MASTER EN SCIENCES MATHÉMATIQUES

L'algorithme de plan de coupe central pour les problèmes de programmation semi-infinie convexe

TALLIER, Benoît

Award date:
1995

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de La Paix

Namur

Faculté des Sciences

L'algorithme de plan de coupe central pour les problèmes de programmation semi-infinie convexe

Mémoire présenté pour l'obtention du grade de Licencié en Sciences mathématiques par

Tallier Benoit

Promoteur : **J.-J. STRODIOT**

Année académique 1994-1995

Je remercie Monsieur le Professeur Jean-Jacques Strodiot pour son aide précieuse et son aimable collaboration tout au long de ce mémoire.

Que toutes les personnes qui ont également participé, de près ou de loin, à la réalisation de ce travail en soient très sincèrement remerciées.

Je remercie vivement mes parents qui m'ont soutenu et encouragé durant toutes mes années d'études ainsi que toute ma famille pour son soutien et son aide.

L'algorithme de plan de coupe central pour les problèmes de programmation semi-infinie convexe

TALLIER BENOIT

Résumé

Dans ce mémoire, nous considérons les problèmes de programmation semi-infinie convexe. Pour résoudre ces problèmes, nous utilisons l'algorithme de plan de coupe central dont le principe est le suivant: à chaque itération, nous réduisons l'ensemble admissible sans éliminer de solutions en effectuant des coupes. Ces coupes se font au moyen d'hyperplans construits à partir de la fonction objectif et des contraintes et passant par le centre de la plus grande sphère inscrite dans la région déterminée par toutes les coupes générées jusqu'à l'itération actuelle. La réalisation informatique de cet algorithme et quelques exemples numériques qui nous ont permis de tester le programme seront donnés.

Abstract

In this work we consider convex semi-infinite programming problems. For solving this problems, we use a central cutting plane algorithm whose principle is: at each iteration, the feasible set is reduced without eliminating solutions by using cuts. These cuts use hyperplanes built on the objective function and the constraints and passing through center of the greatest sphere inscribed within the region determined by all the cuts generated up to the current iteration. We give the computational realization of the algorithm and some numerical examples that have tested the program.

Mémoire de licence en Sciences Mathématiques.
Juin 1995.
Unité d'optimisation.
Promoteur: **J.-J. STRODIOT**

Table des matières

Introduction 5

1 Le centre sphérique 7

1.1 Définition 7

1.2 Théorème 8

1.3 Comment trouver le centre sphérique d'un polytope P ? 8

2 L'algorithme de plan de coupe central 10

2.1 Description du problème 10

2.2 Description de l'algorithme 13

2.3 Théorème de convergence 16

2.3.1 Lemmes 16

2.3.2 Le théorème de convergence 24

2.4 Problème dual et vitesse de convergence 26

2.4.1 Notations 26

2.4.2 Remarque 27

2.4.3 Lemmes et théorèmes 27

3 Implémentation de l'algorithme et exemples numériques 40

3.1 Réalisation informatique de l'algorithme et détails d'implémentation 41

3.1.1 Discrétisation des intervalles de S 41

3.1.2 Structure du fichier externe 42

3.1.3 Remarque 43

3.2 Exemples numériques 44
3.2.1 Exemples 44
3.2.2 Remarque 48

Annexes **50**
A.1 Analyse du problème 50
A.1.1 La situation initiale 50
A.1.2 La décomposition Top-Down 56
A.1.3 La situation finale 59
A.1.4 Les modules 60
A.2 Comment utiliser le programme ? 63
A.3 Listing du programme 64
A.4 Listing du fichier externe 80

Conclusion **85**

Introduction

Dans ce mémoire, nous considérons le problème de programmation semi-infinie convexe de la forme suivante:

$$(D)\begin{cases}\min f^0(\breve{x}) \\ s.c. \\ \qquad g^0(\breve{x}, t) \le 0 \qquad \textit{pour tout } t \in S \\ \qquad \breve{x} \in H\end{cases}$$

où nous supposons que H est un sous-ensemble compact, convexe et non vide de $\Re^n$, S est un sous-ensemble compact et non vide de $\Re^m$, f^0 est convexe et continuement différentiable sur H, g^0 est continue sur $H \times S$, $g^0(\cdot, t)$ est convexe pour tout t et continuement différentiable sur H, $\nabla_x g^0(x, t)$ est continue sur $H \times S$.

Pour résoudre ce problème, nous appliquons l'algorithme de plan de coupe central dont le principe est le suivant: à chaque itération, nous réduisons l'ensemble admissible sans éliminer de solutions en effectuant des coupes. Ces coupes se font au moyen d'hyperplans construits à partir de la fonction objectif et des contraintes et passant par le centre de la plus grande sphère inscrite dans la région déterminée par toutes les coupes générées jusqu'à l'itération actuelle.

Dans le premier chapitre, nous définissons la notion de centre sphérique d'un polytope et nous donnons la méthode à appliquer pour trouver ce type de centre.

Dans le deuxième chapitre, nous présentons d'abord l'algorithme de plan de coupe central et ensuite, nous établissons le lien qui existe avec les centres sphériques. D'autre part,

nous démontrons la convergence de cet algorithme et nous déterminons sa vitesse de convergence. Enfin, nous obtenons les solutions admissibles de la forme lagrangienne du problème dual de (D).

Dans le troisième chapitre, nous expliquons comment nous avons procédé pour implémenter cet algorithme sur machine. Le listing du programme, la façon dont il faut utiliser le programme et l'analyse ayant permis l'élaboration du programme sont exposés dans les annexes.

Pour terminer, nous tirons les conclusions du travail ainsi que les différents points qu'il resterait à approfondir.

Chapitre 1

Le centre sphérique

Dans ce premier chapitre, nous définissons en premier lieu la notion de centre sphérique d'un polytope P à intérieur non vide défini par m inégalités linéaires. Nous énonçons également un théorème assurant l'existence du centre. Finalement, nous montrons comment trouver le centre sphérique du polytope P.

1.1 Définition

Soit P un polytope défini par

$$\{x \in \Re^n \mid Cx \leq b\}$$

où C est une matrice de dimension $m \times n$ et b est un vecteur de $\Re^m$.
Supposons que l'intérieur de ce polytope P est non vide.

Nous appelons centre sphérique de P le centre de la plus grande hypersphère qui y est inscrite.

1.2 Théorème

Théorème 1.1

Le centre sphérique de P existe mais n'est pas nécessairement unique (Il peut même en exister une infinité: cfr FIG 1).

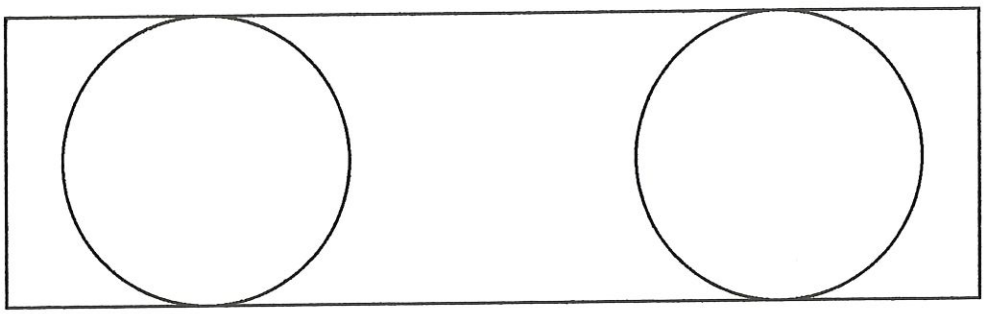


FIG 1

1.3 Comment trouver le centre sphérique de P ?

Pour déterminer le centre sphérique de P, il suffit de résoudre le problème linéaire suivant:

$$\begin{cases} \max \quad \sigma \\ s.c. \\ \qquad c_i x + \|c_i\| \sigma \le b_i \quad i = 1,\ldots,m \end{cases}$$

où c_i représente la i-ème ligne de la matrice C.

Si (x, σ) est une solution, alors x est un centre sphérique et σ le rayon de la plus grande hypersphère inscrite dans le polytope P.

Justifions la forme de ce problème linéaire.

Pour cela, il suffit de remarquer que

$$B(x, \sigma) \subseteq P$$

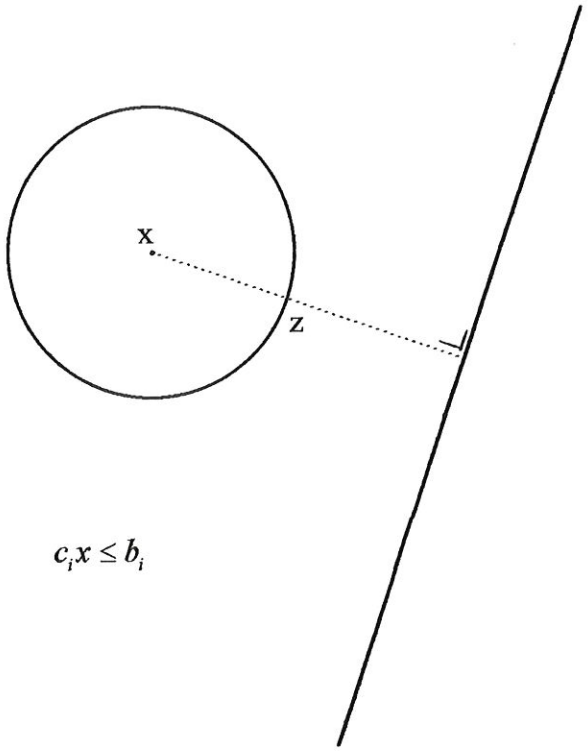
si et seulement si

tout z appartenant à $B(x, \sigma)$ vérifie

$$c_i z \leq b_i \qquad i = 1,\ldots,m$$

ou encore, pour tout i, le point $z = x + \sigma \frac{c_i^T}{\|c_i\|}$

vérifie $c_i z \leq b_i$.



Chapitre 2

L'algorithme de plan de coupe central

Dans ce deuxième chapitre, le problème que nous considérons est un problème de programmation semi-infinie convexe. Tout d'abord, nous décrivons le problème ainsi que les hypothèses considérées. Ensuite, nous présentons l'algorithme de plan de coupe central et nous démontrons sa convergence.
Nous terminons ce chapitre en exposant le dual du problème de départ, et en abordant la vitesse de convergence de l'algorithme.

2.1 Description du problème

Le problème de programmation semi-infinie convexe que nous voulons résoudre est le suivant :

$$(D)\begin{cases}\min f^0(\breve{x}) \\ s.c. \\ \qquad g^0(\breve{x}, t) \le 0 \qquad \textit{pour tout } t \in S \\ \qquad \breve{x} \in H\end{cases}$$

Pour ce problème, nous supposons que les hypothèses suivantes sont satisfaites:

Hypothèses 2.1:

(i) H est un sous-ensemble compact, convexe et non vide de $\Re^n$.

(ii) S est un sous-ensemble compact et non vide de $\Re^m$.

(iii) $f^0:\Re^n \to \Re$ est convexe et continuement différentiable sur H.

(iv) $g^0:\Re^n \times \Re^m \to \Re$ est continue sur $H \times S$, $g^0(\cdot, t)$ est convexe pour tout t et continuement différentiable sur H, et $\nabla_x g^0(x, t)$ est continue sur $H \times S$.

(v) il existe $\hat{x} \in H$ qui n'est pas optimal pour (D) et tel que $g^0(\hat{x}, t) < 0$ pour tout t dans S.

Le remplacement de la fonction objectif f^0 par une fonction linéaire n'implique aucune perte de généralité. Si la fonction objectif f^0 est convexe, alors $f^0 - z$ est évidemment convexe et le problème initial (D) est équivalent au problème suivant:

$$\begin{cases} \min \quad z \\ s.c. \\ \qquad f^0(\breve{x}) - z \le 0 \\ \qquad g^0(\breve{x}, t) \le 0 \qquad \textit{pour tout } t \in S \\ \qquad \breve{x} \in H \\ \qquad \min\limits_{\breve{x} \in H} f^0(\breve{x}) \le z \le \max\limits_{\breve{x} \in H} f^0(\breve{x}) . \end{cases}$$

Posons $x = (\breve{x}, z)$, $x_{n+1} = z$

$f: \Re^{n+1} \to \Re \qquad où \quad f(x) = f^0(\breve{x}) - x_{n+1}$

$g: \Re^{n+1} \times \Re^m \to \Re \quad où \quad g(x, t) = g^0(\breve{x}, t)$

Le problème s'écrit alors:

$$(D') \begin{cases} \min x_{n+1} \\ s.c. \\ \qquad f(x) \le 0 \\ \qquad g(x, t) \le 0 \qquad \textit{pour tout } t \in S \\ \qquad x \in H' \end{cases}$$

où $H' = H \times \left[\min_{\breve{x} \in H} f^0(\breve{x}), \max_{\breve{x} \in H} f^0(\breve{x}) \right]$ est un sous-ensemble de $\Re^{n+1}$.

Sans perte de généralité, nous pouvons supposer que

$$f^0(\hat{x}) < \max_{\breve{x} \in H} f^0(\breve{x}).$$

En effet, si ce n'était pas le cas, alors pour tout t dans S et pout tout $\breve{x}$ appartenant à H tel que $g^0(\breve{x}, t) < 0$, nous aurions que $f^0(\breve{x}) = \max_{y \in H} f^0(y)$. Mais alors par continuité, f^0 serait constante sur le domaine admissible de (D).

Par la proposition suivante, nous désirons montrer qu'un point admissible pour (D) est également admissible pour le problème (D').

Proposition 2.1.1

S'il existe un point $\breve{x}$ *appartenant à* $\Re^n$ *tel que, pour tout t dans S,* $g^0(\breve{x}, t) < 0$ *et* $\breve{x} \in H$

alors il existe $\hat{x}$ *appartenant à* $\Re^{n+1}$ *tel que, pour tout t dans S, nous avons* $\begin{cases} f(\hat{x}) < 0 \\ g(\hat{x}, t) < 0 \\ \hat{x} \in H' \end{cases}$

PREUVE

Si nous choisissons $\hat{x} = (\breve{x}, f^0(\breve{x}))$, nous avons la thèse.

□

Avant d'exposer l'algorithme de Kortanek et No [1], nous expliquons tout d'abord intuitivement la méthode:

au départ, nous partons avec un ensemble admissible H' et nous recherchons alors le point central de cet ensemble.

