2(k) + 1$. Etant donné que A, b et c comportent respectivement m x n, m et n entiers, on définit la taille du problème par

$$L = L_0 + (mn + m + n) \quad \text{avec} \quad L_0 = \sum_{k \in A,b,c} \lceil \log_2 k \rceil + 1$$

où L_0 est la somme des tailles (en bits) des entiers de A, b et c et le terme (mn + m + n) correspond au nombre de bits nécessaires pour séparer ces entiers. L'emploi de la quantité L est quasiment universel dans la littérature consacrée aux méthodes de point intérieur.

C.6.4 Fin de l'algorithme

Les méthodes de point intérieur étant de type itératif, on n'atteint jamais exactement la solution optimale, ce qui entraîne donc théoriquement un temps d'exécution (et un nombre d'opérations) infini. Cependant, il existe dans le cas rationnel une procédure permettant de trouver une solution en un nombre fini d'itérations (celle-ci se base sur le fait que les composantes non-nulles d'une solution optimale sont forcément supérieures à 2^{-L} où L est la taille des données définies plus haut), ce qui permet bien de définir une complexité pour ce modèle.

C.6.5 Résultats

Les meilleures méthodes de point intérieur présentent une complexité arithmétique de pire cas égale à

$$n^{3,5} \log L \text{ opérations arithmétiques élémentaires} \qquad \text{ou} \qquad \sqrt{n} \log L \text{ itérations.}$$

On trouve parmi celles-ci des méthodes de réduction de potentiel et des méthodes de suivi de chemin (on ne sait toujours pas si les méthodes de mise à l'échelle affine sont polynomiales).

Cette complexité peut encore être réduite à n^3 log L à condition d'utiliser des méthodes avancées de résolution du système (NWT), basées sur la mise à jour de la solution de l'itération précédente. Toutefois, l'effort actuel des chercheurs tend plutôt à diminuer la complexité en nombre d'itérations plutôt que la complexité d'une itération. Notons enfin que les meilleures méthodes de type à départ non-admissible disposent également d'une complexité de $n^{½}$ log L itérations.

C.6.6 Analyse critique

Tout d'abord, notons que ces résultats sont uniquement valables pour les algorithmes théoriques. Comme on l'a vu plus haut, certaines méthodes sont inapplicables en pratique, et l'on a alors recours à diverses modifications pour les rendre implémentables. Ces méthodes modifiées n'ont alors plus aucune garantie théorique de convergence ou de polynomialité.

Il existe encore certaines incohérences entre complexité théorique et mesures pratiques du nombre d'itérations. D'une part, les meilleurs méthodes en pratique ne sont pas celles qui bénéficient de la meilleure complexité théorique : ainsi, les méthodes à pas courts ($n^{½}$ log L itérations) sont généralement bien plus lentes que les méthodes à pas long, pourtant de complexité moins bonne (n log L itérations).

D'autre part, les mesures pratiques du nombre d'itérations *[GON94]* révèlent plutôt une complexité de l'ordre de log n plutôt que $n^{½}$! Aucune explication théorique à ce comportement n'est disponible à ce jour. Remarquons simplement que, tout comme dans le cas du simplexe, il peut s'agir d'une différence entre complexité de pire cas et complexité moyenne.

Enfin, tous ces résultats dépendent de L, la taille du problème, et non de n uniquement, ce qui peut être considéré comme gênant. Un algorithme dont la complexité serait un polynôme fonction de n uniquement (sans référence à L) est appelé fortement polynomial, et la question de son existence dans le cas du problème de programmation linéaire n'a pas encore été résolue.

C.6.7 Complexité en arithmétique réelle

Dans ce cas, on considère des opérations sur l'ensemble des nombres réels et plus seulement des nombres rationnels. Dans ce type de modèle (que l'on appelle également complexité algébrique), les méthodes de point intérieur ne sont plus de complexité polynomiale. En effet, il n'existe plus de procédure permettant de terminer à coup sûr l'algorithme en un nombre fini d'itérations, parce que la solution peut avoir des composantes réelles arbitrairement petites (et non plus bornées par 2^{-L} comme dans le cas rationnel). On ne sait d'ailleurs pas encore s'il existe des méthodes polynomiales pour la programmation linéaire dans ce type de modèle.

On s'intéresse alors à un problème légèrement différent, à savoir trouver une solution approchée. Le critère d'approximation employé est la mesure de dualité : on parle de solution ε-approchée si on a un itéré admissible vérifiant $\mu \le \varepsilon$. Dans ce cas, les meilleures méthodes pour trouver une solution ε-approchée au problème de programmation linéaire disposent d'une complexité de l'ordre de

$$\sqrt{n}\log\frac{1}{\varepsilon} \text{ itérations}$$

En fait, on a constaté empiriquement que dans la plupart des cas, on peut passer de la complexité d'une méthode pour la résolution exacte en nombres rationnels à celle pour la résolution approchée en nombres réels en remplaçant simplement L par log(1/ε). On se contente le plus souvent de prouver la complexité pour un seul des deux modèles (rationnel et réel).

C.7 GENERALISATION PAR NESTEROV ET NEMIROVSKY

Peu après la découverte des premières méthodes de point intérieur pour la programmation linéaire, Nesterov et Nemirovsky ont tenté de trouver une justification théorique au bon comportement (c'est-à-dire à la complexité polynomiale) de ces algorithmes *[NES94]*.

Comme on l'a vu au cours des chapitres précédents, ces méthodes reposent essentiellement sur deux éléments :

- l'utilisation de la méthode de Newton pour résoudre les conditions d'optimalité
- l'utilisation d'une fonction barrière logarithmique pour garder les itérés centrés (loin des contraintes)

Reconnaissant le caractère incontournable de la méthode de Newton, ils se sont plus particulièrement intéressés à la fonction barrière, en se posant la question suivante :

> Quelles sont les propriétés de la fonction barrière responsables du comportement polynomial des méthodes ?

En se plaçant dans le cadre très général de l'optimisation sous contraintes convexes (dont la programmation linéaire est un cas particulier), leur travail remarquable arrive à la surprenante conclusion suivante : seulement deux propriétés relativement simples de la fonction barrière F(x) sont responsables du comportement polynomial, à savoir (∇ est l'opérateur différentiel *nabla* classique)

$$\left|\nabla^3 F(x)[h,h,h]\right| \leq 2\left(h^T \nabla^2 F(x) h\right)^{\frac{3}{2}} \quad \text{et} \qquad \left|\nabla F(x)^T h\right| \leq \sqrt{\nu}\left(h^T \nabla^2 F(x) h\right)^{\frac{1}{2}}$$

Une fonction barrière vérifiant ces deux propriétés est dite ν-auto-concordante (où ν est appelé le paramètre de la barrière). En termes quelque peu techniques, la première inégalité est équivalente à la continuité de Lipschitz du hessien de F par rapport à la métrique euclidienne locale induite par le hessien lui-même, tandis que la seconde exprime la continuité de Lipschitz de la barrière elle-même par rapport à cette même métrique euclidienne.

Nesterov et Nemirovsky ont donc montré que la méthode de Newton, associée à une barrière ν-auto-concordante, permet d'augmenter la précision d'au moins t chiffres en $O(\nu^{½} t)$ itérations, ce qui entraîne le caractère polynomial de la méthode. On remarquera au passage qu'une barrière sera d'autant plus efficace qu'elle aura un petit paramètre ν.

Le travail de Nesterov et Nemirovsky ne s'arrête pas là : ces deux auteurs nous livrent une formule générale fournissant une barrière auto-concordante valable pour n'importe quel problème convexe (cette formule est malheureusement inapplicable en pratique, parce qu'elle fournit une fonction barrière dont l'évaluation est trop coûteuse en temps de calcul), développent ensuite une sorte de *calcul des barrières auto-concordantes* (permettant de combiner, projeter, transformer des barrières en gardant l'auto-concordance) et fournissent explicitement une telle barrière dans de nombreux cas particuliers de problèmes convexes.

Dans le cas linéaire, on vérifie aisément que la fonction barrière logarithmique classique

$$-\sum_{i=1}^{n} \ln x_i$$

est une barrière n-auto-concordante pour le domaine des vecteurs à n composantes positives ($\Re^n_+$).

Pour conclure, signalons une des applications les plus importantes de cette théorie : dans le cas de l'optimisation sur les matrices semidéfinies positives (cf. seconde partie), il existe une barrière auto-concordante d'expression particulièrement simple : le logarithme du déterminant. Cela va en fait entraîner la possibilité de construire des méthodes de points intérieur polynomiales pour résoudre ce type de problèmes.

D. ASPECTS RELATIFS A L'IMPLEMENTATION

Nous allons à présent discuter de quelques aspects relatifs à l'implémentation sur ordinateur des méthodes de point intérieurs et à leur efficacité en pratique. Nous nous occuperons plus particulièrement du cas des méthodes primales-duales à départ inadmissible, les plus efficaces et les plus utilisés en pratique.

D.1 RESOLUTION DU SYSTEME D'EQUATIONS LINEAIRES

Comme on l'a déjà souligné plus haut, l'étape la plus importante (au point de vue temps de calcul) d'une itération est la résolution du système (NWT - INF). L'efficacité de l'implémentation dépendra donc grandement de la façon dont ce calcul est réalisé.

D.1.1 Importance de la densité de la matrice A

Il faut tout d'abord souligner que la grande majorité des problèmes que l'on résout en pratique ont une matrice A possédant une faible densité d'éléments non-nuls (typiquement inférieure à 10 %). On qualifie une telle matrice de matrice creuse. La Figure D-1 présente un exemple typique d'une telle matrice, issue d'un problème réel (on a représenté les éléments non-nuls par des carrés noirs) :

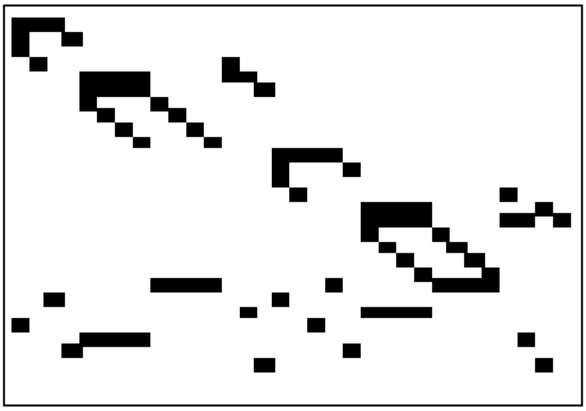


Figure D-1 - Densité d'une matrice typique d'un problème réel

Il est évident que l'on n'a pas intérêt à réserver m.n places en mémoire pour stocker une telle matrice, puisque la plupart de celles-ci seraient occupées par la valeur zéro. On utilise plutôt généralement des techniques basées sur un ensemble de couples (*position*, *valeur*) où *position* est un couple d'indices (i,j) et *valeur* représente l'élément a_{ij} de la matrice A.

D.1.2 Trois possibilités de résolution

S'il est important de tenir compte de la densité de A au moment du stockage, il est encore plus essentiel de s'en servir au moment du calcul. En effet, on constate que le membre de gauche du système est constitué principalement de la matrice A (creuse) et des matrices S et X (diagonales, donc très creuses), ce qui entraîne qu'il sera également très creux.

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} . \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ \tau e - XSe \end{bmatrix} \qquad \text{(NWT - INF)}$$

Nous avons donc besoin d'une méthode efficace de résolution d'un système linéaire creux. On peut également choisir de ne pas résoudre directement le système (NWT - INF), mais de tenter de réduire sa taille. En effet, il est équivalent au système suivant (après la sortie de Δs, on a également posé $D = S^{-½}X^{½}$)

$$\begin{cases} \begin{bmatrix} -D^{-2} & A^T \\ A & 0 \end{bmatrix} \cdot \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} -r_c - X^{-1}(\tau e - XSe) \\ -r_b \end{bmatrix} \\ \Delta s = -X^{-1}(S\Delta x - (\tau e - XSe)) \end{cases} \quad \text{(AUG)}$$

Ce système porte le nom de système augmenté. On peut encore expliciter Δx, pour obtenir la forme la plus compacte possible

$$\begin{cases} \Delta x = -S^{-1}(X\Delta s - (\tau e - XSe)) \\ AD^2A^T\Delta y = -r_b + A(S^{-1}Xr_c - S^{-1}(\tau e - XSe)) \\ \Delta s = -r_c - A^T\Delta y \end{cases} \quad \text{(NORM)}$$

Ce système est connu sous le nom d'équations normales (l'équation de définition de Δy rappelle la forme d'une équation normale dans un problème de régression linéaire au sens des moindre carrés). La seule étape difficile consiste à calculer Δy : en effet, il vient ensuite Δs puis Δx par simple substitution.

D.1.3 Choix de la méthode de résolution

On dispose en général de deux types de méthodes pour résoudre un système linéaire de grande taille : les méthodes directes de factorisation et les méthodes itératives. Chacun de ces deux types peut a priori être combiné avec n'importe lequel des trois systèmes (NWT - INF), (AUG) et (NORM). Cependant, on constate que certaines méthodes sont mieux adaptées à certains systèmes dans certains cas. On rencontre en pratique trois approches différentes : résolution de (NORM) par factorisation de Cholevsky, résolution de (AUG) par factorisation de Bunch-Partlett et méthode itérative appliquée à (AUG) ou (NORM)

a) Méthodes itératives

La méthode itérative la plus couramment utilisée pour résoudre un système linéaire est celle du gradient conjugué. Employée telle quelle, cette méthode converge assez lentement. On la remplace avantageusement par la méthode du gradient conjugué avec préconditionnement. Cette seconde méthode utilise une matrice carrée P, appelée préconditionneur, destinée à accélérer la convergence de l'algorithme (on résout PAx = Pb au lieu de Ax = b).

Cependant, la tâche de trouver un bon préconditionneur est peu aisée, et dépend fortement du type de problème traité. Cette méthode est donc peu compétitive face aux méthodes de factorisation directe, à l'exception de quelques cas particuliers où l'on dispose d'un bon préconditionneur facile et rapide à calculer (cas des problèmes de transport, par exemple *[MIT96]*)

b) Méthodes de factorisation directe

Les méthodes de factorisation directes du système Ax = b visent à exprimer A sous la forme d'un produit de deux matrices plus simples G et H. En effet, résoudre Ax = b revient alors à résoudre successivement Gy = b puis Hx = y. Pour peu que la forme des matrices G et H se prête à une résolution rapide de ces deux systèmes (si ce sont des matrices triangulaires, par exemple, cela peut se faire par une simple série de substitutions), on dispose alors d'une méthode pour calculer x. En général, l'étape de factorisation (trouver G et H) est beaucoup plus lente que la résolution des deux systèmes qui en découlent.

c) Pourquoi la méthode de Gauss n'est elle pas applicable dans ce cas ?

Il est impensable d'appliquer une simple méthode de Gauss à un des systèmes (NWT - INF), (AUG) ou (NORM). En effet, le principe de cette méthode - le pivotage - a pour inconvénient de détruire le caractère creux de la matrice au fur et à mesure de son application (à chaque choix de pivot, on détruit dans *chacune des lignes restantes* les éléments nuls des colonnes correspondant aux éléments non-nuls de la ligne du pivot). L'espace mémoire nécessaire et le temps pour effectuer le pivotage deviennent alors très vite prohibitifs.

D.1.4 Cholevsky et les équations normales

Dans (NORM), l'équation de définition de Δy possède une propriété intéressante, parce qu'elle est du type $M\Delta y = r$ où $M = AD^2A^T$ est une matrice symétrique définie positive. Dans ce cas, on sait qu'il existe une décomposition de M sous la forme $M = L^TL$, où L est une matrice triangulaire supérieure. Cette décomposition se détermine très aisément à l'aide de l'algorithme de Cholevsky. La détermination de Δy se fait ensuite très rapidement à l'aide de deux séries de substitutions (correspondant aux résolutions des systèmes triangulaires $L^T u = r$ et $L\Delta y = u$).

a) Phénomène de remplissage

On constate que, malgré une matrice M relativement creuse, il arrive assez souvent que le facteur de Cholevsky L soit assez dense, ce qui ralentit la résolution et augmente l'espace mémoire nécessaire. Ce phénomène s'appelle le remplissage, et se remarque aisément sur le schéma suivant. Il s'agit d'une matrice dont les éléments non-nuls sont représentés par des carrés noirs. Sa partie triangulaire supérieure est celle d'une matrice M définie positive (partie au dessus de la diagonale), tandis que la partie triangulaire inférieure est celle de son facteur de Cholevsky. On constate que la densité de ce dernier est très supérieure à celle de M sur la Figure D-2 :

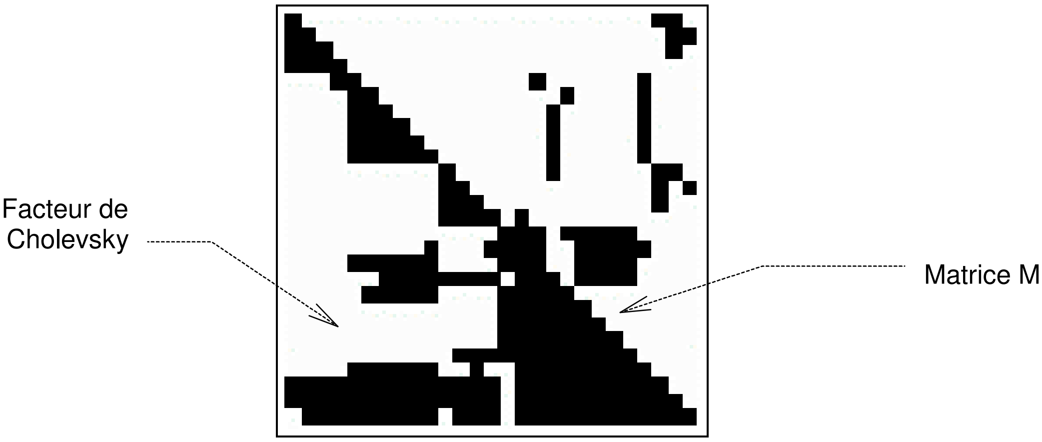


Figure D-2 - Densité de M et de son facteur de Cholevsky

Afin de pallier cet inconvénient, il existe une technique d'ordonnancement des colonnes. En effet, il est possible de permuter les lignes et les colonnes de M de façon à obtenir un facteur de Cholevsky moins dense. Formellement, on définit P, une matrice de permutation, et on cherche alors à factoriser PMP^T.

Le problème de la détermination de la meilleure matrice P pour une matrice M donnée (c'est-à-dire celle qui donne le facteur de Cholevsky *permuté* le moins dense possible) est NP-complet, et donc impossible à résoudre exactement de façon efficace. On utilise alors des heuristiques d'ordonnancement.