Si celui-ci est admissible pour le problème (D'), nous effectuons une coupe suivant la fonction objectif, c'est-à-dire que nous éliminons les points du domaine considéré de valeur supérieure à celle du point central.

Dans le cas contraire, nous effectuons une coupe passant par le point central suivant la fonction f si c'est la première contrainte de (D') qui n'est pas satisfaite, suivant la fonction g sinon, de sorte que les points éliminés soient non admissibles pour (D'). Il est à remarquer que cette réduction de l'ensemble admissible par des coupes n'entraîne aucune perte de solutions.

2.2 Description de l'algorithme

Pour pouvoir résoudre le problème de programmation semi-infinie convexe (D'), K.O. Kortanek et Hoon No proposent l'algorithme suivant:

PAS 0 :

- Choisir une constante β comprise strictement entre 0 et 1.
- Prendre $\bar{f}$ une borne supérieure sur la valeur optimale de (D')
- Considérer le problème

$$(SD_0)\begin{cases}\max\ \sigma \\ s.c. \\ \qquad x_{n+1}+\sigma \le \bar{f} \\ \qquad x \in H'\end{cases}$$

- Choisir $y^{(0)} \in H'$.
- Poser k=1.

PAS 1 :

Soit $(x^{(k)},\sigma^{(k)}) \in \Re^{n+1} \times \Re$ une solution du problème (SD_{k-1}).

Si $\sigma^{(k)} = 0$, alors STOP ($\bar{x}^{(k)}$ est la solution optimale du problème de départ).

Sinon aller au PAS 2

PAS 2 :

Supprimer éventuellement des contraintes du problème (SD_{k-1}) en appliquant les deux règles de suppression décrites ci-après.

Appeler de nouveau (SD_{k-1}) le problème résultant.

PAS 3 :

Si $x^{(k)}$ n'est pas admissible pour la première contrainte du problème (D'), c'est-à-dire si $f(x^{(k)}) > 0$, alors aller en (ii).

Sinon, si $x^{(k)}$ n'est pas admissible pour le système de contraintes infinies de (D'), c'est-à-dire si $g(x^{(k)}, t^{(k)}) > 0$ pour un certain $t^{(k)} \in S$, alors aller en (iii).

Sinon aller en (i).

(i) Ajouter la contrainte

$$x_{n+1} + \sigma \leq x_{n+1}^{(k)}$$

au problème(SD_{k-1}) et poser $y^{(k)} = x^{(k)}$.

(ii) Ajouter la contrainte

$$f(x^{(k)}) + \nabla f(x^{(k)})(x - x^{(k)}) + \left\|\nabla f(x^{(k)})\right\| \sigma \leq 0$$

au problème (SD_{k-1}) et poser $y^{(k)} = y^{(k-1)}$.

(iii) Ajouter la contrainte

$$g(x^{(k)}, t^{(k)}) + \nabla_x g(x^{(k)}, t^{(k)})(x - x^{(k)}) + \left\|\nabla_x g(x^{(k)}, t^{(k)})\right\| \sigma \leq 0$$

au problème (SD_{k-1}) et poser $y^{(k)} = y^{(k-1)}$.

Dans n'importe quel cas, appeler (SD_k) le problème résultant.

Poser k = k+1 et retourner au PAS 1.

REGLE DE SUPPRESSION 1 :

Supprimer la contrainte $x_{n+1} + \sigma \leq \bar{f}$ ou n'importe quelle contrainte générée par le PAS 3 (i) à une itération antérieure si $x^{(k)}$ est admissible pour le problème (D').

REGLE DE SUPPRESSION 2 :

Supprimer une contrainte du problème (SD_{k-1}) si

(a) elle a été générée par le PAS 3 (ii) ou (iii) à une itération j où j < k.

(b) $\sigma^{(k)} \leq \beta\sigma^{(j)}$.

(c) la contrainte n'était pas active dans (SD_{k-1}) en $(x^{(k)}, \sigma^{(k)})$.

Remarque

Nous avons déjà vu que pour trouver la plus grande boule $B(x, \sigma)$ contenue dans le polytope $P = \{x \in \Re^{n+1} \; tel que \quad c_i x \leq b_i \quad i = 1,...,m\}$, il suffit de résoudre le problème linéaire suivant:

$$(RCS)\begin{cases} \max \quad \sigma \\ s.c. \\ \qquad c_i x + \|c_i\| \, \sigma \leq b_i \qquad i = 1,...,m \end{cases}$$

Supposons maintenant que, lorsque nous effectuons l'algorithme, nous ayons le problème linéaire suivant pour tout t appartenant à S:

$$(SD_k)\begin{cases} \max \quad \sigma \\ s.c. \\ \qquad f(x^{(k)}) + \nabla f(x^{(k)})(x - x^{(k)}) + \|\nabla f(x^{(k)})\| \, \sigma \leq 0 \\ \qquad g(x^{(k)}, t) + \nabla_x g(x^{(k)}, t)(x - x^{(k)}) + \|\nabla_x g(x^{(k)}, t)\| \, \sigma \leq 0 \\ \qquad x_{n+1} + \sigma \leq \bar{f} \end{cases}$$

Si nous faisons l'identification $c_1 = \nabla f(x^{(k)})$, $c_2 = \nabla_x\ g(x^{(k)}, t)$ *et* $c_3 = (0,\ldots,0,1)$, nous en déduisons que (SD_k) est équivalent au problème (RCS).

En conclusion, rechercher la solution de chaque problème linéaire (SD_k) revient en fait à rechercher le centre sphérique du polytope Q où

$$Q \equiv \left\{ \begin{array}{l} x \in \Re^{n+1} \textit{ tel que } f(x^{(k)}) + \nabla f(x^{(k)})(x - x^{(k)}) \leq 0 \\ \qquad\qquad g(x^{(k)}, t) + \nabla_x\, g(x^{(k)}, t)(x - x^{(k)}) \leq 0 \\ \qquad\qquad x_{n+1} \leq \bar{f} \end{array} \right\}.$$

Cette méthode génère aussi bien des points admissibles que des points non-admissibles pour le problème (D'). La convergence de l'algorithme sera abordée dans le théorème 2.3.1 du paragraphe suivant . Ce théorème sera précédé de trois lemmes qui en faciliteront la démonstration.

2.3 Théorème de convergence

Avant de démontrer le théorème 2.3.1 qui prouve la convergence de l'algorithme, nous allons d'abord faire la preuve de trois lemmes qui nous aideront à montrer la thèse du théorème.

2.3.1 Lemmes

Lemme 2.3.1

En utilisant l'une ou l'autre, les deux, ou aucune des règles de suppression, si l'algorithme ne se termine pas, la suite $\{\sigma^{(k)}\}_k$ *converge vers 0.*

PREUVE

Comme H' est compact et comme les fonctions f et g sont continues, il existe un point qui est optimal pour (D').

Soit $\bar{x} \in \Re^{n+1}$ ce point. Alors $(\bar{x},0)$ est réalisable pour chaque problème (SD_k) et donc

$\sigma^{(k)} \geq 0$ car $\bar{x}_{n+1} < \bar{f}$.

Les règles de suppression 1 et 2 suppriment seulement les contraintes redondantes.

Dès lors, pour tout k, on a

$$\sigma^{(k)} \geq \sigma^{(k+1)} . \tag{3.1}$$

La suite $\sigma^{(k)}$ étant décroissante et positive, nous en déduisons que

$$\lim_k \sigma^{(k)} = \bar{\sigma} \geq 0 \quad \text{avec } \bar{\sigma} \text{ fini.}$$

Supposons par l'ABSURDE que $\bar{\sigma}$ est strictement positif.

Dans ce cas, la condition (b) de la règle de suppression 2 n'est jamais satisfaite pour k assez grand.

En effet, nous avons qu'il existe $\hat{k}$ tel que

$$\bar{\sigma} \leq \sigma^{(\hat{k})} < \frac{\bar{\sigma}}{\beta} \quad \text{où } 0 < \beta < 1. \tag{3.2}$$

Dès lors de (3.1) et (3.2), pour $k \geq j \geq \hat{k}$, nous en déduisons que

$$\bar{\sigma} \leq \sigma^{(k)} \leq \sigma^{(j)} < \frac{\bar{\sigma}}{\beta} .$$

et donc, pour tout $k \geq j \geq \hat{k}$, que

$$\beta\sigma^{(j)} \leq \sigma^{(k)} .$$

Par conséquent, la condition (b) de la règle de suppression 2 n'est jamais satisfaite pour $j \geq \hat{k}$.

Les contraintes engendrées aux PAS 3 (ii) ou (iii) ne sont jamais supprimées.

En vue d'obtenir une contradiction, nous montrons que, pour tout $k > j$, nous avons

$$\left\| x^{(k)} - x^{(j)} \right\| \geq \sigma^{(k)} \geq \bar{\sigma}$$

est vrai dans chacun des cas qui suivront.

CAS 1 : $x^{(j)}$ est non admissible pour la première contrainte de (D').

Dans ce cas, le PAS 3 (ii) est utilisé pour générer une coupe, et donc, pour tout $k > j$, nous avons

$$f(x^{(j)}) + \nabla f(x^{(j)})(x^{(k)} - x^{(j)}) + \left\|\nabla f(x^{(j)})\right\|\sigma^{(k)} \leq 0. \tag{3.3}$$

De (3.3), nous déduisons

$$\begin{aligned} \left\|\nabla f(x^{(j)})\right\|\sigma^{(k)} &\leq -f(x^{(j)}) - \nabla f(x^{(j)})(x^{(k)} - x^{(j)}) \\ &\leq -\nabla f(x^{(j)})(x^{(k)} - x^{(j)}) \\ &\leq \left\|\nabla f(x^{(j)})\right\| \left\|x^{(k)} - x^{(j)}\right\|. \end{aligned} \tag{3.4}$$

où la deuxième inégalité vient du fait que $f(x^{(j)}) > 0$ et la troisième inégalité découle de l'inégalité de Cauchy-Schwarz.
Comme $\nabla f(x^{(j)}) \neq 0$, pour $k > j$, nous concluons à partir de (3.4)

$$\left\|x^{(k)} - x^{(j)}\right\| \geq \sigma^{(k)} \geq \overline{\sigma}.$$

car $\sigma^{(k)}$ est décroissante et $\sigma^{(k)} \xrightarrow[k\to\infty]{} \overline{\sigma}$.

CAS 2 : $x^{(j)}$ est non admissible pour les contraintes de continuité de (D').

Dans ce cas, le PAS 3 (iii) est utilisé pour générer une coupe, et donc, pour tout $k > j \geq \hat{k}$, nous avons

$$g(x^{(j)}, t^{(j)}) + \nabla_x g(x^{(j)}, t^{(j)})(x^{(k)} - x^{(j)}) + \left\|\nabla_x g(x^{(j)}, t^{(j)})\right\| \sigma^{(k)} \leq 0. \tag{3.5}$$

Comme $(x^{(j)}, t^{(j)})$ est non admissible, nous déduisons que

$$g(x^{(j)}, t^{(j)}) > 0 \quad \text{et} \quad \nabla_x g(x^{(j)}, t^{(j)}) \neq 0. \tag{3.6}$$

En utilisant (3.6) dans (3.5) et en faisant passer $\nabla_x g(x^{(j)}, t^{(j)})(x^{(k)} - x^{(j)})$ dans le second membre, nous obtenons

$$\left\|\nabla_x g(x^{(j)}, t^{(j)})\right\| \sigma^{(k)} \leq -\nabla_x g(x^{(j)}, t^{(j)})(x^{(k)} - x^{(j)})$$

$$\leq \left\|\nabla_x g(x^{(j)}, t^{(j)})\right\| \left\|x^{(k)} - x^{(j)}\right\|$$

où la seconde inégalité découle de l'inégalité de Cauchy-Schwarz.

Par conséquent, pour tout $k > j$, nous avons

$$\left\|x^{(k)} - x^{(j)}\right\| \geq \sigma^{(k)} \geq \overline{\sigma}.$$

CAS 3 : $x^{(j)}$ est admissible pour (D').

Dans ce cas, pour tout $k > j$, nous avons que

$$x_{n+1}^{(k)} + \sigma^{(k)} \leq x_{n+1}^{(j)} \quad .$$

D'où, pour tout $k > j$, nous obtenons que

$$\left\|x_{n+1}^{(k)} - x_{n+1}^{(j)}\right\| \geq \sigma^{(k)} \geq \overline{\sigma}.$$

Par conséquent, dans chaque cas, pour tout $k > j \geq \hat{k}$, nous concluons que

$$\left\|x^{(k)} - x^{(j)}\right\| \geq \sigma^{(k)} \geq \overline{\sigma} > 0 .$$

Ceci CONTREDIT le fait que $\{x^{(k)}\}_k$ se trouvant dans le compact H', doit avoir au moins une valeur d'adhérence.

Donc la suite $\{\sigma^{(k)}\}_k$ converge vers 0.

□

Lemme 2.3.2

Soit $\tilde{x}$ admissible pour (D').
Si $f(\tilde{x}) < 0$ et si $g(\tilde{x}, t) < 0$ pour tout t appartenant à S, alors il existe $\tilde{\sigma} > 0$ tel que $(\tilde{x}, \tilde{\sigma})$ satisfait à n'importe quelle contrainte générée par le PAS 3 (ii) ou le PAS 3 (iii).

PREUVE

Examinons tout d'abord les itérations pour lesquelles une contrainte est générée par le PAS 3 (ii). Notons par K l'ensemble des indices correspondant à ces itérations. Rappelons que pour celles-ci, nous avons que la première contrainte du problème (D') n'est pas vérifiée, c'est-à-dire que, pour tout $k \in K$, $f(x^{(k)}) > 0$.

Montrons maintenant qu'il existe $\tilde{\sigma}_1 > 0$ tel que, pour tout $k \in K$,

$$f(x^{(k)}) + \nabla f(x^{(k)})(\tilde{x} - x^{(k)}) + \left\|\nabla f(x^{(k)})\right\| \tilde{\sigma}_1 \leq 0. \tag{3.7}$$

Afin d'arriver à cette conclusion, nous remarquons que

$$\begin{aligned} & f(x^{(k)}) + \nabla f(x^{(k)})(\tilde{x} - x^{(k)}) + \left\|\nabla f(x^{(k)})\right\| \tilde{\sigma}_1 \\ & \leq f(x^{(k)}) + (f(\tilde{x}) - f(x^{(k)})) + \left\|\nabla f(x^{(k)})\right\| \tilde{\sigma}_1 \\ & = f(\tilde{x}) + \left\|\nabla f(x^{(k)})\right\| \tilde{\sigma}_1 \end{aligned}$$

où la première inégalité découle de la convexité de la fonction f.

Par conséquent, l'inégalité (3.7) est satisfaite pour $\tilde{\sigma}_1$ vérifiant

$$0 < \tilde{\sigma}_1 \leq -\frac{f(\tilde{x})}{M_1}$$

où $M_1 = \max\left\{\left\|\nabla f(x)\right\| \mid x \in H \setminus \{x \mid f(x) \leq 0\}\right\}$ est un nombre réel strictement positif.

Regardons à présent les itérations pour lesquelles une contrainte est générée par le PAS 3 (iii). Notons par L l'ensemble des indices correspondant à ces itérations. Rappelons également que pour celles-ci, la deuxième contrainte de (D') n'est pas vérifiée, c'est-à-dire que, pour tout indice k appartenant à L, il existe une valeur $t^{(k)}$ dans S tel que $g(x^{(k)},t^{(k)}) > 0$.

Par un raisonnement analogue au cas précédent, nous allons démontrer que, pour tout indice k dans L, il existe $\tilde{\sigma}_2 > 0$ tel que

$$g(x^{(k)}, t^{(k)}) + \nabla_x g(x^{(k)}, t^{(k)})(\tilde{x} - x^{(k)}) + \left\|\nabla_x g(x^{(k)}, t^{(k)})\right\| \tilde{\sigma}_2 \le 0. \tag{3.8}$$

Par la convexité de $g(\cdot, t^{(k)})$, nous avons

$$g(x^{(k)}, t^{(k)}) + \nabla_x g(x^{(k)}, t^{(k)})(\tilde{x} - x^{(k)}) + \left\|\nabla_x g(x^{(k)}, t^{(k)})\right\| \tilde{\sigma}_2$$

$$\le g(\tilde{x}, t^{(k)}) + \left\|\nabla_x g(x^{(k)}, t^{(k)})\right\| \tilde{\sigma}_2$$

Comme précédemment, nous posons

$$M_2 = \max_{t \in S}\left\{\max\left\|\nabla_x g(x, t)\right\| \mid x \in H \setminus \left\{x \mid g(x, t^{(k)}) \le 0\right\}\right\}$$

et finalement, nous obtenons que (3.8) est satisfaite pour $\tilde{\sigma}_2$ vérifiant

$$0 < \tilde{\sigma}_2 \le -\frac{g(\tilde{x}, t^{(k)})}{M_2}.$$

Si on pose $\tilde{\sigma} = \min\left\{\tilde{\sigma}_1, \tilde{\sigma}_2\right\}$ qui est strictement positif, alors on a la thèse.

□

Lemme 2.3.3

Si l'algorithme se termine à l'itération k^*, $y^{(k^*-1)}$ *est admissible pour (D').*
Si l'algorithme ne se termine pas, alors il existe $\hat{k}$ *tel que* $y^{(k)}$ *est admissible pour (D'),*
pour $k \geq \hat{k}$.

Interprétation du lemme 2.3.3: Après un certain nombre d'itérations, nous n'ajouterons plus de coupe suivant les fonctions f et g; seules des contraintes du type $x_{n+1} + \sigma \leq x_{n+1}^{(k)}$ seront ajoutées au problème.

PREUVE du Lemme 2.3.3

Supposons par l'ABSURDE que pour tout k, $y^{(k)}$ est non admissible pour (D'). Dès lors, pour chaque k, $x^{(k)}$ est non admissible pour (D') puisque, chaque fois que nous effectuons une itération de l'algorithme, nous affectons à $y^{(k)}$ uniquement les valeurs réalisables de $x^{(k)}$. Par conséquent, le PAS 3 (i) n'est jamais utilisé pour générer une contrainte, et la règle de suppression 1 n'est jamais employée pour supprimer une contrainte car il faut que $x^{(k)}$ soit admissible.

Afin d'obtenir une contradiction, nous allons démontrer qu'il existe x^* appartenant à $\Re^{n+1}$ et $\sigma^* > 0$ tels que (x^*, σ^*) est admissible pour chaque problème (SD_k).

Tout d'abord, prouvons qu'il est possible de trouver x^* appartenant à $\Re^{n+1}$ et $\sigma' > 0$ tels que nous ayons l'inégalité suivante:

$$x_{n+1}^* + \sigma' \leq \bar{f}.$$

Soit $\bar{x}$ optimal pour (D') et $\hat{x}$ le point de l'HYPOTHESE 2.1 (v). Alors $x(\alpha) = \alpha\hat{x} + (1-\alpha)\bar{x}$ est admissible pour $0 \leq \alpha \leq 1$ et, pour tout t appartenant à S,

$$(3.9)\ \begin{cases} f(x(\alpha)) < 0 \\ g(x(\alpha), t) < 0 \end{cases} \qquad 0 < \alpha \leq 1$$

En effet

- $f(x(\alpha)) = f(\alpha\hat{x} + (1-\alpha)\bar{x})$

  $\leq \alpha f(\hat{x}) + (1-\alpha) f(\bar{x})$

  $\leq \alpha f(\hat{x}) < 0$

  puisque f est convexe par HYPOTHESE 2.1 (iii), $f(\bar{x}) \leq 0$ et par la proposition 2.1.1 .

- $g(x(\alpha), t) = g(\alpha\hat{x} + (1-\alpha)\bar{x}, t)$

  $\leq \alpha g(\hat{x}, t) + (1-\alpha) g(\bar{x}, t)$

  $\leq \alpha g(\hat{x}, t) < 0$

  puisque g est convexe par HYPOTHESE 2.1 (iv), $g(\bar{x}, t) \leq 0$ et par la proposition 2.1.1 .

Comme nous avons que $\bar{x}_{n+1} < \bar{f}$, on peut choisir $\tilde{\alpha} \in (0,1)$ suffisamment petit tel que

$$x_{n+1}(\tilde{\alpha}) < \bar{f}.$$

Nous en déduisons alors qu'il existe $\sigma' > 0$ tel que

$$x_{n+1}(\tilde{\alpha}) + \sigma' \leq \bar{f}.$$

D'autre part, par le Lemme 2.3.2 et (3.9), nous pouvons trouver σ'' tel que $(x(\tilde{\alpha})), \sigma'')$ satisfait à n'importe quelle contrainte générée par le PAS 3 (ii) ou le PAS 3 (iii).

Finalement si nous choisissons $\sigma^* = \min(\sigma', \sigma'') > 0$, $(x(\tilde{\alpha}), \sigma^*)$ est admissible pour chaque problème (SD_k).

Donc $\lim \sigma^{(k)} \geq \sigma^* > 0$. Ceci CONTREDIT le Lemme 2.3.1 .

□

2.3.2 Théorème de convergence

Théorème 2.3.1

Si l'algorithme se termine à l'itération k^*, alors $y^{(k^*-1)}$ est optimal pour (D').
Sinon, il existe au moins une valeur d'adhérence pour la suite $\{y^{(k)}\}_k$ et celles-ci sont optimales pour le problème (D').

PREUVE

Pour démontrer ce théorème, nous devons envisager le cas où l'algorithme ne se termine pas et le cas où celui-ci se termine à l'itération k^*.

a) Supposons tout d'abord que l'algorithme ne se termine pas en un nombre fini d'itérations.
En vertu des Lemmes 2.3.1 et 2.3.3, nous pouvons affirmer qu'il existe un indice $\hat{k}$ tel que pour tout $k \geq \hat{k}$, $y^{(k)}$ est admissible pour le problème (D').

Comme la région admissible pour (D') est compacte, la suite $\{y^{(k)}\}_k$ a au moins une valeur d'adhérence admissible. Notons par $\bar{y}$ une valeur d'adhérence et $\{y^{(k)}\}_k$ une sous-suite convergente vers $\bar{y}$.

Supposons par l'ABSURDE que $\bar{y}$ n'est pas optimale.
Afin d'arriver à une contradiction , nous allons prouver que l'on peut trouver x^* appartenant à $\Re^{n+1}$ et $\sigma^* > 0$ tels que (x^*, σ^*) est admissible pour chaque problème (SD_k).

Soit $\bar{x}$ une valeur optimale pour le problème (D') et $\hat{x}$ un point qui vérifie l'HYPOTHESE 2.1 (v).

Alors, nous pouvons dire, en définissant $x(\alpha) = \alpha\bar{x} + (1-\alpha)\hat{x}$, que pour tout t dans S

$$\begin{cases} f(x(\alpha)) < 0 \\ g(x(\alpha), t) < 0 \end{cases} \qquad 0 \le \alpha < 1$$

et $x_{n+1}(\alpha) < \bar{y}_{n+1}$ pour $\max\left(\frac{\bar{y}_{n+1} - \hat{x}_{n+1}}{\bar{x}_{n+1} - \hat{x}_{n+1}}, 0\right) \le \alpha < 1$.

Si nous choisissons $\tilde{\alpha} \in \left(\max\left(\frac{\bar{y}_{n+1} - \hat{x}_{n+1}}{\bar{x}_{n+1} - \hat{x}_{n+1}}, 0\right), 1\right)$ et si nous posons $\sigma' = \bar{y}_{n+1} - x_{n+1}(\tilde{\alpha}) > 0$ alors, pour tout $k \ge \hat{k}$, le point $(x_{n+1}(\tilde{\alpha}), \sigma')$ satisfait à l'inégalité suivante

$$x_{n+1} + \sigma' \le \bar{y}_{n+1} < y_{n+1}^{(k)} .$$

D'après le Lemme 2.3.2, il existe $\sigma'' > 0$ *tel que* $(x(\tilde{\alpha}), \sigma'')$ satisfait à n'importe quelle coupe générée par le PAS 3 (ii) ou le PAS 3 (iii).
Par conséquent, si nous prenons $\sigma^* = \min(\sigma', \sigma'') > 0$, nous obtenons un point $(x(\tilde{\alpha}), \sigma^*)$ réalisable pour chaque problème (SD_k).
Finalement, nous avons que $\lim_k \sigma^{(k)} \ge \sigma^* > 0$ qui CONTREDIT le Lemme 2.3.1 , ce qui démontre le point a).

b) Supposons que l'algorithme se termine à l'itération k^*.
Par le Lemme 2.3.3, nous en déduisons que $y^{(k^*-1)}$ est réalisable pour (D'). Si, par l'ABSURDE, $y^{(k^*-1)}$ n'est pas optimal, nous pouvons prouver de la même manière qu'au point a) que $\sigma^{(k^*)} > 0$. Dès lors, l'algorithme ne pouvait pas être terminé.

Finalement, dans chaque cas, nous avons obtenu des solutions optimales pour le problème (D').

□

2.4 Problème dual et vitesse de convergence

Dans ce paragraphe, nous fixons en premier lieu les notations que nous utilisons. Ensuite, nous donnons trois lemmes qui nous permettront de déterminer des solutions admissibles pour la forme lagrangienne du problème dual de (D) ainsi que la vitesse de convergence de l'algorithme.

2.4.1 Notations

Afin de déterminer le problème dual et la vitesse de convergence de l'algorithme, nous supposerons que H' est un polyèdre défini par un ensemble d'inégalités linéaires, c'est-à-dire que H' soit défini par $\left\{x \in \Re^{n+1} \; tel \; que \quad a_j x \le b_j \;,\; j \in J\right\}$, où $J = \{1,...,m\}$.
Nous faisons également l'hypothèse que la matrice A, où a_j est sa j-ème ligne, est de rang plein.

Nous notons dorénavant par $\bar{x}$ appartenant à $\Re^{n+1}$ un point d'adhérence d'une suite infinie générée par l'algorithme.
Nous nommerons par $\Re^{(S)}$ l'espace linéaire de toutes les fonctions à valeurs réelles, sur S ayant un support fini.
Nous considérons dans ce paragraphe la forme lagrangienne suivante du problème (D):

$$(PD)\begin{cases} \sup\limits_{\xi \in \Re^{(S)},\, \Psi \in \Re^{n}} \left\{ \inf\limits_{x \in \Re^{n}} L(\xi, \Psi, x) \right\} \\ s.c. \\ \quad \inf\limits_{x} L(\xi, \Psi, x) > -\infty \\ \quad \xi \ge 0 \\ \quad \Psi \ge 0 \end{cases}$$

où l'expression de la fonction lagrangienne est

$$L(\xi, \Psi, x) = f^0(x) + \sum_t \xi(t) g^0(x,t) + \sum_j \Psi_j (a_j x - b_j).$$

Enfin, dans le but d'utiliser ses propriétés duales, le problème linéaire (SD_{k-1}) considéré dans la suite sera le suivant:

$$(SD_{k-1})\begin{cases} \max \quad \sigma & \\ s.c. & \\ \quad x_{n+1} + \sigma \le y_{n+1}^{(k)} & \\ \quad f(x^{(i)}) + \nabla f(x^{(i)})(x - x^{(i)}) + \left\| \nabla f(x^{(i)}) \right\| \sigma \le 0 & \text{pour } i \in F_k \\ \quad g(x^{(l)}, t^{(l)}) + \nabla_x g(x^{(l)}, t^{(l)})(x - x^{(l)}) + \left\| \nabla_x g(x^{(l)}, t^{(l)}) \right\| \sigma \le 0 & \text{pour } l \in G_k \\ \quad a_j x \le b_j & \text{pour } j \in J \end{cases}$$

où F_k et G_k sont les ensembles des indices des contraintes qui sont générées respectivement à partir de f(x) et g(x,t).

2.4.2 Remarque

Si nous avons une solution optimale $(x^{(k)}, \sigma^{(k)})$ du problème linéaire (SD_{k-1}) avec $\sigma^{(k)}$ non nul, alors les multiplicateurs de Lagrange associés à la contrainte $\min_{\breve{x} \in H} f(\breve{x}) \le x_{n+1} \le \max_{\breve{x} \in H} f(\breve{x})$ de l'ensemble polyédrique H' sont tous nuls.
Cette affirmation provient du fait que la contrainte n'est pas active puisque le centre sphérique ne peut pas se situer sur la frontière de l'ensemble polyédrique (car sinon $\sigma^{(k)}$ est égal à 0).