Les deux heuristiques les plus employées sont connues sous les noms d'ordonnancement de degré minimum et d'ordonnancement à remplissage local minimal *[AND96]*. Cette seconde méthode est plus lente que la première, mais donne généralement de meilleurs résultats.

Remarquons encore que les positions des éléments non-nuls de la matrice M (égale à AD^2A^T) ne changent pas d'une itération à une autre, ce qui permet l'utilisation de la même matrice P pour toutes les itérations. Il est donc raisonnable de consacrer initialement un peu plus de temps à la recherche d'une bonne permutation, puisque cela profitera par la suite à chaque itération.

A titre d'exemple, la Figure D-3 montre l'effet de l'application de ces deux heuristiques

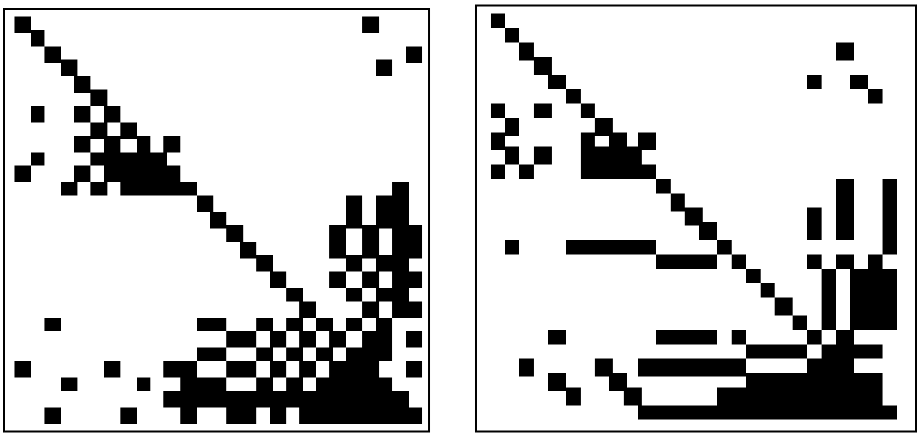


Figure D-3 - Factorisations *minimum ordering* et *minimum local fill in*

Il faut noter que le développement d'implémentations efficaces de la méthode de Cholesky est un domaine de recherche très actif actuellement (nouvelles heuristiques d'ordonnancement, concept de supernoeuds, implémentations parallèles, etc.) *[WRI96,* pp. 212-216*]*

b) Problème des colonnes denses

Un inconvénient de la factorisation de Cholevsky appliquée au système (NORM) est sa grande sensibilité à une colonne dense (c'est-à-dire une colonne comportant très peu d'éléments nuls). Etant donné la structure de M (semblable à celle de AA^T), il suffit d'une seule colonne dense dans A pour rendre la matrice AA^T presque pleine.

En effet, supposons que la colonne k soit dense (par exemple 10% d'éléments nuls). Puisque le $(i,j)^{\text{ème}}$ élément de AA^T est constitué du produit scalaire de la $i^{\text{ème}}$ ligne de A avec la $j^{\text{ème}}$ ligne de A (la $j^{\text{ème}}$ colonne de A^T), il comprendra forcément le terme $a_{ik}.a_{jk}$. Comme seuls 1% des produits $a_{ik}.a_{jk}$ seront nuls, la colonne k remplit à elle seule 99% de la matrice AA^T ! Même une colonne relativement peu dense (50 % d'éléments nuls) est à elle seule responsable de la non-nullité d'un quart de $AA^{T.}$

Il existe néanmoins une façon de traiter les colonnes denses si celles-ci ne sont pas trop nombreuses, en utilisant la formule de Sherman-Morrison-Woodbury. Soit une matrice H égale à I + J. Supposons que I soit facile à inverser, et que J soit de rang faible. La formule de Sherman-Morrison-Woodbury donne une façon de calculer facilement H^{-1} en fonction de I^{-1} et de J. A titre d'exemple, dans le cas où J est de la forme pq^T (donc de rang 1), on a la relation suivante

$$H^{-1} = I^{-1} - \frac{I^{-1}pq^TI^{-1}}{1+q^TI^{-1}p}$$

Appliquons cette formule à notre problème de colonne dense. Soit la matrice $M = AD^2A^T$. On peut isoler l'effet de la colonne dense $A_{.k}$ (cf. plus haut) dans une matrice J égale à $d_k^2\ A_{.k}A_{.k}^T$ (où d_k^2 vaut en fait x_k/s_k). On a donc M = I + J où I est creuse et J de rang faible. On peut alors appliquer la formule précédente avec $p = q = d_k\ A_{.k}$. Puisque l'on cherche à résoudre $M\Delta y = r$, on a

$$\Delta y = M^{-1}r = I^{-1}r - \frac{I^{-1}pq^TI^{-1}}{1+q^TI^{-1}p}r = I^{-1}r - I^{-1}p\frac{q^T(I^{-1}r)}{1+q^T(I^{-1}p)}$$

On constate qu'il suffit alors de calculer $I^{-1}r$ et $I^{-1}p$ pour déterminer Δy. On va donc effectuer une factorisation Cholevsky de la matrice creuse I suivie de deux phases de substitution, appliquées successivement à r puis à q. On a donc réussi à éliminer l'effet désastreux de remplissage d'une colonne dense au prix d'une phase de substitution supplémentaire (dont le coût en temps de calcul est généralement très inférieur au coût de la factorisation en elle-même).

Ce principe se généralise facilement au cas où plusieurs colonnes denses sont présentes, mais perd de son intérêt au fur et à mesure que leur nombre augmente.

D.1.5 Bunch-Partlett et le système augmenté

La matrice de système (AUG) n'est pas définie positive, on ne peut donc pas lui appliquer la factorisation de Cholevsky. On pourrait se tourner vers un type de factorisation plus général (de type LU, par exemple), mais il existe une méthode s'appliquant à notre cas particulier : l'algorithme de Bunch-Partlett. Celui-ci part donc de la matrice

$$M = \begin{bmatrix} -D^{-2} & A^T \\ A & 0 \end{bmatrix},$$

symétrique mais pas définie positive, et cherche une factorisation correspondant à $PMP^T = LEL^T$ où P est une matrice de permutation, L est une matrice triangulaire inférieure et E une matrice quasi diagonale : sa diagonale est constituée uniquement de blocs 1x1 ou 2x2 symétriques. L'algorithme de Bunch-Partlett donne une méthode efficace pour réaliser cette factorisation. Cependant, la matrice P dépend ici de la valeur de D et doit être recalculée à chaque itération. On ne peut donc plus la déterminer au départ une fois pour toutes, comme dans le cas de la factorisation de Cholevsky. Cette méthode a été assez peu employée dans les logiciels jusqu'à présent, probablement à cause de l'absence de librairies implémentant efficacement l'algorithme de Bunch-Partlett. Elle dispose néanmoins de certains atouts par rapport à sa concurrente, mentionnés dans le paragraphe suivant.

D.1.6 Comparaison des trois méthodes

Pour les raisons invoquées plus haut, les méthodes itératives ne sont pas compétitives dans le cadre d'une implémentation traitant des problèmes généraux. Le **Tableau 1** résume les différences entre les deux méthodes de factorisation directe :

	Cholevsky	**Bunch-Partlett**
Colonnes denses	Effet désastreux, correction possible avec Sherman-Morrison-Woodbury	Peu d'effets
Stabilité numérique	Bonne	Excellente
Traitement naturel des variables libres	Non (nécessité d'agir au niveau de la formulation du problème)	Possible (remplacer D_i par 0)
Possibilité de déterminer P une fois pour toutes	Oui	Non
Disponibilité d'implémentations efficaces	Grande	Faible
Temps de calcul	Dépend de la taille et de la densité de la matrice	150 à 200 % de celui de Cholevsky

Tableau 1 - Comparaison des factorisations de Cholevsky et Bunch-Partlett

Comme on peut le constater, les avantages de Cholevsky se situent principalement au niveau de la rapidité, Bunch-Partlett semblant plus approprié lorsque le problème comporte des colonnes denses, beaucoup de variables libres ou présente des instabilités numériques. L'idéal serait de disposer dans le même logiciel du choix entre les deux possibilités en fonction du problème (choix laissé à l'utilisateur ou résultant d'une analyse automatique des caractéristiques du problème) mais cette possibilité n'a pas encore été implémentée en pratique.

D.2 FORMULATION DU MODELE

D.2.1 Analyse initiale du problème

Dans de nombreux cas, le programme linéaire à résoudre n'est pas disponible dans la forme standard : il faut d'abord le convertir en ajoutant les contraintes et variables nécessaires. Une fois cette opération effectuée, il n'est pas rare que l'on puisse réduire la taille du problème (de la matrice A) à l'aide d'une série de règles assez simples. Voici quelques unes des règles les plus couramment rencontrées *[AND96, LUS94]*

- Les lignes et colonnes vides sont soit enlevées (pour cause de redondance) soit utilisées pour déterminer que le problème est impossible
- Les contraintes d'égalité à une seule variable permettent de fixer cette variable et de la sortir du problème. Les contraintes d'inégalité à une seule variable se transforment en borne inférieure ou supérieure sur cette variable
- Pour chaque contrainte, on peut déterminer une borne inférieure et une borne supérieure, selon

  $$b_i^- = \sum_{a_{ij}<0} a_{ij} u_j \text{ et } b_i^+ = \sum_{a_{ij}>0} a_{ij} u_j$$

  (u_j représente la borne supérieure, éventuellement infinie, de la variable x_j). On a alors

  $$b_i^- \leq \sum_j a_{ij} x_j \leq b_i^+$$

  La comparaison de l'intervalle $[b_i^- \ b_i^+]$ avec b_i permet soit d'améliorer les bornes, soit de déclarer la contrainte redondante, soit de détecter un problème impossible selon le type de la contrainte considérée
- Une contrainte d'égalité avec un coefficient positif et uniquement des variables non-négatives peut être utilisée pour éliminer la variable en question par pivotage
- Une contrainte d'égalité avec exactement deux variables permet d'éliminer une des deux variables par pivotage
- Deux lignes multiples l'une de l'autre sont soit redondantes, soit inconsistantes
- De nombreuses autres règles existent, chaque logiciel employant les variantes propres à sa formulation

Lorsqu'il est devenu impossible d'appliquer ces règles, on peut passer à l'optimisation proprement dite. Cependant, il est bon de noter que l'hypothèse de rang maximal pour A, que l'on avait acceptée au début du volet théorique , n'est pas forcément satisfaite.

La seule façon de vérifier le rang de A (et d'éliminer les contraintes redondantes le cas échéant) est de procéder à une élimination de Gauss sur A. Cependant, comme on l'a mentionné plus haut (p. 37), cette procédure est impraticable. Il est néanmoins possible d'appliquer un algorithme de pivotage généralisé de Markowitz, qui détecte la plupart des dépendances linéaires pour un coût en temps de calcul raisonnable *[MAR57]*.

Le danger d'une matrice A de rang non maximal est le risque d'instabilité numérique. Il faut toutefois modérer cet inconvénient : même dans le cas d'une matrice A de rang maximal, la matrice des équations normales AD^2A^T devient singulière lorsqu'on approche de la solution, et peut entraîner de toutes façons des problèmes de stabilité numérique.

D.2.2 Traitement des variables libres

a) Décomposition

Le traitement standard des variables libres, tel qu'appliqué dans l'algorithme du simplexe, consiste à remplacer la variable libre x par la différence $x^+ - x^-$ avec $x^+ \geq 0$ et $x^- \geq 0$. Outre le fait qu'elle augmente la taille du problème, cette façon de faire présente plusieurs inconvénient dans le cas des méthodes de point intérieur :

- A toute solution du problème original où $x = k$, il existe une infinité de solutions du problème modifié, avec, dans le cas $k > 0$ par exemple, $x_+ = \lambda + k$ et $x_- = \lambda$.
- Par conséquent, l'ensemble des solutions optimales n'est pas borné, ce qui entraîne (voir plus haut, p. 23) $\Im_0 = \emptyset$, c'est-à-dire l'inexistence de point strictement admissible. On est alors forcé d'utiliser une méthode à départ non-admissible.
- On constate en pratique que les méthodes de points intérieur tendent à faire croître simultanément x^+ et x^- vers $+\infty$, tout en gardant leur différence constante, ce qui peut amener

des problèmes numériques. Une solution empirique donnant des résultats satisfaisants a été apportée à ce problème : il s'agit de normaliser le couple (x^+, x^-) après chaque itération (en leur retranchant une même quantité), de façon à les empêcher de diverger.

b) Pivotage

Il existe une seconde possibilité : utiliser une des contraintes d'égalité où la variable libre apparaît pour l'exprimer en fonction des autres variables et la remplacer dans les autres contraintes. Cette procédure, connue sous le nom de pivotage, présente néanmoins le grave inconvénient de rendre la matrice des contraintes de plus en plus dense, ce qui a un effet désastreux sur les performances des méthodes de point intérieur. On la réserve généralement pour le traitement d'un petit nombre de variables libres.

c) Reformulation

Il existe enfin une troisième possibilité, propre aux méthodes de point intérieur : modifier la formulation pour tenir compte des variables libres. On peut dans ce cas réécrire les conditions KKT et le système définissant le pas de Newton qui en découle. L'inconvénient de cette reformulation est qu'elle empêche d'arriver à une forme du type des équations normales, présentant une matrice définie positive *[WRI96]*. La factorisation de Cholevsky est alors impraticable, et il faut s'en remettre à l'algorithme de Bunch-Partlett, plus lent et plus difficile à mettre en oeuvre.

d) Traitement des bornes supérieures

Dans le cas où les variables x_i possèdent des bornes supérieures (éventuellement infinies), c'est-à-dire $x_i \leq u_i$, il est également possible de procéder à une reformulation du problème donnant un nouveau système KKT à résoudre *[AND96]*. A l'inverse du cas précédent, ce système est tout à fait semblable au cas sans bornes, et on retrouve également des formes similaires pour (AUG) et (NORM), ce qui permet d'appliquer les factorisations de Cholevsky ou Bunch-Partlett, au choix.Cette façon de procéder est bien sûr plus efficace que l'ajout d'une inégalité explicite $x_i \leq u_i$ pour chaque variable bornée (chacune de ces variables entraînant une contrainte et une variable d'écart supplémentaire).

D.2.3 Détection des problèmes impossibles

La plupart des problèmes impossibles sont décelés dès la phase d'analyse initiale. Dans le cas où cette phase ne parvient pas à établir l'absence de point admissible, on peut montrer que les algorithmes font diverger les itérés (x,s) (leur norme tend vers l'infini) . Ce comportement peut être détecté par les logiciels : ceux-ci arrêtent l'algorithme lorsque la norme de x ou de s dépasse une certaine valeur et décrètent le problème impossible *[GON94]*.

Un autre possibilité, plus élégante, consiste à recourir à la technique du problème homogène auto-dual, décrite plus loin (paragraphe E.5 des compléments).

D.3 CHOIX DES PARAMETRES

D.3.1 Taille des pas

Le choix du coefficient α_k, déterminant la taille du pas de Newton que l'on prend à l'itération k, a déjà été discuté lors des présentations des algorithmes. On utilise généralement une petite modification du schéma théorique : on autorise des longueurs de pas différents pour le primal et le dual. Ceci implique donc le calcul de deux valeurs de $\alpha_{k,max}$, à savoir $\alpha_{k,max}{}^p$ et $\alpha_{k,max}{}^d$. On applique alors le plus souvent un coefficient ρ proche de 1, pour obtenir le nouvel itéré :

$$x_{k+1} = x_k + \rho\alpha_{k,\max}{}^p \Delta x_k$$
$$y_{k+1} = y_k + \rho\alpha_{k,\max}{}^d \Delta y_k$$
$$s_{k+1} = s_k + \rho\alpha_{k,\max}{}^d \Delta s_k$$

Des tests ont montré une réduction significative (de l'ordre de 40%) du nombre d'itérations nécessaires par rapport au cadre strict de la théorie où $\alpha_{k,max}$, = $\min(\alpha_{k,max}{}^p , \alpha_{k,max}{}^d)$ est identique pour les deux espaces *[GON94]*.

D.3.2 Choix d'un point initial

On a déjà discuté les différentes façons de traiter un problème pour lequel on ne dispose pas d'un point initial admissible : modifier le problème (pour obtenir un point initial admissible connu) ou modifier la méthode pour autoriser le départ à partir d'un point non-admissible. Dans ce second cas, le choix d'un bon itéré initial peut se révéler crucial (il n'est pas rare de constater des différences de 25% en nombre d'itérations selon le choix de ce point de départ).

On ne dispose à l'heure actuelle que de méthodes heuristiques pour déterminer ce point. Cependant, on a constaté empiriquement qu'un point proche de la solution optimale mais décentré donnait de moins bons résultats qu'un point plus éloigné mais mieux centré.

Une des heuristiques possible est de calculer la solution du problème quadratique suivant, pour lequel il existe une formule explicite (pour un temps de calcul comparable à celui d'une itération) *[AND96]*

$$\begin{aligned}&\min c^T x + \frac{\omega}{2}(x^T x)\\&\text{avec } Ax = b\end{aligned}$$

L'idée poursuivie par ce problème est de trouver un point admissible (contrainte Ax = b) avec une petite norme (minimisation de $x^T x$) tout en favorisant les points qui sont meilleurs au sens de la fonction objectif du problème ($c^T x$). Le paramètre ω est un poids prédéfini. Un problème similaire est résolu pour les variables duales (y,s). Bien sûr, la solution (x,s) fournie aura probablement des composantes négatives. On remplace alors chacune de celles-ci par une petite constante positive pour obtenir un point de départ vérifiant la condition de non-négativité.

D.3.3 Critère d'arrêt

Les logiciels surveillent généralement trois quantités au fur et à mesure des itérations : l'inadmissibilité primale (erreur résiduelle), l'inadmissibilité duale et le saut de dualité. Ces quantités peuvent par exemple être définies à l'aide des rapports suivants *[ARB91]*

$$\frac{\|Ax - b\|}{1 + \|b\|}, \quad \frac{\|A^T y + s - c\|}{1 + \|c\|}, \quad \frac{|c^T x - b^T y|}{1 + |c^T x|}$$

Les normes présentes aux dénominateurs servent à rendre ces quantités relatives (indépendantes d'une multiplication des données par une constante), le terme constant (égal à 1) qu'on leur ajoute sert à éviter les divisions par zéro.

Le logiciel stoppe lorsque ces trois critères tombent sous une précision ε prédéterminée (10^{-8} par exemple).

D.4 TECHNIQUE DE PREDICTION-CORRECTION DE MEHROTRA

D.4.1 Principe

Les méthodes de suivi de chemin poursuivent deux objectifs simultanément : progresser vers la solution, ce qui se fait à l'aide d'un τ proche de zéro, et rester central, ce qui s'obtient en prenant τ proche de un. La direction du pas de Newton comporte deux composantes : une direction de type mise à l'échelle affine, et une direction de centrage.