2.4.3 Lemmes et théorèmes

Avant de préciser la vitesse de convergence sur l'ensemble des points admissibles donnés par l'algorithme de plan de coupe central et les solutions admissibles pour la forme lagrangienne du problème dual de (D), nous devons démontrer trois lemmes.

Lemme 2.4.1

Considérons pour (SD_{k-1}) *la solution optimale* $(x^{(k)}, \sigma^{(k)})$ *et les variables duales optimales* $\mu_0^{(k)}, \{\upsilon_i^{(k)} \quad \forall i \in F_k\}, \{\mu_l^{(k)} \quad \forall l \in G_k\}$ *et* $\{\lambda_j^{(k)} \quad \forall j \in J\}$.

Soit $\bar{x}$ *appartenant à* $\Re^n$ *admissible pour(D) et supposons que* $\mu_0^{(k)}$ *soit strictement positif. Alors*

$$f^0(\bar{x}) \geq y_{n+1}^{(k-1)} - \left(\frac{\sigma^{(k)}}{\mu_0^{(k)}}\right).$$

Dans ce cas, le membre de droite est une borne inférieure de la valeur de la fonction objectif du problème (D).

PREUVE

Pour démontrer cette inégalité, nous utiliserons les conditions de Kuhn-Tucker sous forme lagrangienne et la convexité des fonctions f et g.

L'expression de la fonction lagrangienne est donnée par:

$$\begin{aligned} L(x,\sigma;\mu_0^{(k)},\upsilon_i^{(k)},\mu_l^{(k)},\lambda_j^{(k)}) &= -\sigma + (x_{n+1} + \sigma - y_{n+1}^{(k-1)})\mu_0^{(k)} \\ &+ \sum_{i\in F_k}\left[f(x^{(i)}) + \nabla f(x^{(i)})(x - x^{(i)}) + \left\|\nabla f(x^{(i)})\right\| \sigma\right] \upsilon_i^{(k)} \\ &+ \sum_{l\in G_k}\left[g(x^{(l)}, t^{(l)}) + \nabla_x g(x^{(l)}, t^{(l)})(x - x^{(l)}) + \left\|\nabla_x g(x^{(l)}, t^{(l)})\right\| \sigma\right] \mu_l^{(k)} \\ &+ \sum_{j\in J}(a_j x - b_j)\lambda_j^{(k)} . \end{aligned}$$

Par la remarque précédente et par les conditions de Kuhn-Tucker sous forme lagrangienne, nous avons

(i) en dérivant par rapport à x_{n+1} que

$$\mu_0^{(k)} + \sum_{i\in F_k}(\nabla_{x_{n+1}} f(x^{(i)}))\upsilon_i^{(k)} + \sum_{l\in G_k}(\nabla_{x_{n+1}} g(x^{(l)}, t^{(l)}))\mu_l^{(k)} = 0$$

ou encore $\mu_0^{(k)} = \sum_{i \in F_k} \upsilon_i^{(k)}$ (4.1)

car, par définition de f et g, $\nabla_{x_{n+1}} f(x) = -1$ et $\nabla_{x_{n+1}} g(x,t) = 0$.

(ii) en dérivant par rapport à σ que

$$-1 + \mu_0^{(k)} + \sum_{i \in F_k} \left\| \nabla f(x^{(i)}) \right\| \upsilon_i^{(k)} + \sum_{l \in G_k} \left\| \nabla_x g(x^{(l)}, t^{(l)}) \right\| \mu_l^{(k)} = 0$$

c'est-à-dire

$$\mu_0^{(k)} + \sum_{i \in F_k} \left\| \nabla f(x^{(i)}) \right\| \upsilon_i^{(k)} + \sum_{l \in G_k} \left\| \nabla_x g(x^{(l)}, t^{(l)}) \right\| \mu_l^{(k)} = 1 \qquad (4.2)$$

(iii) puisque nous sommes dans le cas convexe que

$$L(x^{(k)}, \sigma^{(k)}; \mu_0^{(k)}, \upsilon_i^{(k)}, \mu_l^{(k)}, \lambda_j^{(k)}) \leq L(x, \sigma; \mu_0^{(k)}, \upsilon_i^{(k)}, \mu_l^{(k)}, \lambda_j^{(k)}).$$

Cette inégalité est équivalente à

$$\begin{aligned} \sigma^{(k)} \geq \sigma &- \mu_0^{(k)}(x_{n+1} + \sigma - y_{n+1}^{(k)}) \\ &- \sum_{i \in F_k} \left[f(x^{(i)}) + \nabla f(x^{(i)})(x - x^{(i)}) + \left\| \nabla f(x^{(i)}) \right\| \sigma \right] \upsilon_i^{(k)} \\ &- \sum_{l \in G_k} \left[g(x^{(l)}, t^{(l)}) + \nabla_x g(x^{(l)}, t^{(l)})(x - x^{(l)}) + \left\| \nabla_x g(x^{(l)}, t^{(l)}) \right\| \sigma \right] \mu_l^{(k)} \\ &- \sum_{j \in J} (a_j x - b_j) \lambda_j^{(k)} \end{aligned} \qquad (4.3)$$

pour tout x appartenant à $\Re^{n+1}$ et pour tout σ appartenant à $\Re$.

Dans la suite, nous essayerons de convertir (4.3) en une inégalité de type lagrangien en utilisant la convexité de f et g et en simplifiant avec (4.1) et (4.2).

En effet, par la convexité de f et g, nous obtenons que (4.3) est équivalent à

$$\begin{aligned}\sigma^{(k)} \geq \sigma &- \mu_0^{(k)}(x_{n+1} + \sigma - y_{n+1}^{(k-1)}) \\ &- \sum_{i \in F_k} \upsilon_i^{(k)}(f(x) + \left\|\nabla f(x^{(i)})\right\| \sigma) \\ &- \sum_{l \in G_k} \mu_l^{(k)}(g(x, t^{(l)}) + \left\|\nabla_x\, g(x^{(l)}, t^{(l)})\right\| \sigma) \\ &- \sum_{j \in J}(a_j x - b_j)\lambda_j^{(k)}\end{aligned}$$

Cette expression peut également s'écrire de la manière suivante:

$$\begin{aligned}\sigma^{(k)} \geq \sigma &- \mu_0^{(k)}(x_{n+1} - y_{n+1}^{(k-1)}) \\ &- \left[\mu_0^{(k)} + \sum_{i \in F_k} \upsilon_i^{(k)} \left\|\nabla f(x^{(i)})\right\| + \sum_{l \in G_k} \mu_l^{(k)} \left\|\nabla_x\, g(x^{(l)}, t^{(l)})\right\|\right] \sigma \\ &- \sum_{i \in F_k} \upsilon_i^{(k)} f(x) - \sum_{l \in G_k} \mu_l^{(k)} g(x, t^{(l)}) - \sum_{j \in J}(a_j x - b_j)\lambda_j^{(k)}\end{aligned}$$

Par la définition de f et en vertu des égalités (4.1) et (4.2), nous avons

$$\sigma^{(k)} \geq \mu_0^{(k)}(y_{n+1}^{(k-1)} - f^0(x)) - \sum_{l \in G_k} \mu_l^{(k)} g(x, t^{(l)}) - \sum_{j \in J} \lambda_j^{(k)}(a_j x - b_j).$$

Cette dernière inégalité est équivalente à

$$\sigma^{(k)} \geq \mu_0^{(k)}(y_{n+1}^{(k-1)} - f^0(\hat{x})) - \sum_{l \in G_k} \mu_l^{(k)} g(\hat{x}, t^{(l)}) - \sum_{j \in J} \lambda_j^{(k)}(a_j \hat{x} - b_j) \qquad (4.4)$$

pour tout $\hat{x}$ appartenant à $\Re^n$ car f^0 est défini sur $\Re^n$, par définition de g et par la remarque précédente.

Etant donné que par hypothèse $\mu_0^{(k)}$ est strictement positif, et comme pour tout point réalisable $\hat{x}$ appartenant à $\Re^n$ $g(\hat{x}, t^{(l)}) \leq 0$ *et* $(a_j \hat{x} - b_j) \leq 0$ pour tout l et pour tout j, nous obtenons finalement

$$\sigma^{(k)} \geq \mu_0^{(k)}(y_{n+1}^{(k-1)} - f^0(\hat{x}))$$

et donc

$$f^0(\hat{x}) \geq y_{n+1}^{(k-1)} - \left(\frac{\sigma^{(k)}}{\mu_0^{(k)}}\right)$$

□

Lemme 2.4.2

Toutes suites de vecteurs duaux $\left\{\mu_0^{(k)}\right\}_k, \left\{\upsilon_i^{(k)} \quad \forall i \in F_k\right\}_k, \left\{\mu_l^{(k)} \quad \forall l \in G_k\right\}_k$ *et* $\left\{\lambda_j^{(k)} \quad \forall j \in J\right\}_k$, *optimaux pour le problème linéaire* (SD_{k-1}), *sont bornées.*

De plus, $\left\{\sum_{l \in G_k} \mu_l^{(k)}\right\}_k$ *est également bornée.*

PREUVE

A partir de l'égalité (4.1), nous déduisons immédiatement que $\mu_0^{(k)} \in [0, 1]$ car (4.1) constitue une somme de nombres positifs valant 1, et donc $\left\{\mu_0^{(k)}\right\}_k$ est bornée.

Grâce à ce résultat, en vertu de (4.2) et comme $\sum_{i \in F_k} \upsilon_i^{(k)}$ est une somme de nombres positifs, nous concluons que $\left\{\upsilon_i^{(k)} \quad \forall i \in F_k\right\}_k$ est bornée.

Démontrons maintenant que $\left\{\mu_l^{(k)} \quad \forall l \in G_k\right\}_k$ est bornée. En effet, pour tout l, $\mu_l^{(k)} \in [0, 1]$ par (4.2) car nous avons (i) une somme de nombres positifs égal à 1.

(ii) $\left\|\nabla_x g(x^{(l)}, t^{(l)})\right\|$ est bornée puisque $(x^{(l)}, t^{(l)})$ appartient à un compact et $\|\cdot\| \circ \nabla_x \circ g$ est continue.

(iii) $\mu_0^{(k)} + \sum_{i \in F_k} \upsilon_i^{(k)} \left\|\nabla f(x^{(i)})\right\|$ est bornée.

Nous montrons à présent que $\left\{\sum_{l\in G_k}\mu_l^{(k)}\right\}_k$ est bornée c'est-à-dire que pour tout ε strictement positif il existe un indice k_0 tel que, pour tout $k \geq k_0$, nous avons

$$0 < \sum_{l\in G_k}\mu_l^{(k)} \leq \varepsilon.$$

Considérons $\hat{x}$ appartenant à $\Re^n$ un point vérifiant l'HYPOTHESE 2.1 (v). Nous pouvons écrire, pour tout k, (4.4) en remplaçant $\bar{x}$ par le point particulier $\hat{x}$, ce qui implique que

$$\sigma^{(k)} \geq \mu_0^{(k)}(y_{n+1}^{(k-1)} - f^0(\hat{x})) - \sum_{l\in G_k}\mu_l^{(k)} g(\hat{x}, t^{(l)}) - \sum_{j\in J}\lambda_j^{(k)}(a_j\hat{x} - b_j).$$

Etant donné que $-\sum_{j\in J}\lambda_j^{(k)}(a_j\hat{x} - b_j) \geq 0$ puisque $\hat{x}$ est un point vérifiant l'HYPOTHESE 2.1 (v), nous arrivons à l'inégalité suivante:

$$\sigma^{(k)} \geq -\mu_0^{(k)}(f^0(\hat{x}) - y_{n+1}^{(k-1)}) - \sum_{l\in G_k}\mu_l^{(k)} g(\hat{x}, t^{(l)}). \tag{4.5}$$

A partir de (4.5), nous avons la thèse puisque, pour tout l appartenant à G_k, $\mu_l^{(k)} \in [0, 1]$ et $g(\hat{x}, t^{(l)}) \leq \max_{t\in S} g(\hat{x}, t) < 0$ par l'hypothèse 2.1 (v) mais aussi puisque $\sigma^{(k)}$ décroit vers 0.
En effet,

$$\sigma^{(k)} \geq -\mu_0^{(k)}(f^0(\hat{x}) - y_{n+1}^{(k-1)}) - \sum_{l\in G_k}\mu_l^{(k)} g(\hat{x}, t^{(l)}) \geq 0.$$

Comme $\sigma^{(k)}$ décroit vers 0 lorsque k tend vers l'infini, nous déduisons à partir de l'inégalité précédente que $\sum_{l\in G_k}\mu_l^{(k)}(-g(\hat{x}, t^{(l)}))$ tend vers 0 lorsque k tend vers l'infini, c'est-à-dire que, pour tout ε' strictement positif, il existe un indice k_0 tel que, pour tout $k \geq k_0$, nous avons $\sum_{l\in G_k}\mu_l^{(k)}(-g(\hat{x}, t^{(l)})) < \varepsilon$'.
Vu que $-g(\hat{x}, \cdot)$ est continue sur le compact S, elle atteint donc ses bornes c'est-à-dire

$0 < -g(\hat{x}, \tilde{t}) = \min_{t \in S} (-g(\hat{x}, t)) \leq -g(\hat{x}, t^{(l)})$. Posons m= $-g(\hat{x}, \tilde{t})$. Nous pouvons dès lors affirmer que, pour tout ε' strictement positif, il existe un indice k_0 tel que, pour tout $k \geq k_0$, nous avons

$$0 < m \sum_{l \in G_k} \mu_l^{(k)} \leq \sum_{l \in G_k} \mu_l^{(k)} (-g(\hat{x}, t^{(l)})) < \varepsilon' .$$

En posant $\varepsilon = \frac{\varepsilon'}{m}$, nous obtenons finalement la thèse.