Imaginons que l'on calcule le pas de Newton pour $\sigma = 0$, ce qui nous donne un incrément $(\Delta x^{aff}, \Delta y^{aff}, \Delta s^{aff})$. L'idée de Mehrotra *[MEH92]* est de se servir de l'information contenue dans ce pas, et ce de deux façons différentes :

- D'une part, on va mesurer l'avancement qui peut être réalisé dans cette direction afin d'estimer le besoin de centrage. Cela permettra de déterminer σ_k de façon complètement adaptative (et non plus selon une valeur fixée à l'avance comme dans les algorithmes présentés plus haut)
- D'autre part, étant donné que cette direction représente un pas de Newton pur vers la solution, on peut s'en servir pour estimer la différence entre l'approximation linéaire (par un hyperplan

tangent) faite par le méthode de Newton et la réalité. Cette différence va nous permettre de calculer un terme correcteur supplémentaire, qui va tenter d'annuler cette différence ; cela aura pour effet de transformer notre approximation linéaire en approximation du second ordre

Une des autres caractéristiques de la méthode de Mehrotra est l'utilisation de longueurs de pas séparées pour les problèmes primal et dual, comme déjà expliqué plus haut.

D.4.2 Déroulement

On commence d'abord par résoudre le système (NWT) pour $\sigma = 0$, ce qui nous donne un incrément $(\Delta x^{aff},\Delta y^{aff},\Delta s^{aff})$. Celui-ci a été calculé en deux étapes : une factorisation de Cholevsky suivie d'une substitution. On détermine ensuite les longueurs de pas maximales admissibles pour le primal et le dual ($\alpha^{aff}_{max}{}^{p}$ et $\alpha^{aff}_{max}{}^{d}$) et on calcule - temporairement - le nouvel itéré qui résulterait de la prise de ces pas maximaux : $(x, y, s) + (\alpha^{aff}_{max}{}^{p}\, \Delta x^{aff}, \alpha^{aff}_{max}{}^{d}\, \Delta y^{aff}, \alpha^{aff}_{max}{}^{d}\, \Delta s^{aff})$. Soit μ_{aff} la mesure de dualité de point.

Mehrotra tient le raisonnement suivant : si μ_{aff} est très inférieur à μ (mesure de dualité de (x, y, s), l'itéré courant), la direction affine est bonne et il ne faut pas beaucoup centrer. Par contre, si μ_{aff} est juste légèrement inférieur à μ, l'itéré a besoin de plus de recentrage. Ceci l'amène à la règle heuristique suivante :

$$\sigma = \left(\frac{\mu_{aff}}{\mu}\right)^3$$

La composante de centrage peut alors être calculée à l'aide d'un système similaire à (NWT), où l'on remplacerait le membre de droite par $(0, 0, \sigma\mu e)^T$. Notons que, puisque la matrice est identique à celle qui a été utilisée par calculer le pas $(\Delta x^{aff},\Delta y^{aff},\Delta s^{aff})$, il ne faudra pas procéder à une nouvelle factorisation de Cholevsky. Seule une deuxième substitution (peu coûteuse en temps de calcul) sera nécessaire.

Imaginons à présent que l'on prenne un pas complet dans la direction préconisée par Newton. On se trouverait alors en $(x, y, s) + (\Delta x^{aff},\Delta y^{aff},\Delta s^{aff}) = (x^{aff},y^{aff},s^{aff})$. Calculons les produits $x_i^{aff}s_i^{aff}$: ceux-ci sont égaux à $\Delta x_i^{aff}\Delta s_i^{aff}$, alors que la méthode de Newton avait « visé » des produits nuls. On va donc incorporer un terme correcteur, qui va tenter de corriger cet écart entre réalité et prévision linéaire.

Ce terme se calcule à l'aide d'un système similaire à (NWT) où l'on remplace le membre de droite par $(0, 0, -\Delta X_i^{aff}\Delta S_i^{aff}e)^T$. La même remarque que pour la composante de centrage est valable ici aussi : seule une substitution est nécessaire. On peut même faire encore mieux, en combinant le terme correcteur et le terme de centrage ensemble en un seul système, où le membre de droite serait $(0, 0, \sigma\mu e - \Delta X_i^{aff}\Delta S_i^{aff}e)^T$

Voici donc la description finale de l'algorithme de Mehrotra :

```
Soit un itéré initial $(x_0,y_0,s_0)$ tel que $(x_0,s_0) > 0$
Pour $k = 0,1,2,\dots$ faire
    ■ Résoudre le système (NWT) avec $(x_k,y_k,s_k)$ et $\tau_k = 0_k$ pour obtenir $(\Delta x_k^{aff},\Delta y_k^{aff},\Delta s_k^{aff})$
    ■ Calculer $\alpha^{aff}_{max}{}^{p}$ et $\alpha^{aff}_{max}{}^{d}$, et $\mu_{aff} = (x_k + \alpha^{aff}_{max}{}^{p}\Delta x_k^{aff})^T(s_k + \alpha^{aff}_{max}{}^{d}\Delta s_k^{aff})/n$
    ■ Poser $\sigma_k = (\mu_{aff}/\mu)^3$
    ■ Résoudre le système (NWT) avec pour membre de droite le vecteur $(0, 0, \sigma\mu e - \Delta X_i^{aff}\Delta S_i^{aff}e)^T$ pour obtenir $(\Delta x_k^{c},\Delta y_k^{c},\Delta s_k^{c})$
    ■ Calculer l'incrément final avec $(\Delta x_k,\Delta y_k,\Delta s_k) = (\Delta x_k^{aff},\Delta y_k^{aff},\Delta s_k^{aff}) + (\Delta x_k^{c},\Delta y_k^{c},\Delta s_k^{c})$
    ■ Calculer $\alpha_{max}{}^{p}$ et $\alpha_{max}{}^{d}$
    ■ $(x_{k+1},y_{k+1},s_{k+1}) \leftarrow (x_k,y_k,s_k) + 0.99\,(\alpha_{max}{}^{p}\,\Delta x_k, \alpha_{max}{}^{d}\,\Delta y_k, \alpha_{max}{}^{d}\,\Delta s_k)$
Fin du pour
```

Cette méthode de prédiction-correction est présente dans tous les logiciels disponibles. En effet, elle permet d'améliorer grandement la convergence avec pour seul inconvénient l'effort supplémentaire - minime - de calculer une substitution supplémentaire par itération. Certains auteurs ont tenté de généraliser l'idée de terme correcteur en appliquant plusieurs corrections successives (afin d'obtenir des méthodes d'ordre encore plus élevé), mais leurs résultats semblent indiquer que la méthode à correction unique donne les meilleures performances *[GON94]*.

E. COMPLEMENTS

Nous présentons ici quelques notions complémentaires intervenant dans l'étude des méthodes de point intérieur.

E.1 THEOREME DE GOLDMAN-TUCKER ET LA PARTITION OPTIMALE

E.1.1 Partition optimale

Soit (x^*,y^*,s^*) une solution optimale du problème dans sa forme standard. On sait que $x_i^* s_i^* = 0$ pour tout indice i, ce qui implique qu'au plus une variable de chaque couple (x_i^*, s_i^*) peut être non-nulle.

Définissons deux ensembles d'indices B et N selon

$$B=\left\{j\in\{1,\ 2,\ ...,\ n\}\mid \exists\ x^*\text{ optimal et }x_j^*\neq 0\right\}\text{ et }N=\left\{j\in\{1,\ 2,\ ...,\ n\}\mid \exists\ (y^*,s^*)\text{ optimal et }s_j^*\neq 0\right\}$$

En vertu de la remarque précédente, B et N sont forcément disjoints (un indice commun k impliquerait l'existence d'un couple optimal avec $x_k^* s_k^* \neq 0$). On peut égaler démontrer que $B \cup N = \{1, 2, ..., n\}$: B et N forment donc une partition de l'ensemble {1, 2, ..., n}, qu'on appelle *partition optimale*.

La propriété $B \cup N = \{1, 2, ..., n\}$ est en fait équivalente à un résultat déjà mentionné : il s'agit de l'existence d'une solution optimale strictement complémentaire, connu sous le nom de théorème de Goldman-Tucker (ce résultat est assez ancien, puisqu'il a été démontré pour la première fois en 1956).

Nous allons maintenant démontrer cette équivalence.

E.1.2 Goldman-Tucker ⇒ (B, N) est une partition

Supposons que l'on dispose d'une solution optimale (x^*,y^*,s^*) strictement complémentaire, c'est-à-dire vérifiant $x_i^* s_i^* = 0$ et $x_i^* + s_i^* > 0$. Pour montrer que (B,N) est une partition, il suffit de montrer que $B \cup N = \{1, 2, ..., n\}$ (cf. plus haut). Soit un indice k quelconque. En vertu de la complémentarité de (x^*,y^*,s^*), on a soit $x_k^* \neq 0$, soit $s_k^* \neq 0$. Dans le premier cas, cela implique $k \in B$, dans le second cas, $k \in A$. Dans les deux cas, on a $k \in B \cup N$, ce qui prouve bien que $B \cup N = \{1, 2, ..., n\}$.

E.1.3 (B, N) est une partition ⇒ Goldman-Tucker

Soit $j \in B$. En vertu de la définition de B, il existe un x optimal vérifiant $x_j \neq 0$, que l'on appellera x^j. Définissons x^* par

$$x^*=\frac{1}{|B|}\sum_{j\in B}x^j$$

(|B| est le cardinal de l'ensemble B). Etant donné que les x^j sont admissibles et optimaux, il est facile de montrer que x^* est également une solution optimale. De plus, puisque $x^j \geq 0$ et $x_j^j > 0$, on a bien $x_j^* > 0$ pour tout $j \in B$. On peut de même définir de la même façon un couple optimal (y^*, s^*) tel que $s_j^* > 0$ pour tout $j \in N$.

Cela implique finalement que $x_j^* + s_j^* > 0$ pour tout j (puisque (B,N) est une partition de {1, 2, ..., n}), ce qui achève de prouver que (x^*, y^*, s^*) est une solution strictement complémentaire, d'où le résultat de Goldman-Tucker *[GOL56]*.

E.1.4 Calcul d'une solution optimale à partir de (B, N)

Notons encore que la détermination de B et N est un problème au moins aussi difficile que la résolution du programme linéaire lui-même. En effet, si l'on dispose de (B,N), il est possible de calculer facilement la solution optimale, à l'aide d'un problème de moindre carrés (voir le paragraphe au sujet de la terminaison finie, p. 48).

E.2 FORME DU CHEMIN CENTRAL ET NOMBRE DE CONDITION

Certains auteurs ont étudié en détail les propriétés géométriques et analytiques du chemin central *[VAV96]*. Ainsi, outre les propriétés déjà mentionnées, on sait que cette courbe est différentiable et qu'elle est constituée d'un ensemble de morceaux quasiment rectilignes reliés entre eux par des *tournants* brusques (impliquant des changements de directions). La Figure B-1 donne un exemple typique de chemin central

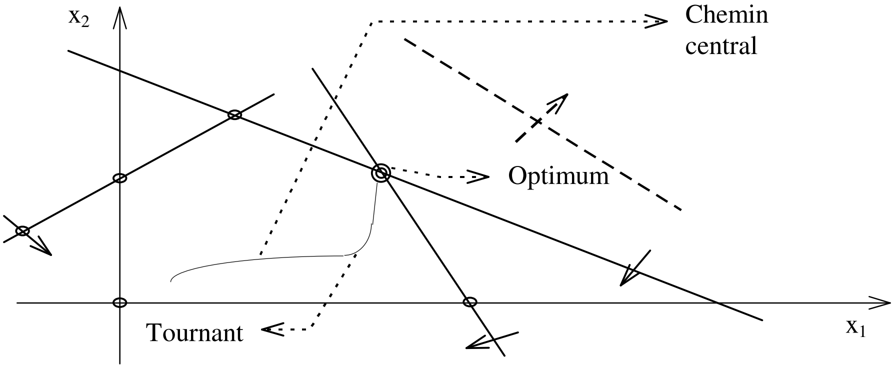


Figure E-1 - Exemple de chemin central comprenant un tournant brusque

Définissons le nombre de condition ε(A,b,c), dépendant uniquement des données du problème linéaire, par

$$\varepsilon(A,b,c) = \min\left(\min_{i\in B} \sup_{x^* \text{ optimal}} x_i^*,\ \min_{i\in N} \sup_{(y^*,s^*) \text{ optimal}} s_i^* \right)$$

(où (B,N) est la partition optimale définie plus haut). Ce nombre est intimement lié au problème et à son chemin central : en effet, on a montré que le dernier tournant de ce chemin survient pour une mesure de dualité μ approximativement égale à ε^2.

Signalons aussi que cette quantité n'est pas une fonction continue des données (A,b,c), car une perturbation infinitésimale de celles-ci peut entraîner un grand changement de la valeur de ε *[WRI96,* p. 152*]*. Tout comme pour la partition optimale (B,N), le calcul de ε est un problème aussi difficile que la résolution du programme linéaire lui-même. Les propriétés du nombre ε ne sont pas encore bien comprises pour le moment.

E.3 CONVERGENCE SUPERLINEAIRE

Durant les dernières itérations de l'optimisation d'un programme linéaire par une méthode de point intérieur, lorsqu'on est très proche de la solution finale, on a constaté que les pas de Newton purs en direction de la solution optimale avaient très peu de chance de sortir de la région admissible (c'est-à-dire de violer les contraintes de non-négativité), contrairement à la situation survenant en début d'optimisation où de tels pas amènent rapidement au bord de la région admissible, ce qui réduit à la fois la taille des pas suivants et la vitesse de convergence.

On peut donc imaginer qu'à partir d'un certain moment, on décide de ne plus utiliser un pas de Newton visant une cible du chemin central ($\tau \neq 0$ dans NWT) mais directement le pas de Newton visant l'optimum ($\tau = 0$, comme dans la méthode de mise à l'échelle affine). Il est alors possible de démontrer que cette modification, si elle est faite au bon moment, entraîne une convergence bien plus rapide vers la solution optimale, de type superlinéaire [*WRI96,* p. 127*]*. Signalons que cette démonstration se base sur le fait que les longueurs des composantes du pas de Newton Δx et Δs ont alors une taille de l'ordre de μ (en d'autres termes : (Δx, Δs) = O(μ)). La nature exacte de cette convergence superlinéaire dépend de l'algorithme employé (certains présentent par exemple une convergence asymptotiquement quadratique).

Cependant, ce changement de stratégie n'accélère vraiment la convergence que s'il est effectué au bon moment, c'est-à-dire lorsque l'itéré courant est suffisamment proche de la solution optimale. On a montré que la convergence superlinéaire se produit effectivement à partir d'un certain seuil de la mesure de dualité μ de l'itéré courant : ce seuil est approximativement égal à ε^2 (où ε est le nombre de condition défini plus haut).

E.4 TERMINAISON FINIE

Il existe une technique permettant de *deviner* en une étape la solution optimale à partir d'un itéré proche de celle-ci. Cette procédure porte le nom de terminaison finie.

Son principe repose sur l'utilisation de l'itéré courant pour estimer la partition optimale (B,N). On sait que la solution optimale strictement complémentaire (x^C,y^C,s^C) possède les propriétés suivantes : $s_i^C > x_i^C = 0\ \forall i \in N$ et $x_i^C > s_i^C = 0\ \forall i \in B$. Il semble donc raisonnable de deviner la partition optimale à partir de l'itéré courant et (x,y,s) de la comparaison des composantes x_i et s_i. Si $x_i > s_i$, l'indice i appartient probablement à B, dans le cas contraire ($x_i < s_i$) il fait vraisemblablement partie de N.

Une fois cette partition (B,N) devinée, on résout le problème suivant (c'est un simple problème de moindres carrés pour lequel il existe une formule directe fournissant la solution)

$$\min_{(x^*,y^*,s^*)} \tfrac{1}{2}\left\|x^* - x\right\| + \tfrac{1}{2}\left\|s^* - s\right\|$$
$$Ax^* = b,\ A^T y^* + s^* = c,\ x_i^* = 0\ \forall i \in N \text{ et } s_i^* = 0\ \forall i \in B$$

Si la solution (x*, y*, s*) de ce problème est admissible pour le problème initial (c'est-à-dire si elle vérifie bien $x_i^* > 0\ \forall i \in B$ et $s_i^* > 0\ \forall i \in N$ en plus des contraintes du problème), c'est une solution optimale strictement complémentaire.

On démontre que si la partition devinée (B,N) est correcte, le problème ci-dessus fournit dans tous les cas une solution optimale. Dans le cas contraire, le couple (x*,s*) qu'elle fournit comporte forcément une composante négative.

Comme dans le cas de la convergence superlinéaire, cette technique ne fonctionne que lorsqu'on se situe près de la solution. On montre en fait qu'il existe aussi un seuil de la mesure de dualité à partir duquel la partition devinée (B,N) est toujours correcte (et le problème de moindres carrés fournit l'optimum) : ce seuil est lui aussi approximativement égal à ε^2.

Pour conclure, il est intéressant de remarquer les connections étroites entre les trois mesures du dualité suivantes, toutes trois approximativement égales à ε^2 :

- μ_1, mesure de dualité de la portion du chemin central où se produit le dernier tournant brusque
- μ_2, mesure de dualité à partir de laquelle la convergence superlinéaire par pas de Newton pur devient effective
- μ_3, mesure de dualité à partir de laquelle la procédure de terminaison finie fournit la solution optimale

Les raisons de ces connections ne sont en fait pas encore totalement comprises par les chercheurs du domaine.

E.5 TECHNIQUE DU PROBLEME HOMOGENE AUTO-DUAL

On peut considérer que deux difficultés ne sont pas résolues de façon satisfaisante par les méthodes de point intérieur décrites plus haut :

- D'une part, il est difficile de trouver un bon point de départ (admissible et bien centré)

- D'autre part, les problèmes impossibles ne sont pas détectés directement, mais par une divergence des itérés. Bien que cette solution soit acceptable d'un point de vue pratique, il serait plus intéressant de pouvoir détecter l'impossibilité du problème par une convergence des itérés.

Le problème homogène auto-dual résout ces deux problèmes simultanément.

Soit un problème posé dans la forme standard. A partir des données (A, b, c), construisons le problème (primal) suivant

$$\begin{array}{llll} \min\ 0 & & & \\ & -c^T x & + b^T y & \geq 0 \\ \text{avec}\quad c\tau & & - A^T y & \geq 0 \\ \quad - b\tau & + Ax & & = 0 \\ \text{et y libre,}\ (\tau, x) > 0 & & & \end{array}$$

On remarque que l'on a introduit une variable scalaire supplémentaire τ. Ce problème est à la fois homogène (parce que son membre de droite est nul) et auto-dual (il est facile de vérifier que le dual de ce problème est identique au primal). Il admet une solution optimale triviale $(x,y,\tau) = 0$, et n'est donc jamais impossible.

Si on introduit des variables d'écart (κ, s) pour les deux inégalités, on est ramené à la forme standard et on peut appliquer une méthode de point intérieur classique pour la résoudre. Soit $(x^*,y^*,s^*,\tau^*,\kappa^*)$ la solution optimale obtenue. Il ne s'agira pas de l'optimum trivial, mais bien d'une solution optimale strictement complémentaire vérifiant $x_i^*s_i^* = 0$, $\tau^*\kappa^* = 0$ (optimalité) et $x^* + s^* > 0$, $\tau^* + \kappa^* > 0$ (complémentarité). En fonction des valeurs de τ^* et κ^*, on déduit des informations sur le programme linéaire initial. En effet, on montre que :

- Le problème initial n'est pas impossible si et seulement si $\kappa^* = 0$ (et donc forcément $\tau^* > 0$). Dans ce cas, le triplet $(x,y,s) = (x^*/\tau^*, y^*/\tau^*, s^*/\tau^*,)$ est une solution optimale strictement complémentaire à ce problème initial.
- Lorsque $\kappa^* > 0$ (et donc forcément $\tau^* = 0$), on a $c^Tx^* - b^Ty^* < 0$. On montre alors que :
  - $- b^Ty^* < 0$ implique que le dual du problème initial est impossible
  - $c^Tx^* < 0$ implique que le primal du problème initial est impossible

  Notons que l'on se trouve alors forcément dans au moins une de ces deux situations.

Etant donné que le problème homogène auto-dual admet une solution triviale, il n'est pas impossible et ses itérés ne divergent jamais. On détecte bien l'impossibilité du problème initial par une convergence.

Le seul inconvénient de cette technique est qu'il n'existe pas de solution strictement admissible ($\Im_0 = \emptyset$) au problème auto-dual, ce qui nous force à employer une méthode à départ non-admissible pour sa résolution.

Il existe une forme légèrement plus compliquée du problème homogène auto-dual due à Ye, Todd et Mizuno *[YE94]*. Celle-ci comporte une ligne et une colonne supplémentaire, ce qui permet d'incorporer l'inadmissibilité d'un point de départ quelconque (x_0, y_0, s_0). Avec cette version, on dispose donc d'un itéré initial vérifiant les contraintes, ce qui permet de choisir une méthode à départ admissible. De plus, $\Im_0 \neq \emptyset$ entraîne que l'ensemble des solutions optimales est borné, ce qui garantit une meilleure stabilité numérique des algorithmes. Des résultats similaires à ceux présentés ci-dessus sont valables pour cette forme plus compliquée.

En résumé, cette méthode permet donc de disposer d'un itéré de départ admissible (que l'on choisira le plus souvent centré) et de détecter l'impossibilité éventuelle par une convergence. Son seul inconvénient est l'augmentation (légère) de la taille du problème à résoudre et l'apparition de deux colonnes denses. Cependant, l'effort de calcul supplémentaire peut être fortement réduit en employant la technique de factorisation basée sur la formule de Sherman-Morrison-Woodbury.

F. RESULTATS NUMERIQUES ET PERSPECTIVES

F.1 COMPARAISONS

Plutôt que d'utiliser des problèmes générés aléatoirement, la majorité des chercheurs du domaine des méthodes de point intérieur utilisent des problèmes issus d'applications réelles pour tester leurs implémentations. La plus célèbre collection de tels problèmes est connue sous le nom de *NETLIB* (disponible sur le réseau Internet) *[GAY85]*, mais il en existe de nombreuses autres (problèmes de grande taille, problèmes impossibles, problèmes présentant une structure particulière, etc.).

Signalons qu'il existe un standard de facto pour les fichiers de données décrivant les programmes linéaires : il s'agit du format MPS (*Mathematical Programming System*, développé à l'origine par IBM) *[IBM79, LUO92]*.

F.1.1 Méthodes de point intérieur et algorithme du simplexe

A l'heure actuelle, une personne désirant résoudre un programme linéaire a le choix entre la méthode du simplexe et une méthode de point intérieur. Les résultats des comparaisons effectuées par de nombreux chercheurs tendent à la même conclusion *[LUS94, AND96]* :

- Pour des problèmes de petite taille (n+m < 2000), la méthode du simplexe semble imbattable
- Pour des problèmes de taille moyenne (n+m < 10000), aucune méthode ne présente d'avantage clair. Le résultat dépendra du problème, en particulier de la structure des éléments non-nuls de la matrice A (qui influence de façon notable l'efficacité des méthodes de point intérieur)
- Pour des problèmes de très grande taille (n+m > 10000), les méthodes de point intérieur semblent prendre le dessus sur leur concurrent

Bien sûr, cette classification est aussi fluctuante (quant aux bornes des intervalles) que provisoire, étant donné que la recherche fait constamment de nombreux progrès, aussi bien du côté du simplexe que des méthodes de point intérieur. Cependant, il semble qu'un consensus soit à présent bien établi au sujet de ces trois catégories de problèmes et que la compétition qui a sévi entre partisans de la méthode du simplexe et des méthodes de point intérieur se soit quelque peu apaisée.

F.1.2 Domaine public et logiciels commerciaux

A l'heure actuelle, les logiciels d'envergure appliquant une méthode de point intérieur pour résoudre des problèmes linéaires sont les suivants (nom, langage, disponibilité et auteur)

- BPMPD, FORTRAN, domaine public (usage non commercial), par Mészàros
- CPLEX/BARRIER, C, commercial, par CPLEX
- HOPDM, FORTRAN, domaine public, par Gondzio
- LIPSOL, MATLAB + FORTRAN, domaine public, par Zhang
- LOQO, C, domaine public (usage non commercial), par Vanderbei
- NEWTON BARRIER XPRESS-MP, C + Fortran, commercial
- OSL/EKKBSLV, FORTRAN, commercial, IBM
- PCX, C + FORTRAN, domaine public, Czyzyk, Mehrotra et Wright

Presque tous ces logiciels sont basés sur une méthode primale-duale à départ non-admissible, avec prédiction-correction de Mehrotra. Certains incluent une analyse initiale plus poussée, d'autres des corrections multiples, d'autres encore la détermination d'une solution de base.

Les tests menés par certains auteurs semblent indiquer qu'il n'y a pas de différence significative de performance entre les logiciels public et les codes commerciaux.

F.2 TESTS NUMERIQUES

F.2.1 Simplexe, méthodes de point intérieur publiques et méthodes de point intérieur commerciales

A titre d'exemple, voici les résultats obtenus par Andersen, Gondzio, Mészàros et Xu *[AND96]* lors d'une comparaison de six logiciels de programmation linéaire. Il s'agissait d'une méthode du simplexe (CPLEX/SimplexMethod), d'une méthode de point intérieur commerciale (CPLEX/Barrier) et de quatre méthodes de point intérieur du domaine public (LIPSOL, LOQO, HOPDM et BPMPD). Celles-ci ont été testées sur six problèmes.

Avant de présenter les résultats contenus dans le Tableau 3, il faut apporter les précisions suivantes :

- Tous ces tests ont été menés sur le même ordinateur, avec les options par défaut de chaque logiciel. Il est clair qu'il est possible d'améliorer les résultats en adaptant les paramètres des algorithmes à chaque problème particulier
- Chaque problème a été résolu avec une précision de huit chiffres significatifs
- Les problèmes ont été choisis dans la catégories « grande taille », le domaine où les méthodes de point intérieur sont normalement le plus à l'aise
- Pour chaque problème, le Tableau 2 - Taille de densité des problèmes retenus pour le test fournit les dimensions m et n ainsi que le nombre d'éléments non-nuls et la densité de la matrice A :

Nom	***n***	***m***	***Non-nuls***	***Densité***
pilot87	2030	4883	73804	0,75 %
dfl001	6071	12230	41873	0,06 %
pds-10	16558	48763	140063	0,02 %
mod2	35664	31728	220116	0,02 %
world	35510	32734	220748	0,02 %
NL	7195	9718	102570	0,14 %

Tableau 2 - Taille de densité des problèmes retenus pour le test

- Le nombre d'itérations d'une méthode de type simplexe n'est pas comparable avec celui d'une méthode de point intérieur, car ces dernières itérations demandent beaucoup plus de calculs. Le seul point de comparaison valable entre méthode de point intérieur et algorithme du simplexe est le temps d'exécution total

Voici donc, pour chaque paire logiciel/problème, le nombre d'itérations effectué et le temps total d'exécution (en secondes, sur une station de travail IBM POWERPC 601 à 66 Mhz)

Nom	***CPLEX/Simplex***		***CPLEX/Bar.***		***LIPSOL***		***LOQO***		***HOPDM***		***BPMPD***	
	Itér.	Temps	Itér	Temps	Itér	Temps	Itér	Temps	Itér	Temps	Itér	Temps
pilot87	101167	722	41	603	38	817	47	1221	27	501	30	392
dfl001	63389	3767	47	3829	85	9907	53	7379	33	5294	34	2921
pds-10	38327	1223	60	3462	51	5824	51	5617	29	3394	30	2715
NL	32273	112408	31	147	35	319	29	210	23	171	30	149
mod2	117360	18200	57	947	72	3737	73	1592	47	1069	45	876
world	134270	22470	62	1146	60	3115	74	1754	51	1345	57	1109

Bien que cet ensemble de résultats soit trop réduit pour en tirer des conclusions définit

F.3 DIRECTIONS ACTUELLES DE RECHERCHE

Outre l'application des méthodes de point intérieur à la programmation non-linéaire (dont la programmation semidéfinie est un exemple, cf. plus loin), citons les directions de recherche suivantes pour le cas linéaire :

- Démarrage à chaud. Après avoir résolu un problème, on aimerait pouvoir le modifier légèrement (par exemple ajouter quelques contraintes) et le réoptimiser rapidement (ce besoin d'optimisations successives survient par exemple lors de la résolution d'un problème en nombres entiers par une méthode de type *branch & bound* ou *branch & cut*, où l'on est amené à résoudre de nombreux problèmes linéaires, différant les uns des autres par l'ajout de quelques contraintes supplémentaires).

  La méthode du simplexe dispose pour le moment d'un gros avantage sur les méthodes de point intérieur, parce qu'elle ne nécessite que très peu d'itérations pour parvenir à la solution optimale du nouveau problème (les méthodes de point intérieur nécessitent quant à elles grosso modo un tiers du nombre d'itérations du problème de départ pour réoptimiser *[GON94,* p. 30*]*). On envisage d'ailleurs des logiciels hybrides qui optimiseraient le premier problème par une méthode de point intérieur pour ensuite réoptimiser les problèmes suivants par un algorithme de type simplexe.

- Certains chercheurs avancent que l'analyse de sensibilité que l'on effectue à partir de la solution de base fournie par l'algorithme du simplexe est incomplète, voire erronée dans certaines situations (lorsque le problème est dégénéré) *[JAN92]*. Des recherches sont en cours pour déterminer un moyen efficace d'effectuer cette analyse correctement (cela nécessiterait la résolution de certains programmes linéaires auxiliaires).

- De travaux sont en cours pour implémenter des méthodes de point intérieur sur des machines à plusieurs processeurs. L'effort porte surtout sur la parallélisation efficace de l'étape de factorisation (la plus gourmande en calculs) *[KAR94, LUO92]*.

DEUXIEME PARTIE
LES METHODES DE POINT INTERIEUR POUR LA PROGRAMMATION SEMIDEFINIE

G. DEVELOPPEMENTS THEORIQUES

G.1 INTRODUCTION

Un des grands attraits des méthodes de point intérieur est qu'elles s'appliquent également à la programmation non-linéaire. Ainsi, le travail de Nesterov et Nemirovsky *[NES94]* a fourni les bases théoriques nécessaires au développement de méthodes de point intérieur polynomiales pour la programmation convexe, un cas particulier important de la programmation non linéaire possédant de nombreuses applications.

Plutôt que d'entamer une énumération des nombreuses variantes qui ont été développées pour des problèmes non-linéaires (les plus courantes concernant le problème de complémentarité non-linéaire monotone et la programmation quadratique convexe, voir *[WRI96]*), nous avons préféré nous en tenir à une seule catégorie de problèmes, pour laquelle nous allons donner une description plus détaillée : il s'agit de la programmation semidéfinie.

En quelques mots, il s'agit d'un problème non-linéaire et non différentiable convexe qui a pour objet d'optimiser une matrice carrée symétrique X (et non plus un vecteur) sous une série de contraintes linéaires. La condition de non-négativité du vecteur x est alors remplacée par l'exigence que la matrice X soit semidéfinie positive.

Nous avons choisi cette catégorie de problèmes pour deux raisons : d'une part, elle admet un nombre très élevé d'applications dans de nombreux domaines (y compris des domaines où la programmation linéaire est peu ou pas employée, comme le contrôle ou les statistiques) ; d'autre part il s'agit de l'extension de la programmation linéaire qui a reçu le plus d'attention de la part de la communauté scientifique, principalement durant ces dernières années, probablement en raison du parallèle simple et élégant que l'on peut établir avec la théorie du cas linéaire.

G.1.1 Définitions

a) Matrices définie positive et semidéfinie positive

Une matrice carrée symétrique A est dite *définie positive* si on a $x^T Ax > 0$ pour tout vecteur $x \neq 0$. Si on a seulement $x^T Ax \geq 0 \ \forall x \neq 0$, on parle de matrice *semidéfinie positive*. On sait qu'une matrice est définie positive si et seulement si toutes ses valeurs propres sont strictement positives (non-négatives pour une matrice semidéfinie positive).

b) Espace S^n et ordre partiel

Soit S^n l'espace vectoriel réel des matrices carrées symétriques (nxn). On définit un ordre partiel sur cet espace selon

$A \geq B$ si et seulement si la matrice $(A - B)$ est semidéfinie positive

et $A > B$ si et seulement si la matrice $(A - B)$ est définie positive

Cet ordre partiel se nomme *ordre de Löwner*. Dans la suite de ce document, toutes les inégalités concernant des matrices seront relative à cet ordre partiel. Remarquons encore qu'on peut exprimer qu'une matrice A est définie positive par $A > 0$ (et qu'une matrice est semidéfinie positive par $A \geq 0$), en adoptant la convention de représenter une matrice nulle par 0 (la taille étant déterminée par le contexte).

c) Produit scalaire sur S^n

Soit $A, B \in S^n$. On définit le *produit scalaire* $A \bullet B$ par le nombre

$$A \bullet B = \sum_{i,j} a_{ij} b_{ij} = \mathrm{trace}(A^T B)$$

(trace(A) dénote la somme des éléments de la diagonale de A). Ce produit scalaire permet de définir la norme matricielle de Frobenius, à savoir (les λ_i sont les valeurs propres de A).

$$\|A\| = \sqrt{A \bullet A} = \sqrt{\sum_{i,j} a_{ij}^2} = \sqrt{\operatorname{trace}(A^T A)} = \sqrt{\sum_i \lambda_i^2}$$

Signalons encore le résultat suivant , utilisé dans la suite : si A,B $\in S^n$ et $A \geq 0$, $B \geq 0$, on a $A \bullet B = 0$ si et seulement si $AB = 0$.

G.1.2 Formulation

Le problème standard de la programmation semidéfinie s'écrit

$$\begin{aligned} &\inf\ C \bullet X \\ &\text{avec } A_i \bullet X = b_i \text{ et } X \geq 0 \text{ (SDP)} \\ &\text{où } X \in S^n \text{ et } i = 1,2,\ldots,m \end{aligned}$$

Les données sont les matrices carrées A_i et C (de dimension n) et le vecteur b (de dimension m). On remarque immédiatement une forte ressemblance avec la forme standard de la programmation linéaire. Cette dernière constitue d'ailleurs un cas particulier de la programmation semidéfinie.

En effet, si on ajoute la contrainte que X soit une matrice diagonale, l'objectif devient $c^T x$ (où c est la diagonale de C et x celle de X, ce qui se note c = diag(C) et x = diag(X)) et les m contraintes deviennent des contraintes linéaires en fonction des éléments de x (plus précisément, $\operatorname{diag}(A_i)^T x = b_i$). Enfin, les valeurs propres d'une matrice diagonale étant les éléments de cette diagonale, la condition $X \geq 0$ est bien équivalente à $x \geq 0$.

La suite de ce chapitre sera principalement consacrée à la mise en évidence et l'analyse des similitudes et différences entre programmation semidéfinie et programmation linéaire. On y retrouve des concepts familiers tels que dualité, chemin central, etc.

G.2 PARALLELE AVEC LE CAS LINEAIRE

G.2.1 Région admissible

L'ensemble des points $X \in S^n$ admissibles (vérifiant les contraintes) porte le nom de spectraèdre. A la différence du polyèdre admissible que l'on rencontre en programmation linéaire, les faces du spectraèdre ne sont plus forcément des hyperplans mais peuvent être des hypersurfaces courbes. La Figure G-1 fournit un exemple de spectraèdre dans le cas bidimensionnel :

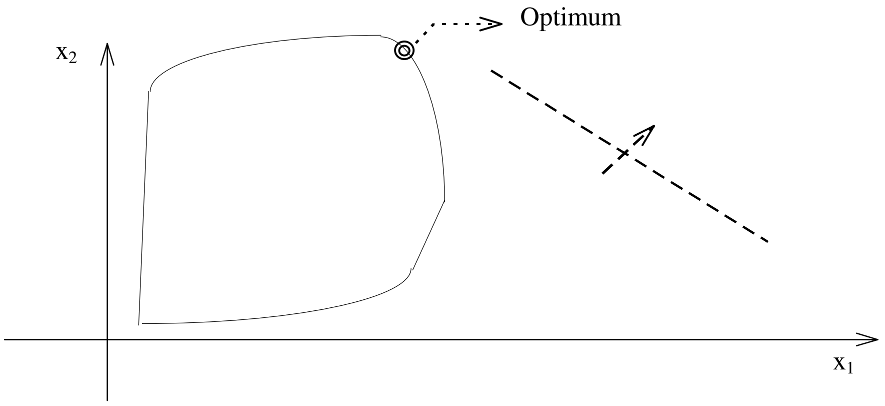


Figure G-1 - Exemple d'optimisation sur un spectraèdre à deux dimensions

On remarque que la frontière du spectraèdre n'est pas différentiable. On peut montrer qu'elle est en fait différentiable par morceaux, les morceaux étant des hypersurfaces algébriques (c'est-à-dire définies par un polynôme) *[VAN96,* p. 50*]*.

Il est possible de montrer que les spectraèdres sont des ensembles convexes (ce qui implique que la programmation semidéfinie fait partie de la programmation convexe, comme annoncé plus haut). Les ellipsoïdes sont des cas particuliers de spectraèdres.