Enfin, le caractère borné de $\left\{ \lambda_j^{(k)} \ \forall j \in J \right\}_k$ suit de l'hypothèse de rang plein de la matrice A. En effet, si nous dérivons le lagrangien par rapport à x appartenant à $\Re^n$, nous obtenons l'équation suivante:

$$\sum_{i \in F_k} \upsilon_i^{(k)} \nabla f(x^{(i)}) + \sum_{l \in G_k} \mu_l^{(k)} \nabla_x g(x^{(l)}, t^{(l)}) + \sum_{j \in J} \lambda_j^{(k)} a_j = 0.$$

Si nous notons $A^T \lambda^k = \sum_{j \in J} \lambda_j^{(k)} a_j$, et si nous multiplions l'équation précédente par A, nous avons

$$AA^T \lambda^k = -A(\sum_{i \in F_k} \upsilon_i^{(k)} \nabla f(x^{(i)}) + \sum_{l \in G_k} \mu_l^{(k)} \nabla_x g(x^{(l)}, t^{(l)})).$$

Comme A est une matrice de rang plein, AA^T est inversible. Nous en concluons alors que

$$\lambda^k = -(AA^T)^{-1} A(\sum_{i \in F_k} \upsilon_i^{(k)} \nabla f(x^{(i)}) + \sum_{l \in G_k} \mu_l^{(k)} \nabla_x g(x^{(l)}, t^{(l)}))$$

où le second terme est borné, ce qui termine la preuve.

□

Lemme 2.4.3

Supposons que l'algorithme ne se termine pas et notons $\underline{\mu_0} = \lim_k \inf \mu_0^{(k)}$ *et* $\overline{\mu^0} = \lim_k \sup \mu_0^{(k)}$. *Alors* $0 < \underline{\mu_0} \leq \overline{\mu^0} < +\infty$.

Si l'algorithme se termine à l'itération k, alors $\mu_0^{(k)}$ *est strictement positif.*

PREUVE

Déterminons d'abord une borne inférieure de $\mu_0^{(k)}$.

Considérons $\delta \equiv \max\left\{\max_{x\in H'}\left\|\nabla f(x)\right\|, \max_{t\in S}(\max_{x\in H'}\left\|\nabla_x g(x,t)\right\|)\right\} > 0$. Par (4.2), pour chaque k, nous avons

$$\sum_{l\in G_k}\left\|\nabla_x g(x^{(l)}, t^{(l)})\right\| \mu_l^{(k)} = 1 - \mu_0^{(k)} - \sum_{i\in F_k}\left\|\nabla f(x^{(i)})\right\| \upsilon_i^{(k)}$$

ou encore

$$\sum_{l\in G_k}\mu_l^{(k)} \geq \frac{1-\mu_0^{(k)}(1+\delta)}{\delta} \tag{4.6}$$

car δ est strictement positif, $\sum_{l\in G_k}\mu_l^{(k)}\delta \geq \sum_{l\in G_k}\mu_l^{(k)}\left\|\nabla_x g(x^{(l)}, t^{(l)})\right\|$

et $1 - \mu_0^{(k)} - \sum_{i\in F_k}\left\|\nabla f(x^{(i)})\right\| \upsilon_i^{(k)} \geq 1 - \mu_0^{(k)} - \sum_{i\in F_k}\upsilon_i^{(k)}\delta \underset{(4.1)}{=} 1 - \mu_0^{(k)}(1+\delta)$.

Pour $\hat{x}$ appartenant à $\Re^n$ un point vérifiant l'HYPOTHESE 2.1 (v), posons $\tau = \max_{t\in S} g(\hat{x}, t) < 0$ et notons par V_D le minimum de $f^0(x)$. En sachant que $V_D \leq y_{n+1}^{(k-1)}$, l'inégalité (4.5) devient

$$\sigma^{(k)} \geq \mu_0^{(k)}(V_D - f^0(\hat{x})) - \sum_{l\in G_k}\mu_l^{(k)}\tau$$

qui est équivalent par (4.6) à

$$\sigma^{(k)} \geq -\mu_0^{(k)}(f^0(\hat{x}) - V_D) - \tau\left(\frac{1-\mu_0^{(k)}(1+\delta)}{\delta}\right).$$

Nous obtenons directement que

$$\mu_0^{(k)} \geq \frac{-(\sigma^{(k)} + \frac{\tau}{\delta})}{f^0(\hat{x}) - V_D - \frac{\tau}{\delta}(1+\delta)} . \tag{4.7}$$

Pour démontrer ce lemme, nous devons envisager les deux cas suivants:

(i) Supposons que l'algorithme ne se termine pas.

Nous avons immédiatement que $\overline{\mu^0} < +\infty$ car par le lemme 2.4.2 nous avons que $\left\{\mu_0^{(k)}\right\}_k$ est bornée.

On peut écrire sans restriction que $\lim_k \mu_0^{(k)} = \underline{\mu_0}$ car on ne considère que les éléments de la suite qui convergent vers $\underline{\mu_0}$.

Puisque $\lim_k \sigma^{(k)} = 0$, (4.7) devient

$$\underline{\mu_0} \geq \frac{(-\frac{\tau}{\delta})}{f^0(\hat{x}) - V_D - \frac{\tau}{\delta}(1+\delta)} \tag{4.8}$$

Nous devons encore prouver que $\underline{\mu_0}$ est strictement positif. Pour cela, il nous suffit de montrer que le second membre de (4.8) est strictement positif. Nous avons immédiatement que le numérateur est strictement positif . Voyons maintenant que

$$f^0(\hat{x}) - V_D - \frac{\tau}{\delta}(1+\delta) > 0.$$

Nous avons que

$$f^0(\hat{x}) > V_D \tag{4.9}$$

car $\hat{x}$ est un point admissible mais n'est pas optimal.

De plus,

$$\frac{-\tau}{\delta}(1+\delta) > 0. \tag{4.10}$$

En additionant (4.9) et (4.10) membre à membre, nous obtenons $f^0(\hat{x}) - V_D - \frac{\tau}{\delta}(1+\delta) > 0.$

(ii) Supposons que l'algorithme se termine à l'itération k.

Alors, $\sigma^{(k)} = 0$ et (4.7) permet de remplacer $\underline{\mu_0}$ dans le membre de gauche de (4.8) par $\mu_0^{(k)}$.

□

Théorème 2.4.1

Soient les vecteurs duaux $\left\{\mu_0^{(k)}\right\}_k, \left\{\upsilon_i^{(k)} \quad \forall i \in F_k\right\}_k, \left\{\mu_l^{(k)} \quad \forall l \in G_k\right\}_k$ *et* $\left\{\lambda_j^{(k)} \quad \forall j \in J\right\}_k$ *optimaux pour le problème linéaire* (SD_{k-1}).

Pour $\mu_0^{(k)}$ *différent de 0, définissons*

$$\xi_l^{(k)} = \begin{cases} \dfrac{\mu_l^{(k)}}{\mu_0^{(k)}} & si\ t \in \left\{t^{(l)} \mid l \in G_k\right\} \\ 0 & si\ t \notin \left\{t^{(l)} \mid l \in G_k\right\} \end{cases}$$

$$\Psi_j^{(k)} = \frac{\lambda_j^{(k)}}{\mu_0^{(k)}} \qquad pour\ j \in J$$

Alors $\xi(t) = \xi^{(k)}(t)$ *et* $\Psi_j = \Psi_j^{(k)}$ *sont admissibles pour la forme lagrangienne du dual de (D). De plus,* $\lim_k L(\xi^{(k)}, \Psi^{(k)}, \bar{x}) = f^0(\bar{x})$, *où* $\bar{x}$ *appartenant à* $\Re^n$ *désigne maintenant la solution optimale de (D).*

PREUVE

Montrons tout d'abord la première partie du théorème c'est-à-dire que la seconde contrainte de (P.D) est vérifiée.

Comme $\mu_0^{(k)}$ est strictement positif par le lemme 2.4.3, nous avons par (4.4) que

$$\inf_{x\in\Re^n}[f^0(x)+\sum_{t\in G_k}\xi_l^{(k)}g(x,t^{(l)})+\sum_{j\in J}\Psi_j^{(k)}(a_jx-b_j)]\geq y_{n+1}^{(k-1)}-\frac{\sigma^{(k)}}{\mu_0^{(k)}}. \tag{4.11}$$

Par conséquent, la deuxième contrainte de (P.D) est satisfaite.

Montrons maintenant la deuxième partie du théorème. Etant donné ε strictement positif, il existe un indice k_0 tel que, pour tout $k \geq k_0$, nous avons

$$y_{n+1}^{(k-1)}-\left(\frac{\sigma^{(k)}}{\mu_0^{(k)}}\right)\geq f^0(\bar{x})-\varepsilon \tag{4.12}$$

en utilisant le lemme 2.4.3, et parce que $\lim_k y_{n+1}^{(k-1)} = f^0(\bar{x})$ et $\lim_k \sigma^{(k)} = 0$.

Vu que la valeur de la fonction objectif du primal en la solution réalisable du PL dual est supérieure ou égale à la valeur de la fonction objectif du dual en la solution du PL dual, $f^0(\bar{x})$ est au moins aussi grand que le membre de gauche de (4.11). Combiné avec (4.12), nous obtenons, pour tout $k \geq k_0$,

$$f^0(\bar{x}) \geq \inf_{x\in\Re^n} L(\xi^{(k)}, \Psi^{(k)}, x) \geq f^0(\bar{x})-\varepsilon$$

qui est valable pour tout ε strictement positif. Par conséquent, nous avons la thèse.

□

Théorème 2.4.2

Sur l'ensemble des points réalisables donnés par l'algorithme, la vitesse de convergence est linéaire.

PREUVE

En combinant les lemmes 2.3.3 et 2.4.3, nous avons qu'il existe un indice k_0 tel que, pour tout $k \geq k_0$, $\mu_0^{(k)}$ est strictement positif et y_{n+1} est admissible pour (D).

Considérant $x^{(k)}$ comme une solution optimale pour le problème linéaire (SD_{k-1}), le corollaire du théorème des écarts complémentaires appliqué à sa première contrainte entraîne que

$$\sigma^{(k)} = y_{n+1}^{(k-1)} - x_{n+1}^{(k)} > 0. \tag{4.13}$$

Posant $\bar{x} = \overline{x}$ l'optimum de (D) dans (4.4), nous avons alors que

$$\begin{aligned} y_{n+1}^{(k-1)} - x_{n+1}^{(k)} &\geq \mu_0^{(k)}(y_{n+1}^{(k-1)} - \overline{x}_{n+1}) - \sum_{l \in G_k} \mu_l^{(k)} g(\overline{x}, t^{(l)}) - \sum_{j \in J} \lambda_j^{(k)}(a_j \overline{x} - b_j) \\ &\geq -\mu_0^{(k)}(\overline{x}_{n+1} - y_{n+1}^{(k-1)}) \end{aligned}$$

car, à l'optimum, $g(\overline{x}, t^{(l)})$ *et* $(a_j \overline{x} - b_j)$ sont strictement négatifs et $f^0(\overline{x}) = \overline{x}_{n+1}$.

D'où, il existe $\rho \in (0, 1)$ tel que pour tout k suffisamment grand,

$$y_{n+1}^{(k-1)} - x_{n+1}^{(k-1)} \geq \rho \underline{\mu_0}(y_{n+1}^{(k-1)} - V_D) \tag{4.14}$$

où $\overline{x}_{n+1} = V_D$.

Or, quand $x^{(k)}$ est réalisable pour (D), $y^{(k)} = x^{(k)}$ et donc, à partir de (4.13), nous déduisons que $y_{n+1}^{(k-1)} > y_{n+1}^{(k)}$. Donc, pour k suffisamment grand, (4.14) (avec addition et soustraction de $\overline{x}_{n+1}$) devient

$$(y_{n+1}^{(k-1)} - \overline{x}_{n+1}) - (y_{n+1}^{(k)} - \overline{x}_{n+1}) \geq \rho \underline{\mu_0}(y_{n+1}^{(k-1)} - \overline{x}_{n+1})$$

où $y_{n+1}^{(k)} = x_{n+1}^{(k)}$.

Finalement, pour k suffisamment grand et pour $\rho \in (0, 1)$,

$$\frac{y_{n+1}^{(k)} - \overline{x}_{n+1}}{y_{n+1}^{(k-1)} - \overline{x}_{n+1}} \leq 1 - \rho \underline{\mu_0} \ .$$

Puisque la valeur de la fonction objectif de (D') est $\overline{x}_{n+1}$ et $y_{n+1}^{(k)}$ est la valeur de la fonction objectif de (D') à l'itération k, l'algorithme a une vitesse linéaire de convergence dans la valeur de la fonction objectif.

□

Chapitre 3

Implémentation de l'algorithme et exemples numériques

Dans ce troisième chapitre, nous abordons en premier lieu la réalisation de l'algorithme de plan de coupe central sur machine. Nous évoquons également certains détails d'implémentation que nous avons effectués. Dans ce paragraphe, nous donnons entre autre un aperçu de la discrétisation qui a été appliquée à chaque intervalle de S. De plus, nous montrons comment se présente le fichier externe de données dont nous avons besoin pour le bon fonctionnement du programme ainsi que la librairie que nous avons employée pour résoudre le problème linéaire qui recherche le centre sphérique.

Dans le second paragraphe de ce chapitre, nous présentons quelques exemples qui nous ont permis de tester l'algorithme implémenté et nous comparons les résultats que nous avons obtenus avec ceux de K.O. Kortanek et Hoon No.

En ce qui concerne l'analyse du problème, la façon dont il faut utiliser le programme et les listings du programme et du fichier extérieur, nous les exposons dans les annexes qui suivront immédiatement ce chapitre.

3.1 Réalisation informatique de l'algorithme et détails d'implémentation

3.1.1 Discrétisation des intervalles de S.

Lorsque nous effectuons l'algorithme de plan de coupe central, nous devons tester à un certain moment de l'itération si la solution $(x^{(k)}, \sigma^{(k)})$ du problème linéaire (SD_k) est admissible pour le problème de programmation semi-infinie convexe (D'). En d'autres termes, il nous faut vérifier en cette solution non seulement que la première contrainte $f(x) \leq 0$ est satisfaite mais également que, pour toutes valeurs de t appartenant à l'ensemble S, la deuxième contrainte $g(x, t) \leq 0$ est satisfaite.

Or, pour cette seconde contrainte, il n'est pas possible de réaliser ce test pour toutes les valeurs de t dans S. C'est pourquoi nous sommes obligés d'élaborer une discrétisation de chaque intervalle de S.