G.2.2 Ensemble des solutions et problème impossible

Soit p* la valeur optimale du critère pour le problème (SDP). Il faut noter qu'il s'agit d'une borne inférieure (inf C • X) et non d'un minimum comme dans le cas linéaire. Ceci s'explique par le fait qu'il existe des situations où la borne inférieure n'est pas atteinte, contrairement au cas linéaire (où l'on peut alors remplacer inf par min). Si l'on appelle X_{OPT} l'ensemble des X optimaux, c'est-à-dire tels que X est admissible et C • X = p*, il se peut donc que $X_{OPT} = \emptyset$ alors que p* est fini.

Une autre différence avec la programmation linéaire est l'existence de problèmes dits faiblement impossibles. En effet, si les problème suivants

$$\begin{aligned}&\inf\ C \bullet X\\&\text{avec } |A_i \bullet X - b_i| \leq \varepsilon \text{ et } X \geq 0\\&\text{où } X \in S^n \text{ et } i = 1,2,\ldots,m\end{aligned}$$

admettent une solution pour tout ε > 0, mais pas pour ε = 0, on qualifie le problème original de faiblement impossible. Par opposition, les autres problèmes impossibles seront appelés fortement impossibles.

G.2.3 Dualité

Le problème (SDP) admet naturellement un problème dual (celui-ci se déduit du primal à l'aide de la théorie des multiplicateurs de Lagrange, on parle donc de dual Lagrangien), qui s'écrit

$$\begin{aligned}&\sup\ b^T y\\&\text{avec } \sum_{i=1}^{m} y_i A_i \leq C\\&\text{où } y \in \Re^m\end{aligned}$$

Comme dans le cas linéaire, la variable y est libre. On peut également introduire une matrice d'écart S semidéfinie positive, ce qui donne la forme suivante

$$\begin{aligned}&\sup\ b^T y\\&\text{avec } \sum_{i=1}^{m} y_i A_i + S = C\\&\text{où } y \in \Re^m,\ S \in S^n \text{ et } S \geq 0\end{aligned}$$

Signalons encore que l'on peut reformuler le problème dual sous une forme identique au problème primal (avec des données A_i, C et b ad hoc). Cela prouve, entre autres, que la région admissible du dual est aussi un spectraèdre.

Une grande partie des propriétés de la dualité linéaire (mais pas toutes) sont encore valables :

- On désigne les ensembles de points (X,y,S) admissibles et strictement admissibles par $\Im$ et $\Im_0$, la valeur optimale du critère (borne supérieure) par d* et l'ensemble des S optimaux par S_{OPT}.
- La quantité $C \bullet X - b^T y$, toujours positive, est appelée le saut de dualité (*duality gap*). On montre aisément, par de simples manipulations algébriques, que $C \bullet X - b^T y = X \bullet S$. Lorsque le saut de dualité est nul, la solution (X,y,S) est optimale, mais la réciproque n'est pas vraie (contrairement au cas linéaire). En effet, on peut assister aux situations pathologiques suivantes
    - Il existe une solution optimale (X,y,S) telle que X • S > 0
    - Pour tout ε > 0, il existe un point admissible (X,y,S) tel que X • S ≤ ε, mais il n'existe pas d'optimum
    - Il existe des problèmes ayant une solution optimale finie dont le dual est impossible
- On peut montrer qu'un problème est fortement impossible si et seulement si il admet un rayon, ce qui revient à dire qu'il existe une matrice symétrique $X^r \geq 0$ telle que $X^r \bullet A_i = 0$ et $C \bullet X^r < 0$. La situation est identique pour la forme duale.

- On appelle solution complémentaire une solution admissible (X,y,S) où l'on a XS = 0 (cette dernière condition pouvant s'exprimer comme $X \bullet S = 0$ puisque $X \geq 0$ et $S \geq 0$). Une solution complémentaire est forcément optimale (puisque son saut de dualité est nul), mais un problème possible n'admet pas forcément de solution complémentaire.
- Signalons encore une propriété d'orthogonalité, propre à la programmation semidéfinie : si $(X^1,y^1,S^1) \in \Im$ et $(X^2,y^2,S^2) \in \Im$, on a $(X^1 - X^2) \bullet (S^1 - S^2) = 0$

La théorie de la dualité en programmation semidéfinie est donc bien plus complexe que son équivalent linéaire, principalement à cause de l'existence de nombreuses situations pathologiques ($X_{OPT} = 0$ avec p^* fini, problème faiblement impossible, saut de dualité non nul à l'optimum). On dispose cependant d'un théorème limitant l'apparition de ces cas *[VAN96]* :

Si le problème primal admet une solution strictement admissible (c'est-à-dire où $X > 0$) ou si son dual admet une solution strictement admissible (c'est-à-dire où $S > 0$), on a $p^* = d^*$, ce qui signifie que le saut de dualité à l'optimum sera nul. Si les deux problèmes admettent simultanément une solution strictement admissible (c'est-à-dire si $\Im_0 \neq \emptyset$), les bornes inférieure et supérieure p* et d* sont atteintes, ce qui entraîne l'existence d'une solution optimale (X,y,S) ($X_{OPT} \neq \emptyset$ et $S_{OPT} \neq \emptyset$).

En pratique, lors du développement d'algorithmes, on supposera cette dernière condition (appelée contrainte de qualification de Slater) toujours vérifiée. Toutefois, tout comme dans le cas linéaire, il existe une reformulation homogène auto-duale, permettant de détecter les cas d'impossibilité ou de problèmes non bornés, ainsi que les cas pathologiques mentionnés plus haut.

G.2.4 Conditions KKT

Comme pour la programmation linéaire, les conditions d'optimalité KKT du problème primal et de son dual sont identiques. Elles sont de plus nécessaires et suffisantes, à cause de la convexité du problème, et s'écrivent

$$\begin{aligned} &A_i \bullet X = b_i \text{ pour } i = 1,2,\ldots,m \\ &\sum_{i=1}^{m} y_i A_i + S = C \\ &XS = 0 \\ &X \geq 0 \text{ et } S \geq 0 \end{aligned}$$

La seule équation non-linéaire parmi cet ensemble est XS = 0. Il a déjà été précisé plus haut (p. 55) que cette troisième condition est équivalente à $X \bullet S = 0$. Les méthodes de point intérieur en programmation semidéfinie vont donc tenter de résoudre ce système par la méthode de Newton.

G.2.5 Fonction barrière

Comme pour la programmation linéaire, on utilise une fonction barrière pour empêcher la méthode de Newton de s'approcher des contraintes $X \geq 0$ et $S \geq 0$. La théorie de Nesterov et Nemirovsky nous apprend qu'une telle barrière doit être auto-concordante pour être efficace (c'est-à-dire entraîner une complexité polynomiale). La plus simple et la plus élégante de ces barrières est définie par

$$\Phi(X) = -\ln \det X$$

(où det X est le déterminant de la matrice X). En effet, lorsqu'on s'approche de $X \geq 0$, une des valeurs propres de X doit tendre vers zéro, ce qui fait également tendre le déterminant vers zéro et Φ vers $+\infty$.

G.2.6 Chemin central et mesure de dualité

La solution du problème auquel on a adjoint la fonction barrière peut encore s'interpréter comme un point d'un chemin central, à l'aide des conditions KKT. Pour une valeur donnée du paramètre de la barrière τ, il est défini par

$$\begin{aligned}&A_i \bullet X = b_i \; i = 1,2,\ldots,m\\&\sum_{i=1}^{m} y_i A_i + S = C\\&XS = \tau I\\&X > 0 \text{ et } S > 0\end{aligned}$$

On a donc remplacé la condition de complémentarité XS = 0 par la relaxation XS = τI. Afin de mesurer la proximité par rapport au chemin central, on définit aussi une mesure de dualité μ par

$$\mu(X,y,S) = \frac{1}{n} X \bullet S$$

On peut vérifier que la mesure de dualité d'un point du chemin central repéré par τ vaut bien τ. Les voisinages du chemin central sont définis de manière analogue à leurs équivalents du cas de la programmation linéaire.

G.2.7 Fonction potentiel

Enfin, la fonction potentiel de Tanabe-Todd-Ye admet l'équivalent suivant

$$\Phi_\rho(X,y,S) = \rho \ln(X \bullet S) - \ln \det XS$$

G.2.8 Mise en parallèle des concepts avec le cas linéaire

Il est possible d'établir un parallèle encore plus fort entre les concepts de la programmation semidéfinie et ceux de la programmation linéaire à l'aide de la remarque suivante :

> Les équations de la programmation semidéfinie sont équivalentes à celles de la programmation linéaire où l'on aurait remplacé les matrices M ∈ S^n par λM, le vecteur formé des valeurs propres de la matrice M rangées en ordre décroissant.

Ainsi, la contrainte X ≥ 0 (ou X > 0) est bien équivalente à $\lambda_X \geq 0$ (ou $\lambda_X > 0$). De même, la condition XS = τI est équivalente à $\lambda_{XS} = \tau e$ ou $(\lambda_{XS})_i = \tau$ pour i=1,2,...,n, et l'on retrouve la contrainte $x_i s_i = \tau$. La mesure de dualité μ est égale à :

$$\mu(X,y,S) = \frac{1}{n} X \bullet S = \frac{1}{n}\operatorname{trace}(XS) = \frac{1}{n}\sum_{i=1}^{n} (\lambda_{XS})_i \text{ , à rapprocher de } \mu(x,y,s) = \frac{1}{n}\sum_{i=1}^{n} x_i s_i$$

(puisque la trace d'une matrice est la somme de ses valeurs propres). Enfin, la fonction barrière

$$\Phi(X) = -\ln \det X = -\ln \prod_{i=1}^{n} \lambda_{Xi} = -\sum_{i=1}^{n} \ln(\lambda_X)_i$$

est à mettre en parallèle avec $\Phi(x) = -\ln \prod_{i=1}^{n} x_i = -\sum_{i=1}^{n} \ln x_i$

G.3 METHODES DE POINT INTERIEUR

G.3.1 Description des méthodes

On peut établir une classification des méthodes de point intérieur en programmation semidéfinie identique à celle déjà vue pour le cas linéaire (méthodes primales, duales, primales-duales, à départ admissible ou pas, etc.). Ici aussi, on constate que les méthodes primales-duales sont les plus efficaces, et sont les seules à être implémentées en pratique.

Les méthodes de mise à l'échelle affine, de suivi de chemin et de réduction de potentiel se transposent intégralement à la programmation semidéfinie. Ainsi, les algorithmes décrits au point C-C.3.3 peuvent être suivis à la lettre, à condition de remplacer x par X, s par S.

Le calcul de μ_k s'effectue selon l'équation définie plus haut, et le nouveau système (NWT) à résoudre se présente sous la forme suivante

$$
\begin{aligned}
&A_i \bullet \Delta X = 0 \ , \ i = 1,2,\ldots,m \\
&\sum_{i=1}^{ù} \Delta y_i A_i + \Delta S = 0 \\
&S\Delta X + X\Delta S = \tau I - XS
\end{aligned}
$$

Il faut remarquer deux différences importantes par rapport au système (NWT) du cas linéaire. D'une part, bien que linéaire, ce système n'est pas sous la forme Ax=b, et ne peut donc plus se résoudre directement par les méthodes classiques applicables aux systèmes linéaires (Cholevksy, gradient conjugué, etc.). Les logiciels devront donc le transformer avant de lui appliquer une de ces méthodes.

D'autre part, il existe une autre difficulté plus importante, parce qu'elle se situe au niveau conceptuel. Les matrices X et S appartenant à S^n, il faut que ΔX et ΔS soient symétriques pour que X+ΔX et S+ΔS appartiennent également à S^n. Or, comme le produit XS n'est pas symétrique, on constate que la composante ΔX fournie par le système ne le sera pas non plus (ΔS l'est à cause de la seconde équation). Une façon simple de surmonter ce problème est de rendre symétrique la composante ΔX après chaque itération (par exemple en prenant $\frac{1}{2}(\Delta X+\Delta X^T)$). Cette façon de procéder est connue sous le nom de méthode XZ.

Certaines auteurs préfèrent cependant modifier directement le système d'équations pour qu'il fournisse un ΔX symétrique. A cet effet, on définit un *opérateur de symétrisation* H_P pour toute matrice non singulière P selon *[ZHA95]*

$$H_P(M) = \tfrac{1}{2}(PMP^{-1} + (PMP^{-1})^T)$$

On reformule alors le système de Newton selon

$$
\begin{aligned}
&A_i \bullet \Delta X = 0 \ , \ i = 1,2,\ldots,m \\
&\sum_{i=1}^{ù} \Delta y_i A_i + \Delta S = 0 \\
&H_P(S\Delta X + X\Delta S) = \tau I - H_P(XS)
\end{aligned}
$$

ce qui lui confère une solution donnant un pas (ΔX, Δy, ΔS) totalement symétrique. Parmi les choix possibles pour P, citons P=I (méthode AHO, *[ALI96]*), $P=S^{½}$ *[ZHA95]* ou encore $P^TP = X^{-1}$ ou S (Monteiro). Aucune de ces méthodes ne s'est encore affirmée comme clairement préférable aux autres, et de nombreux chercheurs étudient d'autres possibilités.

G.3.2 Complexité

Nous terminons ce volet théorique par quelques commentaires sur la complexité algorithmique des méthodes de point intérieur appliquées à la programmation semidéfinie *[ALI96]*.

Les meilleurs algorithmes font état d'une complexité de pire cas en nombre d'itérations égale à

$$\sqrt{n}\,\log\frac{1}{\varepsilon}$$

Il est assez surprenant de constater qu'il s'agit de la même borne que pour la programmation linéaire, alors que le problème possède beaucoup plus de variables (l'espace S^n comporte n(n+1)/2 dimensions contre n dimensions pour $\Re^n$).

Des test numériques ont également mis en évidence le fait que la dépendance réelle du nombre d'itérations avec n est beaucoup plus faible, du type $n^{¼}$ ou log n, comme dans le cas linéaire.

H. APPLICATIONS

La formulation de problèmes sous forme d'un programme semidéfini a été beaucoup étudiée, bien avant le développement des méthodes de point intérieur. On distingue les deux cas suivants :

- La programmation semidéfinie permet de modéliser le problème exactement. L'optimum fourni par les méthodes de point intérieur est le véritable optimum du problème.
- La programmation semidéfinie permet de modéliser une relaxation d'un problème. L'optimum fourni par les méthodes de point intérieur est une borne inférieure du véritable optimum du problème (dans le cas d'une minimisation).

Nous allons à présent décrire quelques applications dans ces deux catégories *[VAN96, ALI95, HEL96]*.

H.1 MODELISATION EXACTE

H.1.1 Valeurs propres et norme matricielle

De nombreux problèmes de minimisation ou de maximisation ayant trait aux valeurs propres d'une combinaison linéaire de matrices peuvent se mettre sous la forme d'un programme semidéfini.

Le plus simple de ces problèmes consiste à minimiser la valeur propre maximale de la matrice A, définie comme une combinaison linéaire des matrices A_i auxquelles on ajoute un terme constant A_0. On peut le formuler selon

$$\begin{aligned}&\min\ t\\&\text{avec } tI - \left(A_0 + \sum_{i=1}^{m} y_i A_i\right) \geq 0\end{aligned}$$

On reconnaît ici un programme semidéfini dans sa forme duale. On peut justifier sa validité en remarquant que la contrainte $M \leq kI$, c'est-à-dire $kI - M$ semidéfinie positive, est équivalente à imposer à la valeur propre maximale de M d'être inférieure à k. En effet, si v est un vecteur propre de M, on a $Mv = \lambda v$ et donc $v^T(kI\text{-}M)v = k\, v^Tv - v^TMv = (k\text{-}\lambda)\, v^Tv \geq 0$ puisque (kI-M) est semidéfinie positive, d'où $\lambda \leq k$. Comme on a ici $tI - A \geq 0$, on en déduit que le programme va donc bien minimiser la valeur propre maximale de A, combinaison linéaire des A_i.

Beaucoup de problèmes en relation avec celui-ci peuvent être formulés. Citons entre autres :

- Minimiser la somme des r plus grandes valeurs propres de A (toujours égale à une combinaison linéaire des A_i comme ci-dessus)
- Minimiser la somme *pondérée* des r plus grandes valeurs propres de A, c'est-à-dire $m_1\lambda_1 + ... + m_r\lambda_r$, à condition que $m_1 \geq m_2 \geq ... \geq m_k > 0$.
- Minimiser la somme des r plus grandes valeurs propres de A prises en valeur absolue
- Minimiser la norme spectrale de A

H.1.2 Approximation logarithmique de Tchebycheff

Imaginons que nous cherchions à résoudre *approximativement* le système d'équations $a_i^Tx = b_i$ pour i=1,2,...,m, comme cela se pratique par exemple lors de la recherche des paramètres d'une régression.

Le terme *approximativement* a été utilisé parce que la résolution exacte est impossible (trop de contraintes) . On peut dès lors choisir plusieurs critères pour déterminer la meilleure solution. Le plus courant est la minimisation de la somme des carrés des écarts $(a_i^Tx - b_i)$; cependant, on peut aussi vouloir minimiser l'écart maximal (en valeur absolue), selon

$$\min \ \max_i \left| a_i^T x - b_i \right|$$

Cela peut se mettre sous la forme d'un simple programme linéaire, avec des contraintes du type $-t \leq a_i^T x - b_i \leq t$ et une minimisation de t. Cependant, il existe des applications où les b_i sont des quantités s'exprimant dans une unité de puissance. Dans ce cas, il est préférable d'un point de vue purement physique de minimiser le maximum des écarts entre les logarithmes de $a_i^T x$ et b_i, ce qui se traduit par

$$\min \ \max_i \left| \log\left(a_i^T x\right) - \log\left(b_i\right) \right|$$

On transforme alors les contraintes en

$$-\log\ u \leq \log \frac{a_i^T x}{b_i} \leq \log u \text{ , d'où } \frac{1}{t} \leq \frac{a_i^T x}{b_i} \leq t$$

Ceci peut s'écrire sous la forme du programme semidéfini suivant

$$\min t$$

$$\begin{bmatrix} t - a_i^T x / b_i & 0 & 0 \\ 0 & a_i^T x / b_i & 1 \\ 0 & 1 & t \end{bmatrix} \geq 0$$

Le premier terme de la diagonale est responsable de l'inégalité $a_i^T x / b_i \leq t$, tandis que $a_i^T x / b_i \geq 1/t$ se traduit par la matrice 2x2 inférieure droite. En effet, pour une matrice 2x2, le fait d'être semidéfiine positive implique la non-négativité du déterminant, d'où on déduit $(a_i^T x / b_i)\ t - 1 \geq 0$ et finalement l'inégalité recherchée. On remarque au passage la puissance supplémentaire de modélisation offerte par la programmation semidéfinie par rapport à la programmation linéaire (utilisation de 1/t, par exemple).