La vérification de la deuxième contrainte du problème (D') et la discrétisation des intervalles de l'ensemble S s'effectuent en même temps et de la manière suivante:

Supposons que l'ensemble S = [a,b]. Nous déterminons d'abord le milieu de cet intervalle [a,b] qui est ici $\frac{a+b}{2}$. Nous regardons ensuite si l'élément $(x^{(k)}, \frac{a+b}{2})$ ne satisfait pas la seconde condition de (D'). Si c'est le cas, nous concluons alors que la solution $(x^{(k)}, \sigma^{(k)})$ n'est pas admissible pour le problème convexe (D') et nous arrêtons la discrétisation. Sinon, nous considérons les deux intervalles $\left[a, \frac{a+b}{2}\right]$ et $\left[\frac{a+b}{2}, b\right]$. Nous réalisons le même travail que pour l'intervalle initial. Si la condition a été vérifiée pour le milieu du premier de ces deux intervalles, nous passons au second intervalle. Si nous arrivons à la conclusion que la contrainte est vérifiée pour le centre de ces deux intervalles, nous envisageons alors quatre intervalles et ainsi de suite ...

jusqu'à ce que, soit nous trouvons une valeur de t appartenant à l'ensemble S qui ne satisfait pas la seconde contrainte du problème (D'), soit nous déterminons le centre d'un intervalle se situant à une distance de la frontière de S qui est inférieure ou égale à la tolérance imposée par l'utilisateur.

3.1.2 Structure du fichier externe.

Dans le but de faciliter l'utilisation du programme, pour éviter que l'utilisateur ne modifie la réalisation informatique de l'algorithme et pour que celui-ci ne soit pas obligé d'introduire des données au cours de l'exécution du programme, nous avons créé un fichier externe DONNEES.FOR qui devra contenir toutes les données nécessaires au bon fonctionnement du programme.
En fait, le fichier externe est composé des éléments suivants:

1) une fonction entière qui contiendra le nombre maximum de contraintes que le programme peut gérer. De cette manière, l'utilisateur imposera à l'ordinateur de ne pas devoir résoudre des systèmes trop importants.

2) une fonction entière contenant le nombre maximum d'itérations que le programme pourra effectuer. Grâce à cette fonction, l'utilisateur empêchera une trop longue exécution du programme.

3) une fonction entière comprenant la dimension de l'espace S. Cette donnée sera utilisée pour la discrétisation de chaque intervalle de l'ensemble S.

4) une fonction entière contenant la dimension de l'espace H à laquelle nous ajoutons deux unités car dans la recherche de la solution du problème linéaire (SD_k), nous avons deux variables supplémentaires: x_{n+1} et σ.

5) deux fonctions double précision contenant respectivement le maximum et le minimum de la fonction objectif sur l'ensemble H. Nous avons besoin de ceux-ci car la composante x_{n+1} du vecteur x ne peut varier qu'entre ces deux valeurs.

6) une routine qui contient les valeurs des trois variables suivantes: PRECIS donnant la précision voulue par l'utilisateur pour la solution, TOLER indiquant la tolérance à laquelle les points que nous considérons lors de la discrétisation doivent se trouver de la frontière de l'ensemble S et la valeur de la constante BETA strictement comprise entre 0 et 1.

7) une routine contenant la longueur et la borne inférieure de chaque intervalle de l'ensemble S. Ces informations seront utilisées lorsque le programme effectuera la discrétisation de S.

8) une routine comprenant les bornes inférieure et supérieure de chaque variable de H et la borne supérieure de la contrainte générale du problème linéaire initial (SD_0) du PAS 0 de l'algorithme.

9) une fonction double précision contenant l'expression de la fonction f.

10) une routine contenant l'expression du gradient de la fonction f.

11) une fonction double précision comprenant l'expression de la fonction g.

12) une routine qui contient l'expression du gradient de la fonction g.

3.1.3 Remarques: Détails d'implémentation.

(a) Lors de la description de l'algorithme, nous avons vu qu'il fallait introduire au départ une solution réalisable $y^{(0)}$ appartenant à H'. Par contre, nous ne demandons pas au cours de l'exécution du programme l'introduction d'une telle solution, ce qui n'empêche pas le programme de bien fonctionner.

(b) Pour implémenter l'algorithme sur machine, nous devons résoudre le problème linéaire (SD_k) à chaque itération. C'est pourquoi nous utilisons la sous-routine DDPLRS de la librairie IMSL [6] qui résoud un problème linéaire de la forme

$$\begin{cases} \min\limits_{x\in\Re^n} c^T x \\ s.c. \\ \qquad b_l \le Ax \le b_u \\ \qquad x_l \le x \le x_u \end{cases}$$

via l'algorithme du simplexe révisé.

Il est important de noter que lorsque la borne inférieure d'une variable x n'est pas spécifiée, nous lui imposons la valeur **1.0E30** pour que l'on puisse utiliser DDPLRS. Par contre, si nous n'avons pas de borne supérieure pour la variable x, nous lui fixons la valeur **-1.0E30** .

3.2 Exemples numériques

Dans ce paragraphe, nous présentons quelques exemples qui nous ont permis de tester la réalisation informatique faite. Nous comparons les résultats obtenus avec ceux décrits dans l'article de Kortanek et No.

Exemples

Exemple 1: (Tichatschke et Nebeling [3])

Les données introduites dans le fichier externe sont les suivantes:

$f(x) = (x_1 - 2)^2 + (x_2 - 0.2)^2$.

$g(x,t) = \left(\dfrac{5\sin(\pi\sqrt{t})}{(1+t^2)}\right) x_1^2 - x_2$.

$H = \left\{x \in \Re^2 \mid -1 \le x_1 \le 1\,,\, 0 \le x_2 \le 0.2\right\}$.

$S = \left\{t \in \Re \mid 0 \le t \le 8\right\}$.

Tolérance à la frontière de S = 10^{-2} .

Précision demandée = 10^{-10} .

La constante $\beta = 0.75$.

La borne supérieure $\bar{f} = 4$.

Les résultats donnés sont:

	Résultats obtenus par Kortanek et No	Résultats donnés par le programme sans la règle de suppression 2	Résultats donnés par le programme avec la règle de suppression 2
x_1	**0.205236774**	**0.20525761**	**0.20525761**
x_2	**0.2**	**0.2**	**0.2**
Valeur de la fonction objectif en la solution.	**3.22117504**	**3.2211002**	**3.2211002**
Nombres d'itérations effectuées.	-	**160**	**160**
Nombres de contraintes ajoutées suivant la fonction f.	-	**4**	**4**
Nombre de contraintes ajoutées suivant la fonction g.	-	**4**	**4**
Temps CPU écoulé (secondes).	**0.37**	**30.74**	**28.88**

Exemple 2: (Polak et He [5]; Tanaka,Tukushima et Ibaraki [4])

Les données introduites dans le fichier externe sont les suivantes:

$f(x) = x_1^2 + x_2^2 + x_3^2$.

$g(x, t) = x_1 + x_2\, e^{x_3 t} + e^{2t} - 2\sin(4t)$.

$H = \{x \in \Re^3 \mid -2 \le x_1 \le 2, -2 \le x_2 \le 2, -2 \le x_3 \le 2\}$.

$S = \{t \in \Re \mid 0 \le t \le 1\}$.

Tolérance à la frontière de S = 10^{-2} .

Précision demandée = 10^{-10} .

La constante $\beta = 0.75$.

La borne supérieure $\bar{f} = 6$.

Les résultats donnés sont:

	Résultats obtenus par Kortanek et No	Résultats donnés par le programme sans la règle de suppression 2	Résultats donnés par le programme avec la règle de suppression 2
x_1	**- 0.213312578**	**- 0.53288926**	**- 0.53288926**
x_2	**- 1.36145045**	**- 1.3585124**	**- 1.3585124**
x_3	**1.85354733**	**1.8085374**	**1.8085374**
Valeur de la fonction objectif en la solution.	**5.33468728**	**5.4003347**	**5.4003347**
Nombres d'itérations effectuées.	-	**207**	**207**

Nombres de contraintes ajoutées suivant la fonction f.	-	**23**	**23**
Nombre de contraintes ajoutées suivant la fonction g.	-	**6**	**6**
Temps CPU écoulé (secondes).	**2.71**	**41.78**	**16.35**

Exemple 3: (Tichatschke et Nebeling [3])

Les données introduites dans le fichier externe sont les suivantes:

$f(x) = x_1^2 + x_2^2$.

$g(x,t) = \left((x_1-2)^2 + (x_2-2)^2 - 4\right) t_1 + (x_1^2 + x_2^2 - 4)\, t_2$.

$H = \left\{x \in \Re^2 \mid 0 \le x_1 \le 2\,,\, 0 \le x_2 \le 2\right\}$.

$S = \left\{t \in \Re^2 \mid 0 \le t_1 \le 1\,,\, 0 \le t_2 \le 1\right\}$.

Tolérance à la frontière de S = 10^{-2} .

Précision demandée = 10^{-10} .

La constante $\beta = 0.75$.

La borne supérieure $\bar{f} = 1$.

Les résultats donnés sont:

	Résultats obtenus par Kortanek et No	Résultats donnés par le programme sans la règle de suppression 2	Résultats donnés par le programme avec la règle de suppression 2
x_1	**0.585786438**	**0.57646554**	**0.57645303**

x_2	**0.585786438**	**0.57645078**	**0.57646330**
Valeur de la fonction objectif en la solution.	**0.686291501**	**0.66460803**	**0.66460802**
Nombres d'itérations effectuées.	-	**88**	**91**
Nombres de contraintes ajoutées suivant la fonction f.	-	**18**	**17**
Nombre de contraintes ajoutées suivant la fonction g.	-	**12**	**15**
Temps CPU écoulé (secondes).	**0.30**	**64.91**	**60.63**

La solution exacte de ce problème de programmation semi-infinie convexe est en fait

$$(x_1, x_2) = (2-\sqrt{2}, 2-\sqrt{2}).$$

3.2.2 Remarque

Nous remarquons dans chaque exemple une certaine différence entre les résultats obtenus par K.O. Kortanek et Hoon No et ceux donnés par notre implémentation. Cette différence peut s'expliquer par plusieurs raisons:

- La borne supérieure de la contrainte générale du problème linéaire initial (SD_0) n'a pas été spécifiée par K.O. Kortanek et Hoon No.

- La discrétisation appliquée aux différents intervalles de l'ensemble S n'a pas été exposée dans l'article de Kortanek et No.

- K.O. Kortanek et Hoon No n'ont pas implémenté l'algorithme comme il a été décrit au paragraphe 2.2 du second chapitre. En effet, ils ont utilisé un système non linéaire pour accélérer la convergence de l'algorithme. C'est pourquoi ils ont obtenu un temps CPU beaucoup plus court.

ANNEXES

A.1. Analyse du problème.

Dans cette annexe, nous allons présenter tout d'abord la situation initiale du problème c'est-à-dire les variables dont nous avons besoin pour élaborer le programme. Ensuite, nous donnons la décomposition TOP-DOWN du problème et la situation finale c'est-à-dire la situation à laquelle nous devons aboutir. Nous exposons enfin les modules employés dans le programme.

A.1.1 Situation initiale.

(i) **Les constantes:**

entières:

NBMAX : constante entière que le nombre maximum d'itérations et le nombre maximum de contraintes ne peuvent pas dépasser.

DIMMAX : constante entière que les dimensions de S et H ne peuvent pas dépasser.

(ii) **Les variables :**

entières:

MAXITE : variable entière contenant le nombre maximum d'itérations que le programme peut effectuer.

MAXCON : variable entière contenant le nombre maximum de contraintes que le programme peut gérer.

NBITER : variable entière contenant le nombre d'itérations que l'on a effectuées.

NBCONT : variable entière contenant le nombre de contraintes que le programme gère pour le moment.

DIMH : variable entière contenant la dimension de H à laquelle on a ajouté deux unités.

DIMS : variable entière contenant la dimension de S.

NBCOUF : variable entière contenant le nombre de coupe que l'on effectue suivant la fonction f.

NBCOUG : variable entière contenant le nombre de coupe que l'on effectue suivant la fonction g.

OK : variable entière qui contient la valeur 1 si les données introduites dans le fichier externe vérifie les conditions imposées par le programme et l'algorithme et la valeur 0 sinon.

réelle:

BETA : variable réelle contenant une valeur qui sert dans la règle de suppression 2 de l'algorithme et qui est comprise strictement entre 0 et 1.

caractère:

REP : variable caractère contenant la réponse aux questions posées à l'utilisateur.

double précision:

PRECIS : variable double précision contenant la précision que la solution du problème doit avoir.

TOLER : variable double précision contenant la tolérance à laquelle on s'arrête de la frontière de S lors de la discrétisation.

(iii) **Les vecteurs :**

d'entiers:

PASCON : vecteur d'entiers contenant la valeur 1 si la contrainte a été engendrée par le PAS 3 (i) de l'algorithme ou la valeur 2 si la contrainte a été engendrée par les PAS 3 (ii) ou 3 (iii).

ITERA : vecteur d'entiers contenant l'itération à laquelle la contrainte a été introduite dans le problème de recherche sphérique.

REALIS : vecteur d'entiers de dimension 2 contenant dans sa première composante la valeur 1 si $f(x)>0$ et 0 sinon et sa deuxième composante la valeur 1 si $g(x,t)>0$ et 0 sinon.

IRTYPE : vecteur d'entiers contenant dans chaque élément la valeur 1 indiquant que les contraintes générales du problème de maximisation (SD_k) sont des inégalités plus petites ou égales.

double précision:

BORNIN : vecteur double précision contenant la borne inférieure de chaque intervalle de S.

LONGIN : vecteur double précision contenant la longueur de chaque intervalle de S.

BU : vecteur double précision contenant la borne supérieure des contraintes générales .

C : vecteur double précision contenant les coefficients de chaque variable de l'expression de la fonction objectif.

XLB : vecteur double précision contenant les bornes inférieures des variables.

XUB : vecteur double précision contenant les bornes supérieures des variables.

SIGMA : vecteur double précision contenant des valeurs de sigma qui ont été calculées à certaine itération.

VALEUR : vecteur double précision contenant les valeurs de S qui rendent la fonction g négative.

BL : vecteur double précision contenant les bornes inférieures des contraintes générales du problème de maximisation (SD_k).

OBJ : vecteur double précision contenant la valeur de la fonction en la solution optimale du problème de maximisation (SD_k) au cours de chaque itération.