H.1.3 Problèmes géométriques avec formes quadratiques

Le fait d'optimiser sur des matrices (objets à deux dimensions) se prête particulièrement bien à la résolution de problèmes faisant intervenir des formes quadratiques, d'équation générale $x^T A x + b^T x + c = 0$. En particulier, la condition nécessaire et suffisante pour que cette forme décrive un ellipsoïde est précisément $A > 0$ (ou $A \geq 0$ si on accepte la dégénérescence). Citons à titre d'exemple, le problème consistant à trouver la plus petite sphère contenant un ensemble d'ellipsoïdes donnés. Un autre problème de ce type, celui de la séparation de points par un ellipsoïde, a été étudié en détail et implémenté sur ordinateur (cet exemple sera développé plus en détail au point I).

H.1.4 Programmation quadratique convexe

Soit la contrainte $(Ax+b)^T(Ax+b) - c^T x - d \leq 0$. Il s'agit en fait de la forme générale d'une contrainte convexe quadratique. On montre qu'elle est convexe et qu'elle peut s'écrire selon

$$\begin{bmatrix} I & Ax + b \\ (Ax + b)^T & c^T x + d \end{bmatrix} \geq 0$$

Ceci implique qu'un programme quadratique convexe peut se mettre sous la forme d'un programme semidéfini. Cependant, en utilisant la théorie de Nesterov et Nemirovsky, il est possible de concevoir des algorithmes spécifiques à ce cas particulier, plus efficaces que le passage par la formulation semidéfinie.

H.1.5 Autres applications

La résolution exacte d'un problème par la programmation semidéfinie intervient encore dans bien d'autres domaines, en particulier en contrôle et en optimisation structurelle. Il serait trop long de décrire les applications en question ou d'effectuer les développements conduisant aux formulations semidéfinies ; contentons nous de mentionner les deux exemples typiques suivants :

- Conception d'un contrôleur de bruit actif (c'est-à-dire neutralisant le bruit ambiant par l'émission d'un contre-bruit). Les données sont ici des matrices contenant les réponses impulsionnelles finies entre les différents éléments du système. On pourra trouver plus d'informations dans *[OLK96]*.
- Minimisation de la constante de temps dominante dans un circuit formé de résistances et capacités. On peut également minimiser la consommation du circuit ou son aire sous la contrainte d'une constante de temps maximale permise.

H.2 RELAXATIONS

H.2.1 Programmation quadratique non-convexe

Le problème le plus général d'optimisation quadratique s'écrit

$$\begin{aligned}&\min\ f_0(x)\\&\text{avec } f_i(x) \leq 0 \text{ et } i = 1,2,\ldots,m\end{aligned}$$

où $f_i(x) = x^TA_ix + 2b_i^Tx + c_i$ pour $i = 0,1,...,m$. Comme on n'impose pas aux matrices A_i d'être définies positives, ces contraintes sont non convexes, ce qui rend le problème NP-difficile. Signalons par exemple que tous les problèmes à objectif polynomial et contraintes polynomiales peuvent se mettre sous cette forme.

Il est intéressant de disposer d'un moyen de calcul raisonnablement rapide d'une borne inférieure de la valeur optimale de ce type de problème, par exemple lors de l'application d'un algorithme de résolution basé sur le *branch-and-bound*.

Posons $U = xx^T$. Le problème peut se réécrire selon

$$\begin{aligned}&\min\ U \bullet A_0 + 2b_0^Tx + c_0\\&\text{avec } U \bullet A_i + 2b_i^Tx + c_i \leq 0\\&\text{et } U - xx^T = 0,\ i = 1,2,\ldots,m\end{aligned}$$

Relaxons $U - xx^T = 0$ par $U - xx^T \geq 0$, qui peut encore s'écrire sous la forme

$$\begin{bmatrix} U & x^T \\ x & 1 \end{bmatrix} \geq 0$$

En combinant avec le problème ci-dessus, on arrive bien à une relaxation semidéfinie du problème quadratique non-convexe. Ce procédé de relaxation est appelée convexification, et a été découvert par Shor *[SHO87]*. Il est intéressant de noter qu'on peut retransformer le problème relaxé en son problème original par l'adjonction d'une contrainte sur le rang de U, à savoir rang(U) = 1. En effet, on montre dans ce cas que l'on a forcément $U = xx^T$, ce qui redonne bien la forme originale.

H.2.2 Optimisation combinatoire

Les relaxations linéaires sont usuelles en optimisation combinatoire, on les emploie principalement au cours de la résolution de problèmes par *branch-and-bound*. La programmation semidéfinie fournit également de bonnes relaxations, dont on peut souvent prouver la supériorité par rapport aux relaxations linéaires. Citons quelques exemples de telles relaxations.

a) Le problème MAX-CUT

Le problème MAX-CUT, issu de la théorie des graphes, consiste à trouver une partition des sommets d'un graphe complet à arêtes valuées en deux ensembles, de telle façon que la somme des poids des arêtes joignant les deux ensembles soit maximale. Cette partition est appelée coupe (*cut*) et la somme à maximiser est appelée poids de la coupe.

Soit A la matrice fournissant les poids des arêtes (a_{ij} représentant le poids de l'arête joignant le sommet i au sommet j). On définit L = diag(Ae) - A, la *matrice Laplacienne* du graphe. Si on représente la coupe par le vecteur $x \in \{-1,+1\}^n$, où x_i est égal à +1 ou -1 selon son appartenant à l'un ou l'autre des deux ensembles, on vérifie aisément que le poids total de la coupe vaut $\frac{1}{4} x^TLx$.

On poursuit en posant X = ¼ $x^T x$, pour aboutir à la relaxation suivante (on a appliqué le procédé de convexification décrit plus haut).

$$\begin{aligned} &\max\ L \bullet X \\ &\text{avec } \operatorname{diag}(X) = \frac{1}{4} e \qquad \text{(GWR)} \\ &\text{et } X \geq 0 \end{aligned}$$

La contrainte diag(X) = ¼ e impose en fait $x_i^2 = 1$, ce qui restreint bien le vecteur x à $\{-1,+1\}^n$. Cette relaxation est très intéressante en vertu de la propriété suivante, démontrée par Goemans et Williamson lorsque les poids des arêtes sont non-négatifs *[GOE95]*

> La valeur optimale de (GWR) est au plus égal à 1,14 fois celle du problème original non relaxé

On dispose donc de la valeur maximale du poids de la coupe à 14% près, en temps polynomial, alors que le problème est NP-difficile.

b) Autres problèmes

Citons quelques autres problèmes d'optimisation combinatoire admettant de bonnes relaxations semidéfinies :

- Le problème de la partition d'un graphe à arêtes valuées à n sommets en k sous-ensembles de tailles s_1, s_2, ..., s_k, où la somme des s_i vaut n. Il s'agit d'une généralisation du problème MAX-CUT, le critère à maximiser étant la somme des poids des arêtes dont les extrémités appartiennent à des sous-ensembles différents.
- Trouver une clique de taille maximale dans un graphe (rappelons qu'une clique est un ensemble de sommets reliés deux à deux). Ce problème est équivalent à celui consistant à trouver un ensemble stable maximal (un ensemble de sommets non-reliés deux à deux). Signalons encore que les relaxations semidéfinies de ces deux problèmes sont exactes pour une certaine catégorie de graphes, appelés graphes *parfaits*.
- Knapsack (sac à dos) quadratique. Etant donné n objets de poids p_i, on cherche à en trouver un sous-ensemble dont le poids total est inférieur à une limite prédéfinie P tout en maximisant un objectif quadratique. Celui-ci associe un profit c_{ij} à chaque paire d'objets i et j présents dans le sac à dos (rappelons que le knapsack ordinaire associe simplement un profit à chaque objet individuellement).

I. IMPLEMENTATION D'UNE APPLICATION A LA PROGRAMMATION SEMIDEFINIE

I.1 PROBLEME POSE

I.1.1 Description du problème : analyse discriminante par ellipsoïdes

Soit N individus répartis a priori en k sous-ensembles S_1, S_1, S_3, ..., S_k. L'ensemble S_i contient n_i individus, et l'union de ces sous-ensembles est appelée S. Chaque individu est décrit par un vecteur réel à d dimensions, c'est-à-dire par un point de $\Re^d$. Chacune des d composantes représente la mesure d'une caractéristique donnée de l'individu.

On se propose d'effectuer une analyse discriminante de ces données en utilisant des ellipsoïdes. Plus précisément, cela signifie que l'on va chercher k ellipsoïdes (à d dimensions) séparant les sous-ensembles S_i, c'est-à-dire tels que le $k^{ème}$ ellipsoïde contiendra tous les individus de S_k et aucun autre. Comme ce problème admettra le plus souvent une infinité de solutions, on essaiera d'obtenir les ellipsoïdes les plus sphériques possibles.

I.1.2 Signification et application du problème

Ce problème intervient principalement lorsqu'on veut réaliser une classification dans un groupe d'individus, selon le processus suivant :

- Définition de k classes d'individus que l'on souhaite distinguer
- Recherche d'un nombre suffisant d'individus représentatifs de chaque classe
- Calcul à l'aide des individus représentatifs des k ellipsoïdes discriminant les k classes
- Utilisation des ellipsoïdes calculés pour classer un individu quelconque dans une des k classes.

Cette procédure est particulièrement recommandée lorsque les mesures sur les individus sont faciles à réaliser alors que leur interprétation est difficile (par exemple à cause de la nécessité de faire appel à un expert). On peut alors se contenter de classer un faible nombre d'individus, puis d'utiliser les ellipsoïdes pour classer les individus restants.

La recherche d'un ellipsoïde le plus sphérique possible peut s'interpréter comme le choix d'essayer d'accorder dans la mesure du possible la même importance aux d caractéristiques des individus.

Ce problème a été étudié bien avant le développement des méthodes de point intérieur pour la programmation semidéfinie (voir *[ROS65]*). Citons les quelques exemples d'applications pratiques suivants :

- Reconnaissance d'un caractère (classe) à partir d'une représentation numérisée (matrice de coefficients - les caractéristiques - correspondant à des niveaux d'intensité de gris des pixels d'une image numérique)
- Prédiction du type de temps à venir (classe) en fonction de la pression atmosphérique mesurée à divers endroits (caractéristiques)
- Reconnaissance des déficiences cardiaques (classe) à l'aide de points caractéristiques de son électrocardiogramme (tensions prises à certains instants spécifiques).

I.1.3 Modélisation par la programmation semidéfinie

Nous ne résoudrons pas directement ce problème par la programmation semidéfinie, mais nous allons le décomposer en k sous-problèmes, à savoir la recherche des k ellipsoïdes séparément. Pour cela, nous nous intéressons au problème suivant :

Soit $X = \{ x^1, x^2, ..., x^r \}$ et $Y = \{ y^1, y^2, ..., y^s \}$ deux ensembles de points de $\Re^d$. On va chercher un ellipsoïde E contenant tous les points de X et aucun point de Y.

Il est évident que l'on peut effectuer l'analyse discriminante par ellipsoïdes à l'aide de plusieurs résolutions successives de ce problème. En effet, pour déterminer le $i^{\text{ème}}$ ellipsoïde, il suffit de poser $X = S_i$ et $Y = S \setminus S_i$. On a donc $r = n_i$ et $s = N - n_i$.

Un ellipsoïde est une forme quadratique, d'équation générale $x^T Ax + b^T x + c \leq 0$, où A est une matrice symétrique (semi)définie positive. La programmation semidéfinie va nous permettre d'exprimer cette condition sur A par $A \geq 0$. Les contraintes mentionnées ci-dessus (appartenance et non appartenance à E) s'écrivent donc $x^{iT}Ax^i + b^T x^i + c \leq 0$ et $y^{jT}Ay^j + b^T y^j + c > 0$ pour i = 1, 2, ..., r et j = 1, 2, ..., s. Celles-ci sont linéaires en les coefficients de A, b et c.

Il nous reste à optimiser la sphéricité de l'ellipsoïde. On sait que les valeurs propres de A sont proportionnelles aux longueurs des axes de l'ellipsoïde, ce qui nous amène à introduire une variable γ et les inégalités suivantes :

$$I \leq A \leq \gamma I$$

L'inégalité de gauche va borner inférieurement les valeurs propres de A par 1, tandis que celle de droite va les borner supérieurement par γ. On comprend dès lors aisément qu'il suffira de minimiser γ pour minimiser le rapport de la longueur du plus grand axe à celle du plus petit, ce qui va bien imposer l'ellipsoïde le plus sphérique possible. Remarquons que la contrainte $A \geq I$ rend $A \geq 0$ superflue.

Le programme semidéfini qu'on a à résoudre s'écrit finalement

$$\begin{aligned}
&\min\ \gamma \\
&\text{avec } x^{iT}Ax^i + b^T x^i + c \leq 0,\ i = 1,2,\ldots,r \\
&y^{jT}Ay^j + b^T y^j + c > 0,\ j = 1,2,\ldots,s \\
&I \leq A \leq \gamma I
\end{aligned}$$

Il ne nous reste plus qu'à le transformer dans une des deux formes standard (primale ou duale). Il nous a semblé plus simple de passer à la forme duale. A cet effet, nous définissons ½ d(d+1) variables a_{ij} pour les coefficients de A (la matrice étant symétrique, il suffit de définir les valeurs des coefficients situés au dessus de la diagonale), d variables b_i pour ceux de b, une variable pour c et une autre pour γ. On aboutit alors, après manipulations, à la formulation suivante, qui correspond bien à la forme duale standard

$$\begin{aligned}
&\min\ \gamma \\
&\text{avec } \sum_{1 \leq i \leq j \leq d} a_{ij} A_{ij} + \sum_{1 \leq i \leq d} b_i B_i + cC + \gamma\Gamma \leq D
\end{aligned}$$

Les matrices A_{ij}, B_i, C et Γ sont des matrices carrés de dimension r+s+2d définies en fonction des donnés du problème. On retrouvera leur expression complète en annexe.

I.2 IMPLEMENTATION

I.2.1 Description de la bibliothèque utilisée

Plutôt que de partir de zéro pour développer cette implémentation, nous avons préféré tirer profit de l'expérience des chercheurs établis dans le domaine et décidé d'utiliser une librairie de fonctions relatives à la programmation semidéfinie. Plusieurs de ces librairies ont été développées très récemment, et il est généralement possible de les télécharger à l'aide du réseau Internet.

Parmi les implémentations existantes, nous avons choisi SdpPack par Alizadeh, Haeberly, Nayakkankuppam et Overton, principalement en raison du fait qu'elle était écrite dans le langage Matlab. Celui-ci présente de nombreux avantages, tels que des fonctions évoluées de visualisation et de manipulation matricielle (y compris les matrices creuses et la factorisation de Cholevsky), permet un débogage et une mise au point rapides, réalise une allocation totalement dynamique de la mémoire, son seul inconvénient résidant dans sa vitesse d'exécution plus faible que celle d'un langage traditionnel. Notre but n'étant pas de rechercher la performance en temps de calcul, cette solution nous semble convenir parfaitement.

La procédure principale de cette librairie se nomme `sdp`. Elle permet d'optimiser le programme semidéfini standard décrit dans le volet théorique à l'aide d'une méthode de point intérieur à départ non-admissible. Elle accepte en entrée les paramètre suivants :

- `A`, `b` et `C` constituent les données du problème. `A` contient en fait les matrices A_i rangées sous forme de lignes.
- `blk` est une vecteur indiquant la structure diagonale des matrices A_i. En effet, la plupart du temps, les matrices A sont très creuses et formées de la juxtaposition de blocs sur la diagonale. Ce vecteur donne la taille de ces différents blocs, afin d'éviter des calculs inutiles (en effet, si l'on veut par exemple effectuer le produit de deux matrices diagonales par blocs, il suffit d'effectuer les produits des blocs entre eux, puis de les rassembler sur la diagonale de la matrice résultat). Dans le cas de notre formulation, les matrices sont effectivement très creuses, et `blk` est égal à [1, 1, ..., 1, d, d] (où le nombre de 1 est égal à r+s).
- `X`, `y`, `Z` doivent contenir l'itéré de départ. Signalons que les auteurs ont préféré suivre une convention différente de la nôtre et nommer Z au lieu de S la matrice d'écart du problème dual. Il existe une procédure (`initvars`) fournissant une valeur par défaut (`X` = `Z` = matrice identité multipliée par une constante, `y` = 0)
- D'autres paramètres déterminent le comportement de l'algorithme. Citons `maxit` (nombre maximal d'itérations autorisé), `tau` (correspond au paramètre α, coefficient réducteur appliqué au pas de Newton maximal), la valeur de l'erreur résiduelle (sur l'admissibilité et l'optimalité) qu'il faut atteindre pour arrêter l'algorithme et `bndtol`, la norme maximale autorisée pour les matrices X et Z (si l'on dépasse cette valeur, l'algorithme conclut à une divergence des itérés due à un problème impossible et s'arrête).

En sortie, `sdp` fournit

- Les valeurs optimales de `X`, `y` et `Z`
- Le nombre d'itérations effectuées et, pour chacune de ces itérations, la valeur du saut de dualité, de l'admissibilité (primale et duale) et de l'objectif (primal et dual)
- Une variable indiquant la raison de l'arrêt de l'algorithme (optimum atteint, problème de convergence, instabilité numérique, nombre maximal d'itérations atteint ou divergence des itérés).

Pour plus de précisions sur cette librairie (qui contient entre autres une fonction spécialisée dans l'optimisation de problèmes du type MAX-CUT, nous renvoyons au manuel de référence *[ALI97]*.

I.2.2 Description du programme

Le programme que nous avons écrit consiste donc à transformer les données brutes concernant les individus à classer en données pour la procédure `sdp`, puis à interpréter les résultats fournis par cette dernière en termes du problème initial.

La procédure principale se nomme `elliSep`, elle accepte comme arguments deux matrices reprenant les vecteurs des caractéristiques des individus de X et Y (rangés en colonne). Elle fournit en sortie les paramètres (`A`,`b`,`c`) de l'ellipsoïde séparateur et la valeur optimale de γ.

On peut alors tester la présence d'autres points à l'intérieur de l'ellipsoïde défini par (`A`,`b`,`c`), à l'aide de la fonction `testIn`, et en déduire leur appartenance ou non à la classe correspondante.

Dans le cas bidimensionnel (d=2), il est possible de visualiser les points à séparer (à l'aide de la fonction `pntPlot`) ainsi que l'ellipsoïde séparateur (fonction `elliPlot`). Nous utiliserons ces possibilités dans nos tests afin de valider le programme.