XSOL : vecteur double précision contenant la solution du problème de maximisation (SD_k) au cours de chaque itération.

DSOL : vecteur double précision contenant la solution optimale du dual du problème de maximisation (SD_k) au cours de chaque itération.

(iv) **Les matrices :**

double précision:

A : matrice double précision contenant les coefficients de chaque variable des contraintes générales.

SOLUTI : matrice double précision contenant la valeur de la solution du problème de maximisation (SD_k) à chaque itération.

De plus, avant de commencer à exécuter le programme, il faut introduire dans le fichier externe DONNEES.FOR les données suivantes:

MAX1 : fonction entière qui contiendra le nombre maximum de contraintes que le programme pourra gérer.

MAX2 : fonction entière qui contiendra le nombre maximum d'itérations que le programme pourra effectuer.

DIMEN1 : fonction entière qui contiendra la dimension de H à laquelle on ajoutera deux unités (cela correspond au fait que l'on introduit une composante supplémentaire pour x et une variable sigma pour rechercher le centre sphérique).

DIMEN2 : fonction entière qui contiendra la dimension de S.

MXMF : fonction double précision qui contiendra le maximum de la fonction f sur H.

MNMF : fonction double précision qui contiendra le minimum de la fonction f sur H.

CONST : routine qui contiendra les valeurs des variables double précision BETA, PRECIS, TOLER.

BORLON : routine qui contiendra la longueur et la borne inférieure de chaque intervalle de S.

BORNES : routine qui contiendra la borne supérieure de la première contrainte générale du problème de maximisation du PAS 0 de l'algorithme, ainsi que les bornes inférieure et supérieure de chaque variable

FONF : fonction double précision qui contiendra l'expression de la fonction f.

FONG : fonction double précision qui contiendra l'expression de la fonction g.

GRF : routine qui contiendra l'expression du gradient de f.

GRG : routine qui contiendra l'expression du gradient de g.

A.1.2 La décompostion Top-Down.

1. Demander à l'utilisateur s'il a introduit toutes les données requises dans le fichier externe DONNEES.FOR .
2. Tester s'il n'a pas introduit les données requises.
3. Si oui, sortir du programme pour permettre à l'utilisateur d'introduire ces données.

4. Introduction des données nécessaires au bon fonctionnement du programme.
   4.1 Initialisation de la variable MAXCON à la valeur introduite dans DONNEES.FOR .
   4.2 Tester si MAXCON est supérieur à NBMAX.
   4.3 Si oui, on sort du programme.
   4.4 Initialisation de la variable MAXITE à la valeur introduite dans DONNEES.FOR .
   4.5 Tester si MAXITE est supérieur à NBMAX.
   4.6 Si oui, on sort du programme.
   4.7 Initialisation de la variable DIMH à la valeur introduite dans DONNEES.FOR .
   4.8 Tester si DIMH est supérieur à DIMMAX.
   4.9 Si oui, on sort du programme.
   4.10 Initialisation de la variable DIMS à la valeur introduite dans DONNEES.FOR .
   4.11 Tester si DIMS est supérieur à DIMMAX.
   4.12 Si oui, on sort du programme.
   4.13 Initialisation des valeurs BETA, PRECIS, TOLER aux valeurs introduites dans DONNEES.FOR .
   4.14 Tester si BETA n'a pas une valeur comprise strictement entre 0 et 1
   4.15 Si oui, on sort du programme.
   4.16 Initialisation de la borne inférieure et de la longueur de chaque intervalle de S aux valeurs introduites dans DONNEES.FOR .
   4.17 Initialisation de la variable NBCONT et du vecteur PASCON(1) à 1.
   4.18 Initialisation des bornes supérieure et inférieure de chaque variable de H et de la borne supérieure de la contrainte générale du problème linéaire (SD_0) de départ.

   Pour I=1,DIMH-2

   4.19 Introduction dans A des coefficients des variables de H de la contrainte générale.

   4.20 Introduction des bornes inférieure et supérieure de σ et de x_{n+1} et des coefficients de ceux-ci de la contrainte générale dans A.

Pour I=1,DIMH-1

4.21 Introduction dans C des coefficients des variables de H et de x_{n+1} de la fonction objectif.

4.22 Introduction dans C du coefficient de σ de la fonction objectif.

5. Initialisation de la variable NBITER à 1 et des variables NBCOUF et NBCOUG à 0.
6. Tester si le nombre de contraintes que le programme gère est inférieur à MAXCON.
7. Si oui, voir si on n'a pas atteint le nombre maximum d'itérations.
   7.1 Tester si le nombre d'itérations actuel est inférieur à MAXITE
   7.2 Si oui, application de l'algorithme de plan de coupe central.
      7.2.1 Détermination de la solution du problème de maximisation par la sous-routine DDPLRS.
      7.2.2 Stockage de la solution dans les vecteurs SOLUTI et SIGMA.
      7.2.3 Tester si la solution n'a pas la précision demandée par l'utilisateur.
      7.2.4 Si oui, voir quel type de contrainte il faut ajouter au problème linéaire (SD_k).
         7.2.4.1 Voir si la solution XSOL obtenue est admissible pour le problème (D) de départ.
            7.2.4.1.1 Tester si f(XSOL) sera strictement positif.
            7.2.4.1.2 Si oui, mettre REALIS(1) à 1.
            7.2.4.1.3 Sinon, mettre REALIS(1) à 0.
            7.2.4.1.4 Voir s'il y a une valeur de t dans S tel que g(XSOL,t)>0.
               7.2.4.1.4.1 Tester si DIMS est 1.
               7.2.4.1.4.2 Si oui, discrétisation de l'intervalle où chaque valeur est placée dans VALEUR(1) et puis regarder si nous avons g(XSOL,VALEUR(1))>0. Si c'est le cas, mettre 1 dans REALIS(2) et 0 sinon.
               7.2.4.1.4.3 Sinon, discrétisation de chaque intervalle de S où les valeurs sont placées dans VALEUR(1) et VALEUR(2) et puis regarder si g(XSOL,(VALEUR(1),VALEUR(2))>0. Si c'est le cas, mettre 1 dans REALIS(2) et 0 sinon.
            7.2.4.1.5 Tester si REALIS(1) et REALIS(2) sont égaux à 0.
            7.2.4.1.6 Si oui, supprimer une contrainte qui a été ajoutée par le PAS 3 (i).

7.2.4.1.6.1 Tester si nous avons une contrainte engendrée à une itération précédente.

7.2.4.1.6.2 Si oui, tester si la contrainte a été engendrée par le PAS 3 (i).

7.2.4.1.6.3 Si oui, retirer tous les renseignements sur la contrainte des vecteurs ITERA, PASCON, BU, SIGMA et du tableau A et retrancher également 1 à NBCONT.

7.2.4.1.6.4 Sinon, passer à la contrainte suivante et retourner en 7.2.4.1.6.1

7.2.4.2 Voir si on peut supprimer des contraintes qui ont été engendrées par les PAS 3 (ii) ou (iii).

7.2.4.2.1 Tester si la contrainte a été engendrée à une itération précédente.

7.2.4.2.2 Si oui, voir si on peut supprimer la contrainte.

7.2.4.2.2.1 Tester si la contrainte a été engendrée par les PAS 3 (ii) ou (iii).

7.2.4.2.2.2 Si oui, voir si la contrainte vérifie la condition $\sigma^{(k)} \geq \beta\sigma^{(j)}$.

7.2.4.2.2.2.1 Tester si on a $\sigma^{(k)} \geq \beta\sigma^{(j)}$.

7.2.4.2.2.2.2 Si oui, voir si la contrainte n'est pas active en $(x^{(k)}, \sigma^{(k)})$.

7.2.4.2.2.2.2.1 Tester si la contrainte n'est pas active en $(x^{(k)}, \sigma^{(k)})$.

7.2.4.2.2.2.2.2 Si oui, retirer tous les renseignements sur la contrainte des vecteurs ITERA, PASCON, BU, SIGMA et du tableau A et retrancher également 1 à NBCONT.

7.2.4.2.2.2.2.3 Sinon, passer à la contrainte suivante et retourner en 7.2.4.2.1 .

7.2.4.2.2.2.3 Sinon, passer à la contrainte suivante et retourner en 7.2.4.2.1 .

7.2.4.2.2.3 Sinon, passer à la contrainte suivante et retourner en 7.2.4.2.1 .

7.2.4.3 Tester si REALIS(1) est 1.

7.2.4.4 Si oui, ajouter une contrainte suivant la fonction f (les renseignements seront placés dans A, BU. De plus, nous mettons 2 dans PASCON et nous ajoutons 1 à NBCONT et NBCOUF).

7.2.4.5 Sinon, voir si REALIS(2) est 1.

7.2.4.5.1 Tester si REALIS(2) est 1.

7.2.4.5.2 Si oui, ajouter une contrainte suivant la fonction g (les renseignements seront placés dans A, BU. De plus, nous mettons 2 dans PASCON et nous ajoutons 1 à NBCONT et NBCOUG).

7.2.4.5.3 Sinon, ajouter une contrainte du type $x_{n+1} + \sigma \leq x_{n+1}^{(k)}$ (les renseignements seront placés dans A, BU. De plus, nous mettons 1 dans PASCON et nous ajoutons 1 à NBCONT et NBCOUF).

7.2.4.6 Ajouter 1 à NBITER er retourner en 6.

7.2.5 Sinon, voir si l'utilisateur veut afficher les résultats à l'écran.

7.2.5.1 Demander à l'utilisateur s'il veut afficher les résultats à l'écran.

7.2.5.2 Tester si l'utilisateur veut afficher les résultats à l'écran.

7.2.5.3 Si oui, affichage du nombre d'itérations effectuées, de la solution trouvée, de la valeur de f en la solution optimale et du nombre de contraintes ajoutées suivant la fonction f et la fonction g.

7.3 Sinon, voir si l'utilisateur veut afficher quand même les résultats à l'écran.

7.3.1 Demander à l'utilisateur s'il veut afficher les résultats à l'écran.

7.3.2 Tester si l'utilisateur veut afficher les résultats à l'écran.

7.3.3 Si oui, affichage du nombre d'itérations effectuées, de la solution trouvée, de la valeur de f en la solution optimale et du nombre de contraintes ajoutées suivant la fonction f et la fonction g.

8. Sinon, voir si l'utilisateur veut afficher quand même les résultats à l'écran.

8.1 Demander à l'utilisateur s'il veut afficher les résultats à l'écran.

8.2 Tester si l'utilisateur veut afficher les résultats à l'écran.

8.3 Si oui, affichage du nombre d'itérations effectuées, de la solution trouvée, de la valeur de f en la solution optimale et du nombre de contraintes ajoutées suivant la fonction f et la fonction g.

A.1.3 La situation finale.

Lorsque le programe a déterminé la solution du problème de programmation semi-infinie convexe, nous affichons à l'écran, si l'utilisateur le désire, le nombre d'itérations effectuées, la solution trouvée, la valeur de f en la solution optimale et le nombre de contraintes ajoutées suivant la fonction f et la fonction g.

A.1.4 Les modules.

Titre: But: Cette routine affiche à l'écran ce que le programme va effectuer.

Variables d'entrée: /

Variables de sortie: /

Variables d'entrée-sortie: /

Intro: But: Cette routine introduit dans les différents vecteurs et matrice les renseignements nécessaires au bon fonctionnement du programme.

Variables d'entrée: NBMAX,DIMMAX.

Variables de sortie: DIMH, DIMS, NBCONT, BETA, TOLER, PRECIS, LONGIN, BORNIN, PASCON, ITERA, BU, C, XLB, XUB, A,MAXITE, MAXCON, OK.

Variables d'entrée-sortie: /

Admiss: But: Cette routine regarde si la solution donnée par la sous-routine DDPLRS est admissible pour le problème (D). Si c'est le cas, on supprime les contraintes engendrées par le PAS 3 (i) à une itération précédente.

Variables d'entrée: NBMAX, DIMMAX, DIMH, DIMS, TOLER, XSOL, LONGIN, BORNIN, NBITER.

Variables de sortie: VALEUR, REALIS.

Variables d'entrée-sortie: SIGMA, A, BU, ITERA, PASCON, NBCONT.

Suppr2: But: Cette routine supprime éventuellement une ou plusieurs contraintes engendrées par les PAS 3 (ii) ou (iii) à une itération précédente.

Variables d'entrée: NBMAX, DIMMAX, DIMH, NBITER, XSOL, BETA, PRECIS.

Variables de sortie: /

Variables d'entrée-sortie: A, BU, PASCON, ITERA, SIGMA, NBCONT.

Ajout2: But: Cette routine ajoute une contrainte suivant la fonction f dans les vecteurs et matrice prévus à cet effet.

Variables d'entrée: NBMAX, DIMMAX, DIMH, XSOL.

Variables de sortie: /

Variables d'entrée-sortie: A, BU, PASCON, NBCONT, NBCOUF.

Ajout3: But: Cette routine ajoute une contrainte suivant la fonction g dans les vecteurs et matrice prévus à cet effet.

Variables d'entrée: NBMAX, DIMMAX, DIMS, DIMH, VALEUR, XSOL.

Variables de sortie: /

Variables d'entrée-sortie: A, BU, PASCON, NBCONT, NBCOUG.

Ajout1: But: Cette routine ajoute une contrainte du type $x_{n+1} + \sigma \leq x_{n+1}^{(k)}$ dans les vecteurs et matrice prévus à cet effet.

Variables d'entrée: NBMAX, DIMMAX, DIMH, XSOL.

Variables de sortie: /

Variables d'entrée-sortie: A, BU, PASCON, NBCONT.

Affich2: But: Cette routine a pour but d'afficher à l'écran le nombre d'itérations effectuées, la solution trouvée, la valeur de f en la solution optimale et le nombre de contraintes ajoutées suivant la fonction f et suivant la fonction g.

Variables d'entrée: NBMAX, DIMMAX, DIMH, NBITER, NBCONT, SOLUTI, XSOL, NBCOUF, NBCOUG.

Variables de sortie: /

Variables d'entrée-sortie: /

Test: But: Cette routine regarde s'il y a une valeur de t dans S qui ne vérifie pas la deuxième condition de (D) en la solution.

Variables d'entrée: DIMMAX, DIMH, DIMS, TOLER, LONGIN, BORNIN, XSOL.

Variables de sortie: VALEUR, REALIS.