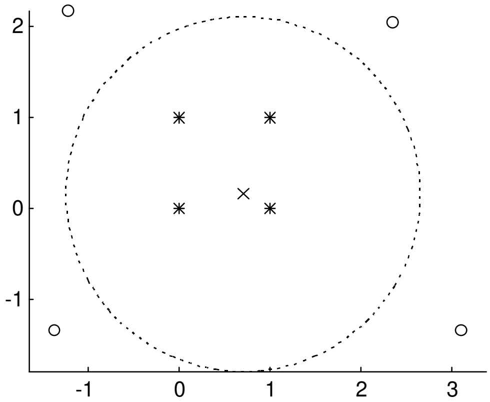
Signalons que le cœur du programme est constitué par la fonction `prepareA`, qui calcule les matrices A_i telles que définies plus haut en fonction des caractéristiques des individus à séparer.

Pour plus de détails, nous renvoyons au code source commenté, disponible en annexe.

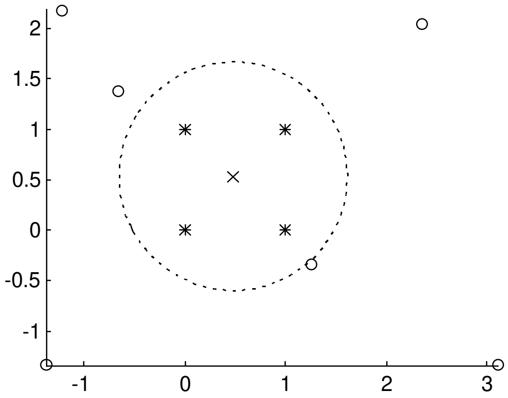
I.3 RESULTATS

I.3.1 Tests à deux caractéristiques et visualisation

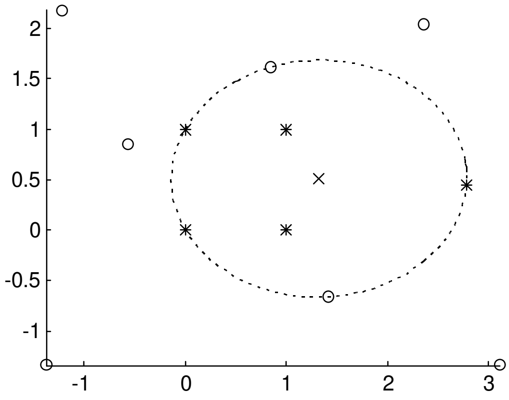
Nous avons utilisé les possibilités de visualisation afin de valider notre implémentation. Nous illustrons ici les situations survenant le plus couramment dans le cas où on a à séparer deux ensembles de points. Les points à inclure sont représentés par des astérisques, les points à exclure par des cercles, le centre de l'ellipse étant indiqué par une croix.



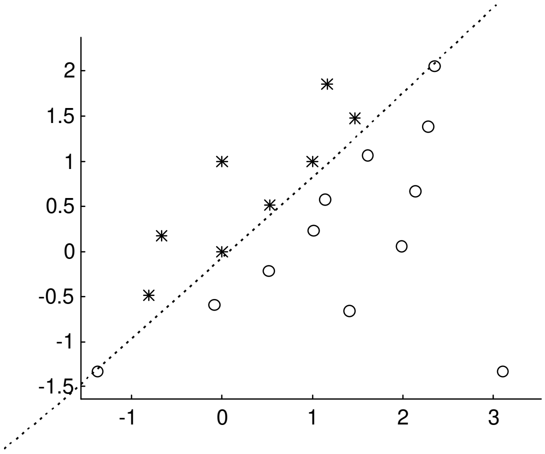
Dans cette situation, il existe bien une ellipse capable de discriminer nos deux ensembles de points. De plus, on constate que l'algorithme a fourni un cercle, à cause du critère de sphéricité introduit dans l'objectif. Notons que dans ce cas simple, il existe plusieurs cercles séparant les points. Comme aucun élément de l'objectif du programme semidéfini ne permet de distinguer des cercles de rayons différents, l'algorithme nous fournit une solution parmi celles qui sont possibles (cette solution dépendra en fait des itérés initiaux et des paramètres de l'algorithme).



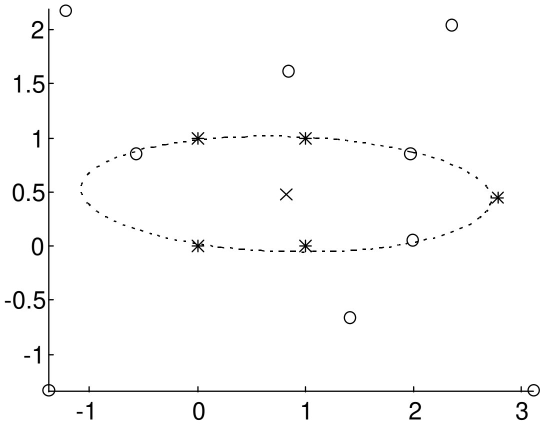
Si l'on ajoute un point à l'intérieur du cercle précédent, on constate que l'algorithme fournit un nouveau cercle, plus petit afin de séparer correctement le point nouvellement introduit.



Dans cette situation plus complexe, il n'existe plus de cercle capable de discriminer nos ensembles. L'algorithme fournit néanmoins l'ellipse séparatrice la plus sphérique possible. Notons que la solution est à présent unique : il n'existe plus qu'une seule ellipse de sphéricité maximale séparant correctement les points. C'est en fait cette situation qui se produira le plus souvent dans des situations de classification réelles.



Dans cette configuration particulière, les points sont linéairement séparables. L'algorithme fournit alors un cercle de rayon élevé, approchant une droite au niveau de la zone où se situent les points.



Enfin, dans cette situation, il n'existe plus d'ellipse capable d'effectuer une discrimination complète. L'algorithme fournit néanmoins une « bonne » solution (en tentant de minimiser les écarts résiduels par rapport à l'admissibilité).

I.3.2 Test avec les *Iris de Fisher*

Afin de comparer l'efficacité des méthodes d'analyse discriminante existantes, Fisher a constitué un ensemble test de données décrivant trois espèces d'iris (Setosa, Versicolor et Virginica). Pour chacune de ces espèces, on dispose des caractéristiques de 50 individus. Ces caractéristiques sont au nombre de quatre : longueur des pétales, largeur des pétales, longueur des sépales et largeur des sépales.

Cette collection de données est un standard dans le domaine de l'analyse discriminante et est réputée difficile à séparer par des méthodes classiques. A titre d'exemple, la Figure I-1 présente un graphique répertoriant les individus de deux espèces (cercles et croix) selon trois des quatre caractéristiques disponibles. On y perçoit aisément la complexité du problème.

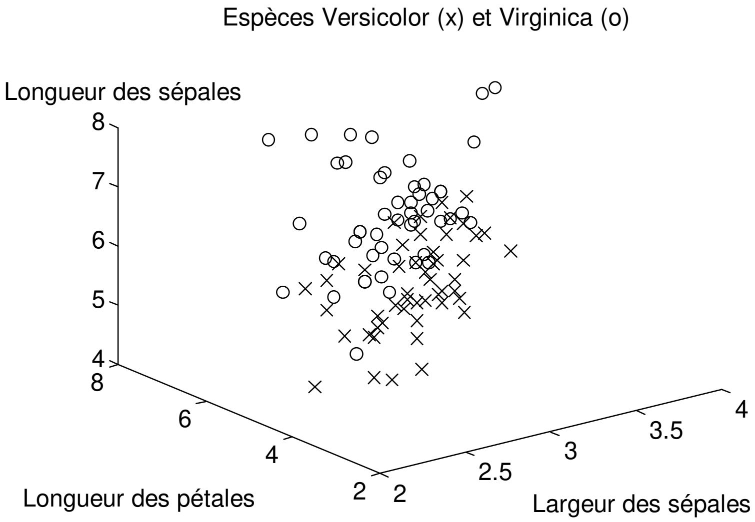


Figure I-1 - Deux ensembles d'*Iris de Fisher* selon trois de leurs caractéristiques

Nous nous trouvons donc dans le cas où k=3 et d=4. Afin de tester l'efficacité de notre procédure de discrimination, nous avons effectué une classification sur une partie des individus seulement (ensemble d'apprentissage), et utilisé les données restantes (ensemble de validation) pour tester la validité des classes ellipsoïdales obtenues. Cette façon de procéder est connue sous le nom de validation croisée (*cross-validation*). Le Tableau 4 résume les résultats obtenus (la première colonne mentionne le pourcentage de données utilisées pour l'apprentissage et la validation, les autres chiffres sont les pourcentages d'appartenance des données de validation aux différentes classes)

Apprentissage / Validation	**Reconnue comme ...**	**Espèce 1**	**Espèce 2**	**Espèce 3**
20% / 80%	Espèce 1	100	0	0
	Espèce 2	0	90	27
	Espèce 3	0	7	100
40% / 60%	Espèce 1	100	0	0
	Espèce 2	0	90	6
	Espèce 3	0	6	98
60% / 40%	Espèce 1	100	0	0
	Espèce 2	0	100	10
	Espèce 3	0	5	95
80% / 20%	Espèce 1	100	0	0
	Espèce 2	0	100	0
	Espèce 3	0	0	100

Tableau 4 - Performance de l'analyse discriminante par ellipsoïdes sur les *Iris de Fisher*

- Notons tout d'abord que la somme des pourcentages d'une ligne ne doit pas forcément être égale à cent. En effet, lorsque l'apprentissage est imparfait, il se peut qu'un individu appartienne à plusieurs ellipsoïdes (ou à aucun d'entre eux), puisque ceux-ci ont été déterminés indépendamment les uns des autres.

- L'espèce 1 est facile à discriminer des deux autres. En effet, il n'y a eu d'erreur concernant cette espèce à aucun moment. Les espèces 2 et 3 sont plus difficiles à distinguer
- Plus on augmente la taille de l'ensemble d'apprentissage (c'est-à-dire plus on diminue celle de l'ensemble de validation) et meilleurs sont les résultats. Ceci s'explique par le fait que l'on fournit à l'algorithme de classification plus d'individus, y compris certains situés aux limites du domaine, ce qui permet d'affiner la délimitation des ellipsoïdes et d'améliorer la classification des cas extrêmes.
- L'algorithme réalise un score parfait avec 20 % de données de validation. Signalons qu'avec un ensemble d'apprentissage égal à 100 % des données, l'algorithme parvient encore à séparer les trois espèces, ce qui implique que l'ensemble des *Iris de Fisher* est complètement séparable par des ellipsoïdes.

I.3.3 Etude de la complexité empirique

Nous avons mesuré le nombre d'itérations et d'opérations en virgule flottante (*flops*) effectués par l'algorithme pour diverses tailles des données dans le cas du problème des *Iris de Fisher*. Les résultats sont résumés dans les deux graphiques suivants :

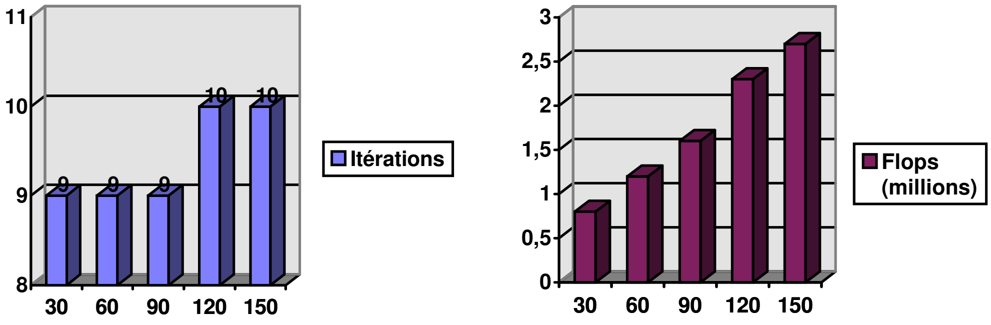


Tableau 5 - Nombre d'itérations et de flops pour diverses tailles du problème

On constate que le nombre d'itérations effectuées pour atteindre la précision demandée ne varie pratiquement pas en fonction de la taille du problème. Le nombre de flops effectués, quant à lui, progresse linéairement avec le nombre de points à discriminer. Le comportement polynomial attendu est donc clairement apparent. Il est plus favorable que celui prévu par la théorie (croissance du nombre d'itérations de l'ordre de la racine carrée de la taille des données).

I.4 DISCUSSION

- L'utilisation d'un programme semidéfini a permis la résolution d'un problème intraitable par la programmation linéaire
- L'implémentation d'un algorithme utilisant une méthode de point intérieur en programmation semidéfinie peut se faire relativement aisément étant donné la disponibilité de librairies dédiées à cet effet
- La méthode d'analyse discriminante par ellipsoïde que nous avons proposée s'est révélée particulièrement efficace puisqu'elle a réussi à discriminer la totalité des *Iris de Fisher*, pourtant réputés constituer un ensemble difficile à séparer.
- Le caractère polynomial de la méthode et la faible croissance du nombre d'itérations en fonction du nombre d'individus à séparer autorisent l'application de cette méthode à de grands ensembles d'individus

J. CONCLUSIONS

En publiant le premier article décrivant une méthode de point intérieur appliquée à la programmation linéaire en 1984, Karmarkar n'imaginait certainement pas, malgré le caractère novateur de ses résultats, qu'il allait déclencher une si longue et si fructueuse série de recherches et de découvertes dans le domaine pourtant très étudié de l'optimisation.

Nous espérons, par le biais de ce travail, avoir réussi à faire découvrir au lecteur ce domaine passionnant, et, chose plus importante, avoir dépassé le stade de la simple présentation descriptive pour lui apporter une compréhension profonde de ces méthodes, de leur justification et des concepts sous-jacents.

Le domaine a beaucoup évolué, et les méthodes les plus récentes (et les plus efficaces) n'ont plus grand chose à voir avec leur ancêtre de 1984. Rappelons que l'état de l'art - en 1997 - consiste à employer une méthode primale-duale à départ non admissible, agrémentée de la technique de Mehrotra, comme c'est le cas dans la plupart des logiciels disponibles.

J.1 PROGRAMMATION LINEAIRE

On peut affirmer sans crainte que le domaine des méthodes de point intérieur pour la programmation linéaire a atteint à l'heure actuelle une certaine maturité, ce qui se traduit par l'apparition d'ouvrages de référence *[WRI96, TER96]* faisant suite aux innombrables articles de recherche déjà publiés. On trouve même dans *[TER96]* une théorie complète de la programmation linéaire entièrement basée sur les méthodes de point intérieur.

Il reste cependant encore quelques problèmes à résoudre, comme celui de l'analyse de sensibilité, du démarrage à chaud ou de l'étude de la parallélisation efficace des calculs.

Plusieurs chercheurs ainsi que quelques sociétés commerciales ont entrepris la réalisation de logiciels basés sur ces méthodes. Il ressort des tests numériques effectués que les codes publics et commerciaux sont comparables, compétitifs avec la méthode du simplexe pour des problèmes de taille moyenne et plus efficaces pour des problèmes de grande taille.

La théorie relative à ces méthodes est plus ardue que celle de l'algorithme du simplexe. De plus, bien que cela ne se remarque pas dans ce travail, il n'existe pas encore de véritable consensus quant aux notations et à la façon de présenter les problèmes. Ainsi, certains auteurs préfèrent baser leur théorie sur d'autres formes que le couple standard primal-dual présenté plus haut. Le domaine a donc encore besoin d'une certaine uniformisation dans la présentation et dans les notations.

J.2 PROGRAMMATION NON-LINEAIRE ET PROGRAMMATION SEMIDEFINIE

L'étude des méthodes pour la programmation non-linéaire (en particulier pour la programmation convexe) est à présent le centre d'intérêt de la communauté scientifique s'intéressant aux méthodes de point intérieur. Il est en effet possible d'adapter avec un effort relativement limité la plupart des méthodes existantes à des problèmes non-linéaires.

Comme on l'a vu au cours de la partie de ce travail consacrée à la programmation semidéfinie, de nombreux parallèles existent avec le cas linéaire, bien que certaines différences subsistent et rendent le problème plus complexe.

La programmation semidéfinie nous apparaît comme le domaine le plus prometteur, à cause des nombreuses applications qu'elle comporte, du nombre important de recherches qui lui sont consacrées et de l'existence d'implémentations efficaces. On peut dès lors s'attendre à de nombreuses découvertes et améliorations dans un avenir proche.

J.3 ANALYSE DISCRIMINANTE PAR ELLIPSOÏDES

La méthode d'analyse discriminante par ellipsoïdes que nous avons implémentée se base sur la programmation semidéfinie. Notons qu'il aurait été à la fois plus compliqué et inutile de vouloir implémenter à partir de zéro une méthode de résolution de ce problème. Nous avons préféré nous baser sur l'une des librairies de programmation semidéfinie existantes, ce qui nous a permis de réaliser et tester cette application en un temps relativement court.

Les résultats que nous avons obtenus sont très encourageants, en particulier en ce qui concerne la séparation de l'ensemble des *Iris de Fisher*, et nous pensons que cette méthode mérite d'être étudiée un peu plus en détail, par exemple sur des problèmes de plus grande taille.

J.4 REMARQUE FINALE

Etant donnés les résultats déjà obtenus et les perspectives d'évolution future, nous pensons pouvoir affirmer que les théoriciens et praticiens des méthodes de point intérieur peuvent être optimistes quant à l'avenir de leur discipline, tant du point de vue des développements théoriques que des applications.

K. REFERENCES ET BIBLIOGRAPHIE

K.1 RÉFÉRENCES

[ALI95] - F. ALIZADEH, Interior point methods in semidefinite programming with applications to combinatorial optimization, SIAM Journal on Optimization, 5 (1995), pp. 13-51.

[ALI96] - F. ALIZADEH, J.-P. A. HAEBERLY ET M. L. OVERTON, Primal-dual interior-point methods for semidefinite programming : Convergence rates, stability, and numerical results, Technical Report, Computer Science Department, Courant Institute of Mathematical Sciences, New York University, New York, N.Y., 1996.

[ALI97] - F. ALIZADEH, J.-P. A. HAEBERLY, NAYAKKANKUPPAM ET M. L. OVERTON, SdpPack 0.8 : User's guide, téléchargeable à partir du l'archive des méthodes de point intérieur (`http://www.mcs.anl.gov/home/otc/InteriorPoint/archive.html`)

[AND96] - E. D. ANDERSEN, J. GONDZIO, C. MESZAROS, X. XU, Implementation of interior-point methods for large scale linear programs, in Interior point methods of mathematical programming, T. Terlaky (Ed.), Applied Optimization series, Kluwer Academic Publishers, 1996.

[ANS90] - K. M. ANSTREICHER, On Long Step Path Following and SUMT for Linear and Quadratic Programming, Technical Report, Yale School of Management, Yale University, New Haven, CT, 1990.

[ARB91] - A. ARBEL, Exploring Interior-Point Linear Programming, Algorithms and Software, Foundations of Computing series, Research Reports and Notes, The MIT Press, Massachusetts, 1991.

[BOR87] - K.-H. BORGWARDT, The Simplex Method : A Probabilistic Analysis, Springer-Verlag, Berlin, 1987.

[DAN63] - G. B. DANTZIG, Linear Programming and extensions, Princeton University Press, Princeton, N.J., 1963.

[FIA68] - A. V. FIACCO ET G. P. MCCORMICK, Nonlinear Programming : Sequential Unconstrained Minimization Techniques, Wiley, New York, 1968. Réimprimé par SIAM Publications, 1990.

[FRE96] - R. M. FREUND ET S. MIZUNO, Interior Point Methods : Current Status and Future Directions, Optima, 51, October 1996, pp. 1-9.

[FRI55] - K. R. FRISCH, The logarithmic potential method of convex programming, Technical Report, University Institue of Economics, Oslo, Norway, 1955.

[GAR79] - M. R. GAREY ET D. S. JOHNSON, Computers and Intractability : A Guide to the Theory of NP-Completeness, W. H. Freeman, New York, 1979.

[GAY85] - D. GAY, Electronic Mail Distribution of Linear Programming Test Problems, Mathematical Programming Society COAL Newsletter, 1985.

[GOE95] - X. M. GOEMANS ET D. P. WILLIAMSON, Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming, J. ACM, Vol. 42 (1995), pp. 1115-1145.

[GOL56] - A. J. GOLDMAN ET A. W. TUCKER, Theory of linear programming, in Linear Equalities and Related Systems, H. W. Kuhn and A. W. Tucker eds., Princeton University Press, Princeton N.J., 1956, pp. 53-97

[GON94] - J. GONDZIO ET T. TERLAKY, A computational view of interior-point methods for linear programming, Technical Report 94-73, Faculty of Technical Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands, 1994.

[HEL96] - C. HELMBERG, F. RENDL, R. J. VANDERBEI ET H. WOLKOWICZ, An interior-point method for semidefinite programming, SIAM Journal on Optimization, 6 (1996), pp. 342-361.

[IBM79] - IBM MATHEMATICAL PROGRAMMING SYSTEM EXTENDED/370 (MPSX/370), Program Reference Manual, 4th edition, 1979.

[JAN43] - B. JANSEN, C. ROOS ET T. TERLAKY, An interior point approach to postoptimal and parametric analysis in linear programming, in Interior point methjods, Eövös Lorànd University, Department of Operations Research, H-1088 Budapest, Mùzeum krt. 6-8., Hungary, 1992.

[JAN95] - B. JANSEN, C. ROOS ET T. TERLAKY, A short survey on ten years interior point methods, Technical Report 95-45, Faculty of Technical Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands, 1995.

[KAR84] - N. KARMARKAR, A new polynomial-time algorithm for linear programming, Combinatorica, 4 (1984), pp. 373-395.

[KAR94] - G. KARYPIS, A. GUPTA ET V. KUMAR, A parallel formulation of interior-point algorithms, Technical Report 94-20, Computer Science Department, University of Minnesota, Minneapolis, Minn., 1994.

[KHA79] - L. G. KHACHIYAN, A polynomial algorithm in linear programming, Soviet Mathematics Doklady, 20 (1979), pp. 191-194.

[KLE72] - V. KLEE ET G. J. MINTY, How good is the simplex algorithm ?, in Inequalities, O. Shisha, ed., Academic Press, New York, 1972, pp. 159-175.

[KOJ91] - M. KOJIMA, S. MIZUNO ET A. YOSHISE, An $O(n^{½}L)$ iteration potential reduction algorithm for linear complementarity problems, Mathematical Programming, 50 (1991), pp. 331-342.

[LUO92] - J. LUO, Parallel processing for linear programming, PhD thesis, Delft University of Technology, 1992.

[LUS94] - I. J. LUSTIG, R. E. MARSTEN ET D. F. SHANNO, Interior point methods for linear programming : Computational state of the art, ORSA Journal on Computing, 6 (1994), pp. 1-14.

[MAR57] - H. M. MARKOWITZ, The elimination form of the inverse and its application to linear programming, Management Science, vol. 3, 1957, pp 255-269.

[MEH92] - S. MEHROTRA, On the implementation of a primal-dual interior point method, SIAM Journal on Optimization, 2 (1992), pp. 575-601.

[MIT96] - J. E. MITCHELL, Interior point methods for combinatorial optimization, in Interior point methods of mathematical programming, T. Terlaky (Ed.), Applied Optimization series, Kluwer Academic Publishers, 1996.

[NES94] - Y. E. NESTEROV ET A. S. NEMIROVSKY, Interior Point Polynomial Methods in Convex Programming, SIAM Publications, Philadelphia, 1994.

[OLK96] - J. A. OLKIN, Using Semi-Definite Programming for Controller Design in Active Noise Control, in SIAG/OPT Views-and-News, A Forum for the SIAM Activity Group on Optimization, 8, 1996.

[ROS65] - J. B. ROSEN, Pattern separation by convex programming, Journal of Mathematical Analysis and Applications, 10 (1965), pp. 123-134.

[SCH86] - A. SCHRIJVER, Theory of Linear and Integer Programming, John Wiley and Sons, New York, 1986.

[SHO70] - N. Z. SHOR, Utilization of the operation of space dilatation in the minimization of convex functions, Kibernetika, 1 (1970), pp. 6-12. Traduction anglaise : Cybernetics, 6, pp. 7-15.

[SHO87] - N. Z. SHOR, Quadratic Optimization Problems, Soviet Journal of Computer and Systems Sciences, Vol. 25 (1987), pp. 1-11.

[TAN87] - K. TANABE, Centered Newton method for mathematical programming, in System Modeling and Optimization : Proceedings of the 13th IFIP conference, Lecture Notes in Control and Information Systems 113, Berlin, August/September 1987, Springer-Verlag, New York, 1988, pp. 197-206.

[TER96] - T. TERLAKY (ED.), Interior Point Methods of Mathematical Programming, Applied Optimization series, Kluwer Academic Publishers, 1996.

[TOD90] - M. J. TODD ET Y. YE, A centered projectif algorithm for linear programming, Mathematics of Operations Research, 15 (1990), pp. 508-529.

[VAN96] - L. VANDENGERGHE ET S. BOYD, Semidefinite programming, SIAM Review 38 1, 1996, pp. 49-55.

[VAV96] - S. A. VAVASIS ET Y. YE, A primal-dual interior point method whose running time depends only on the constraint matrix, Mathematical Programming, Series A, 74 , (1996), pp. 79-120.

[WRI96] - S. J. WRIGHT, Primal-Dual Interior-Point Methods, SIAM, Philadelphia, 1996.

[YE94] - Y. YE, M. J. TODD ET S. MIZUNO, An $O(n^{1/2}L)$ homogeneous and self-dual linear programming algorithm, Mathematics of Operations Research, 19 (1994), pp. 53-67.

[ZHA95] - Y. ZHANG, On extending primal-dual interior-point algorithms from linear programming to semidefinite programming, Technical Report TR95-20, Department of Mathematics and Statistics, University of Maryland, Baltimore County, Baltimore, Md., November 1995.

K.2 BIBLIOGRAPHIE

Nous recommandons tout particulièrement les ouvrages suivants au lecteur qui désirerait approndir l'un des sujets traités au cours de ce travail de fin d'études :

■ *[FRE96]*	Résumé de la *situation actuelle* en ce qui concerne les méthodes de point intérieur.
■ *[JAN95]*	Rapide *survol théorique* des dix premières années de méthodes de point intérieur.
■ *[ARB91]*	Introduction *pratique* (nombreux exemples numériques) et *très accessible* aux méthodes de point intérieur pour la programmation linéaire.
■ *[WRI96]*	Traitement complet des *méthodes primales-duales*.
■ *[GON94, AND96]*	Etat de l'art en ce qui concerne les *implémentations* des méthodes de point intérieur pour la programmation linéaire.
■ *[VAN96]*	*Programmation semidéfinie* : théorie et applications.

L. ANNEXES

L.1 FORMULATION DU PROBLEME D'ANALYSE DISCRIMINANTE PAR ELLIPSOÏDES

On vérifie aisément, en remplaçant dans la formulation du problème d'analyse discriminante les matrices A, B, C, Γ et D par les expressions suivantes, que l'on obtient bien les contraintes désirées (on a posé la matrice I^{ab} ègale à une matrice carrée de dimension d où $I^{ab}_{ij} = \delta(i,a)\delta(j,b)$ - δ étant le symbole de Kronecker classique - , ce qui implique que tous les éléments de I^{ab} sont nuls sauf le $(a,b)^{\text{ème}}$, égal à 1)

$$A_{ij} = \begin{bmatrix} 2x_i^1x_j^1 &&&&&&&&& \\ & 2x_i^2x_j^2 &&&&&&&& \\ && \ddots &&&&&&& \\ &&& 2x_i^rx_j^r &&&&&& \\ &&&& -2y_i^1y_j^1 &&&&& \\ &&&&& -2y_i^2y_j^2 &&&& \\ &&&&&& \ddots &&& \\ &&&&&&& -2y_i^ry_j^r && \\ &&&&&&&& -(I_{ij}+I_{ji}) & \\ &&&&&&&&& (I_{ij}+I_{ji}) \end{bmatrix}, \ i \neq j$$

$$A_{ii} = \begin{bmatrix} \left(x_i^1\right)^2 &&&&&&&&& \\ & \left(x_i^2\right)^2 &&&&&&&& \\ && \ddots &&&&&&& \\ &&& \left(x_i^r\right)^2 &&&&&& \\ &&&& -\left(y_i^1\right)^2 &&&&& \\ &&&&& -\left(y_i^2\right)^2 &&&& \\ &&&&&& \ddots &&& \\ &&&&&&& -\left(y_i^r\right)^2 && \\ &&&&&&&& -I_{ii} & \\ &&&&&&&&& I_{ii} \end{bmatrix}$$

$$B_i = \begin{bmatrix} x_i^1 &&&&&&&&& \\ & x_i^2 &&&&&&&& \\ && \ddots &&&&&&& \\ &&& x_i^r &&&&&& \\ &&&& -y_i^1 &&&&& \\ &&&&& -y_i^2 &&&& \\ &&&&&& \ddots &&& \\ &&&&&&& -y_i^r && \\ &&&&&&&& O & \\ &&&&&&&&& O \end{bmatrix}$$

$$C = \begin{bmatrix} 1 &&&&&&&&& \\ & 1 &&&&&&&& \\ && \ddots &&&&&&& \\ &&& 1 &&&&&& \\ &&&& -1 &&&&& \\ &&&&& -1 &&&& \\ &&&&&& \ddots &&& \\ &&&&&&& -1 && \\ &&&&&&&& O & \\ &&&&&&&&& O \end{bmatrix}$$

$$\Gamma = \begin{bmatrix} 0 &&&&&&&&& \\ & 0 &&&&&&&& \\ && \ddots &&&&&&& \\ &&& 0 &&&&&& \\ &&&& 0 &&&&& \\ &&&&& 0 &&&& \\ &&&&&& \ddots &&& \\ &&&&&&& 0 && \\ &&&&&&&& O & \\ &&&&&&&&& -I \end{bmatrix} \quad D = \begin{bmatrix} \varepsilon &&&&&&&&& \\ & \varepsilon &&&&&&&& \\ && \ddots &&&&&&& \\ &&& \varepsilon &&&&&& \\ &&&& -\varepsilon &&&&& \\ &&&&& -\varepsilon &&&& \\ &&&&&& \ddots &&& \\ &&&&&&& -\varepsilon && \\ &&&&&&&& -I & \\ &&&&&&&&& O \end{bmatrix}$$

(ε est une constante égale à la précision des opérations arithmétiques effectuées).

L.2 CODE SOURCE DU PROGRAMME

L.2.1 addMat

```
function Res = addMat(A, B)
%addMat addMat(A,B) construit la matrice [ A   0 ]
%                                        [ 0   B ]
%
   Res = [A zeros(size(A,1),size(B,2)) ; zeros(size(B,1),size(A,2)) B];
end
```

L.2.2 elliPlot

```
function elliPlot(A,b,c)
%elliPlot elliPlot(A,c) affiche l'ellipse d'équation x' A x + b' x + c <=
0
%
   nPoints = 250;                  % Nombre de points de l'ellipse à
afficher
   centre = (-1/2) * inv(A) * b;   % Calcul du centre de l'ellipse
   indep = centre' * A * centre - c;     % Calcul du terme indépendant
   % On effectue la décomposition de Schur de la matrice A
   [Rot, Diag] = schur(A/indep);
   % Diag est une matrice portant les valeur propres sur sa diagonale
   % Les valeurs propres sont égales aux inverses des carrés des demi-
axes
   axes = 1 ./ sqrt(diag(Diag));
   % On affiche l'ellipse à l'aide de la déformation d'un cercle par Rot
   % et Diag. Les points à afficher sont stockés dans Point
   for i=1:nPoints,
      Point(:,i) = Rot * [axes(1)*cos(i*pi*2/nPoints) ; ...
                          axes(2)*sin(i*pi*2/nPoints)] + centre;
   end
   % On affiche le centre et les points
   plot(centre(1),centre(2),'yx',Point(1,:),Point(2,:),'y:');
end
```

L.2.3 elliSep

```
function [A, b, c, gamma] = elliSep(inPoints, outPoints)
%elliSep elliSep(inPoints, outPoints) retourne dans A,b,c les paramètres
%      définissant l'ellipsoïde de sphéricité maximale contenant tous
%      les points de inPoints et aucun de outPoints. Gamma contient le
%      paramètre de sphéricité de cet ellipsoide (1 si spherique)
%

%  On commence par preparer les données du programme semidéfini, à
%  savoir les matrices Ai et C, le vecteur b et blk.
   [A, b, C, blk] = prepareA(inPoints, outPoints);
% Initialisation des paramètres de l'algorithme à leurs valeurs par
défaut
   setpars;
% Initialisation de l'itéré initial
   initvars;
% Résolution du programme semidéfini
   sdp;
% Conversion du vecteur y en paramètres de l'ellipsoïde
   [A, b, c, gamma] = ytoAbc(y, size(inPoints,1));
end
```

L.2.4 pntPlot

```
function pntPlot(inPoints, outPoints)
%pntPlot pntPlot(inPoints, outPoints) affiche en vert les points inPoints
%      et en rouge les points outPoints
```

```
%
plot(inPoints(1,:)',inPoints(2,:)','g*',outPoints(1,:)',outPoints(2,:)','
r*');
end
```

L.2.5 prepareA

```
function [A,b,C,blk] = prepareA(inPoints, outPoints)
%prepareA  prepareA(inPoints, outPoints) calcule les matrices Ai et C, le
%         vecteur b et blk,  consituuant les données d'entrée de la
%         procédure de résolution de programme semidéfini sdp
%
   epsilon = 1e-8; scaling = 1;           % Paramètres du modèle
   dimPoints = size(inPoints, 1);   % On calcule la dimension des points
   if size(outPoints, 1) ~= dimPoints
      error('inPoints et outPoints doivent avoir les mêmes dimensions');
   end
   % Calcul du nombre de données
   nIn = size(inPoints, 2); nOut = size(outPoints, 2);
   % Calcul du vecteur donnant la structure de bloc de la diagonale
   blk = [ones(nIn + nOut, 1) ; dimPoints ; dimPoints];
   % Affichage des paramètres
   disp(['Dimension : ' int2str(dimPoints) ', il y a ' int2str(nIn) '
"in"
          points et ' int2str(nOut) ' "out" points.']);
   disp(['Tolerance : ' num2str(epsilon)]);
   % index va parcourir le vecteur y des inconnues, les variables duales
   index = 1;
   % Matrices Ai pour les éléments de la matrice A
   for indexI = 1:dimPoints
      for indexJ = indexI:dimPoints
         Ai = diag([inPoints(indexI,:) .* inPoints(indexJ,:)  -
                    outPoints(indexI,:) .* outPoints(indexJ,:)]);
         if indexJ ~= indexI
            Ai = 2 * Ai;
         end
         Suppl = zeros(dimPoints);
         Suppl(indexI, indexJ) = 1; Suppl(indexJ, indexI) = 1;
         Ai = addMat(Ai, -Suppl);
         Ai = addMat(Ai, Suppl);
         A(index,:) = svec(Ai, blk, 1)';
         index = index + 1;
      end
   end
   % Matrices Ai pour les éléments du vecteur b
   for indexI = 1:dimPoints
      Ai = diag([inPoints(indexI,:) -outPoints(indexI,:)]);
      Ai = addMat(Ai, zeros(2*dimPoints));
      A(index,:) = svec(Ai, blk, 1)';
      index = index + 1;
   end
   % Matrice Ai pour le coefficient c
   Ai = diag([ones(1,nIn) -ones(1,nOut)]);
   Ai = addMat(Ai, zeros(2*dimPoints));
   A(index,:) = svec(Ai, blk, 1)';
   % Matrice Ai pour le coefficient gamma
   Ai = diag([zeros(1,nIn+nOut)]);
   Ai = addMat(Ai, zeros(dimPoints));
   Ai = addMat(Ai, -eye(dimPoints));
   A(index+1,:) = svec(Ai, blk, 1)';
   % Matrice C du membre de droite
   C = diag([epsilon*ones(1,nIn) -epsilon*ones(1,nOut)]);
   C = addMat(C, -scaling*eye(dimPoints));
   C = addMat(C, zeros(dimPoints));
   % Vecteur b de l'objectif
```

```
   b = [zeros(index,1); -1];
end
```

L.2.6 testIn

```
function res = testIn(points, A, b, c)
%testIn     testIn(points, A, b, c) retourne un vecteur booleen
spécifiant
%     si les éléments de points sont à l'intérieur de l'ellipsoïde
%     défini par A, b et c
%
   for index = 1:size(points,2)
      pointCourant = [points(:,index)];
      res(index) = (pointCourant' * A * pointCourant + b' * pointCourant
+ c) <= 0;
   end
end
```

L.2.7 ytoAbc

```
function [A, b, c, gamma] = ytoAbc(y, dimPoints)
%ytoAc       ytoAbc(y, dimPoints) reconstruit la matrice A, les vecteur
b, c
%      et le coefficient gamma à partir du vecteur des variables duales y
%
   index = 1;
%  Boucle pour les éléments de A
   for indexI = 1:dimPoints
      for indexJ = indexI:dimPoints
         A(indexI, indexJ) = y(index);
         A(indexJ, indexI) = y(index);
         index = index + 1;
      end
   end
%  Boucle pour les éléments de b
   for indexI = 1:dimPoints
      b(indexI,1) = y(index);
      index = index + 1;
   end
%  Et les paramètres c et gamma
   c = y(index);
   gamma = y(index+1);
end
```

L.3