Variables d'entrée-sortie: /

Discre: But: Cette routine regarde s'il y a une valeur de t dans S qui ne vérifie pas la deuxième condition de (D) en la solution (ceci dans le cas où S est de dimension 1).

Variables d'entrée: DIMMAX, DIMH, DIMS, TOLER, LONGIN, BORNIN, XSOL.

Variables de sortie: VALEUR, REALIS.

Variables d'entrée-sortie: /

A.2 Comment utiliser le programme ?

Pour pouvoir utiliser le programme MEM.FOR correctement, nous devons effectuer les étapes suivantes:

1) Taper: **EMACS DONNEES.FOR** , afin d'accéder au fichier externe pour vérifier ou introduire les données dont nous avons besoin pour pouvoir exécuter le programme. La non-initialisation d'une de ces données entrainerait des erreurs.

2) Taper: **COMPILE DONNEES.FOR** si le fichier externe a été modifié, ceci dans le but de compiler ce fichier.

3) Taper: **LINK MEM,DONNEES,IMSL/LIB** pour créer un lien entre notre programme, le fichier externe et la librairie IMSL.

4) Taper: **RUN MEM.FOR** . Cette instruction exécute le programme.

En ce qui concerne l'exécution du programme, elle se déroulera de la manière suivante: tout d'abord, le programme demande à l'utilisateur s'il a introduit toutes les données dans le fichier externe. Si l'utilisateur tape N ou n, le programme s'arrête immédiatement pour que celui-ci puisse modifier DONNEES.FOR et ensuite faire à nouveau les quatres opérations décrites ci-dessus. Sinon, l'ordinateur affichera à l'écran un texte expliquant ce que le programme résout. Ensuite, l'ordinateur recherchera la solution du problème et quand il l'aura trouvée, il demandera à l'utilisateur s'il veut afficher les résultats à l'écran. Si c'est le cas, il y aura affichage du nombre d'itérations effectuées, de la solution optimale, de la valeur de la fonction objective en la solution et le nombre de contraintes ajoutées suivant les fonctions f et g.

A.3 Listing du programme

```
      PROGRAM ALGOR

C     ************************************************************************
C     *                          DECLARATION DES VARIABLES                   *
C     ************************************************************************
C
C     ************************************************************************
C     * NBMAX     : constante entiere que le nombre maximum d'iterations et le *
C     *             le nombre maximum de contraintes ne peuvent pas depasser.  *
C     * DIMMAX    : constante entiere que les dimensions de H et S ne peuvent  *
C     *             pas depasser.                                              *
C     * I,J       : variable entiere de comptage.                              *
C     * NBITER    : variable entiere contenant le nombre d'iterations que l'on *
C     *             a effectuees lors de l'execution du programme.             *
C     * NBCONT    : variable entiere contenant le nombre de contraintes que le *
C     *             programme gere.                                            *
C     * PASCON    : vecteur d'entiers contenant la valeur 1 si la coupe a ete  *
C     *             engendree par le PAS 3 (i) et la valeur 2 si la contrainte *
C     *             a ete engendree par les PAS 3 (ii) et (iii). De plus, ce   *
C     *             vecteur est de dimension maximale NBMAX.                   *
C     * ITERA     : vecteur d'entiers contenant l'iteration a laquelle la      *
C     *             contrainte a ete introduite dans le probleme de recherche  *
C     *             du centre spherique.                                       *
C     * REALIS(2): vecteur d'entiers a deux composantes contenant dans sa      *
C     *             premiere partie la valeur 1 si f(x)>0 et 0 sinon et sa     *
C     *             deuxieme composante contient la valeur 1 si g(x,t)>0 et    *
C     *             la valeur 0 sinon.                                         *
C     * DIMH      : variable entiere contenant la dimension de H a laquelle on *
C     *             a ajoute 2.                                                *
C     * DIMS      : variable entiere contenant la dimension de S.              *
C     * MAXCON    : variable entiere contenant le nombre maximum de contraintes*
C     *             que le programme peut gerer.                               *
C     * MAXITE    : variable entiere contenant le nombre maximum d'iterations  *
C     *             que le programme peut effectuer.                           *
C     * NBCOUF    : variable entiere contenant le nombre de coupe que l'on     *
C     *             effectue suivant la fonction f.                            *
C     * NBCOUG    : variable entiere contenant le nombre de coupe que l'on     *
C     *             effectue suivant la fonction g.                            *
C     * OK        : variable entiere qui contient la valeur 1 si les donnees   *
C     *             introduites dans le fichier externe verifie les conditions *
C     *             imposees par l'algorithme et le programme et la valeur 0   *
C     *             sinon.                                                     *
C     * BETA      : variable reelle contenant une valeur qui sert dans la regle*
C     *             de suppression 2 de l'algorithme.                          *
C     * REP       : variable caractere contenant la reponse aux questions      *
C     *             posees a l'utilisateur.                                    *
C     * IRTYPE    : vecteur d'entiers qui contient la valeur 1 dans chacun de  *
C     *             ces elements indiquant que les contraintes du probleme sont*
C     *             des inegalites "plus petite ou egale" et de dimension      *
C     *             maximale NBMAX.                                            *
C     * BORNIN    : vecteur double precision contenant la borne inferieure de  *
C     *             chaque intervalle de S et de dimension maximum DIMMAX.     *
C     * LONGIN    : vecteur double precision contenant la longueur de chaque   *
C     *             intervalle de S et de dimension maximale DIMMAX.           *
C     * PRECIS    : variable double precision contenant le precision demandee  *
C     *             pour la solution du probleme.                              *
C     * TOLER     : variable double precision contenant la tolerance a laquelle*
C     *             on s'arrete de la frontiere de S lors de la discretisation.*
C     * BU        : vecteur double precision contenant la borne superieure des *
C     *             contraintes generales et de dimension maximum NBMAX.       *
C     * C         : vecteur double precision contenant les coefficients de     *
C     *             chaque variable de la fonction objectif et de dimension    *
C     *             maximale DIMMAX.                                           *
C     * XLB       : vecteur double precision contenant les bornes inferieures  *
C     *             de chaque variable et de dimension maximale DIMMAX.        *
C     * XUB       : vecteur double precision contenant les bornes superieures  *
C     *             de chaque variable et de dimension DIMMAX.                 *
C     * SIGMA     : vecteur double precision contenant des valeurs de sigma    *
```

```
C     *            qui ont ete calculees a certaine iteration et de dimension *
C     *            maximale MNMAX.                                            *
C     * VALEUR   : vecteur double precision contenant les valeurs de chaque   *
C     *            variable de S qui rend la fonction g negative et de        *
C     *            dimension maximale DIMMAX.                                 *
C     * A        : tableau double precision contenant les coefficients de     *
C     *            chaque variable des contraintes generales et de dimension  *
C     *            maximale NMMAX x DIMMAX.                                   *
C     * BL       : vecteur double precision contenant les bornes inferieures  *
C     *            des contraintes generales et de dimension maximale NBMAX.  *
C     * SOLUTI   : tableau double precision contenant la valeur de la solution*
C     *            du probleme de maximisation a chaque iteration et de       *
C     *            de dimension maximale NBMAX x DIMMAX.                      *
C     * OBJ      : variable double precision contenant la valeur de la        *
C     *            fonction a la solution optimale du probleme de maximisation*
C     *            a chaque iteration.                                        *
C     * XSOL     : vecteur double precision contenant la solution optimale du *
C     *            probleme de maximisation a chaque iteration et de dimension*
C     *            maximale DIMMAX.                                           *
C     * DSOL     : vecteur double precision contenant la solution optimale du *
C     *            dual du probleme de maximisation a chaque iteration et     *
C     *            de dimension maximale NBMAX.                               *
C     ************************************************************************
C
      INTEGER NBMAX,DIMMAX
      PARAMETER (NBMAX=600,DIMMAX=10)
      INTEGER I,J,NBITER,NBCONT,IRTYPE(NBMAX),OK
      INTEGER PASCON(NBMAX),ITERA(NBMAX),REALIS(2)
      INTEGER DIMH,DIMS,MAXCON,MAXITE,NBCOUF,NBCOUG
      REAL BETA
      CHARACTER REP
      DOUBLE PRECISION BORNIN(DIMMAX),LONGIN(DIMMAX)
      DOUBLE PRECISION PRECIS,TOLER,BU(NBMAX),C(DIMMAX),XLB(DIMMAX)
      DOUBLE PRECISION XUB(DIMMAX),SIGMA(NBMAX)
      DOUBLE PRECISION VALEUR(DIMMAX),A(NBMAX,DIMMAX),BL(NBMAX)
      DOUBLE PRECISION SOLUTI(NBMAX,DIMMAX)
      DOUBLE PRECISION OBJ,XSOL(DIMMAX),DSOL(NBMAX)



      DATA IRTYPE/NBMAX*1/

C     ************************************************************************
C     *                          PROGRAMME PRINCIPAL                         *
C     ************************************************************************
C
      WRITE(*,900)
  900 FORMAT(X,'Avez-vous introduit tous les renseignements ',
     $       X,'necessaires au bon fonctionnement ',/,
     $       X,'du programme dans le fichier externe DONNEES.FOR ?',
     $       X,'(N ou n = non): ')
      READ(*,950) REP
  950 FORMAT(A1)
      WRITE(*,*)
C     ------------------------------------------------------------------
C     La condition suivante teste si on a introduit dans le fichier externe
C     tout ce dont on a besoin.
C     ------------------------------------------------------------------
C
      IF ((REP.EQ.'N').OR.(REP.EQ.'n')) THEN
         GOTO 1
      ENDIF

      CALL TITRE

      CALL INTROD(NBMAX,DIMMAX,DIMH,DIMS,NBCONT,BETA,TOLER,PRECIS,
```

```
     $            LONGIN,BORNIN,PASCON,ITERA,BU,C,XLB,XUB,A,XSOL,
     $            MAXITE,MAXCON,OK)
      OK = 0
      IF (OK.EQ.1) THEN
         GOTO 1
      ENDIF
      NBITER=1
      NBCOUF=0
      NBCOUG=0
C
C     ------------------------------------------------------------------
C     Les deux conditions suivantes testent si premierement nous avons
C     atteint le nombre maximum de contraintes que le programme pouvait gerer
C     et deuxiemement si nous avons atteint le nombre maximum d'iterations que
C     le programme pouvait effectuer.
C     ------------------------------------------------------------------
C
 2    IF (NBCONT.LE.MAXCON) THEN
         IF (NBITER.LE.MAXITE) THEN
            CALL DDLPRS(NBCONT,DIMH,A,NBMAX,BL,BU,C,IRTYPE,XLB,
     $                  XUB,OBJ,XSOL,DSOL)
C
C     ------------------------------------------------------------------
C     La boucle suivante stocke la solution du probleme de maximisation dans
C     le vecteur SOLUTI.
C     ------------------------------------------------------------------
C
            DO 20 I=1,DIMH
               SOLUTI(NBITER,I)=XSOL(I)
 20         CONTINUE
            SIGMA(NBCONT)=XSOL(DIMH)
            ITERA(NBCONT)=NBITER
C
C     ------------------------------------------------------------------
C     La condition suivante teste si la solution du probleme de maximisation
C     a la precision demandee.
C     ------------------------------------------------------------------
C
            IF (SIGMA(NBCONT).GT.PRECIS) THEN
               CALL ADMISS(NBMAX,DIMMAX,DIMH,DIMS,NBCONT,TOLER,VALEUR,
     $                     REALIS,XSOL,LONGIN,BORNIN,SIGMA,A,BU,ITERA,
     $                     PASCON,NBITER)
               CALL SUPPR2(NBMAX,DIMMAX,DIMH,NBITER,XSOL,A,BU,
     $                     PASCON,ITERA,SIGMA,BETA,NBCONT,PRECIS)
C
C     ------------------------------------------------------------------
C     La condition suivante teste si la contrainte f(x)<=0 n'est pas verifiee.
C     Si oui, on ajoute une coupe suivant la fonction f. Sinon, on teste si la
C     la contrainte g(x,t)<=0 n'est pas verifiee. Si oui, on ajoute une coupe
C     suivant la fonction g, sinon on ajoute la coupe X(N+1)+sigma<=Xk(n+1).
C     ------------------------------------------------------------------
C
               IF (REALIS(1).EQ.1) THEN
                  CALL AJOUT2(NBMAX,DIMMAX,DIMH,XSOL,A,BU,PASCON,
     $                        NBCONT,NBCOUF)
               ELSE
                  IF (REALIS(2).EQ.1) THEN
                     CALL AJOUT3(NBMAX,DIMMAX,DIMS,DIMH,VALEUR,XSOL,A,
     $                           BU,PASCON,NBCONT,NBCOUG)
                  ELSE
                     CALL AJOUT1(NBMAX,DIMMAX,DIMH,XSOL,A,BU,
     $                           PASCON,NBCONT)
                  ENDIF
               ENDIF
               NBITER=NBITER+1
               GOTO 2
            ELSE
               WRITE(*,1400)
```

```
 1400          FORMAT(X,'La solution a la precision demandee.',/,
     $X,'Voulez-vous afficher a l''ecran la solution obtenue ? ',
     $X,'(O ou o =oui): ')
               READ(*,1500) REP
 1500          FORMAT(A1)
               WRITE(*,*)
C
C     ------------------------------------------------------------------
C     La condition suivante teste si on veut afficher les resultats a l'ecran.
C     ------------------------------------------------------------------
C
               IF ((REP.EQ.'O').OR.(REP.EQ.'o')) THEN
                  CALL AFFIC2(NBMAX,DIMMAX,DIMH,NBITER,NBCONT,SOLUTI,
     $                        XSOL,NBCOUF,NBCOUG)
               ENDIF
            ENDIF
         ELSE
            WRITE(*,1600)
 1600       FORMAT(X, 'Vous n''avez pas la solution du probleme avec',/,
     $x,'la precision demandee. Le programme s''est arrete car nous',/,
     $X,'avons atteint le nombre maximum d''iterations. Si vous',/,
     $X,'voulez recommencer pour avoir une solution precise, il vous',/,
     $X,'faut modifier MAX2 dans le fichier externe DONNEES.FOR .',/,
     $X,'Voulez vous quand meme afficher les resultats a ',/,
     $X,'l''ecran ? (O ou o =oui) : ')
            READ(*,1700) REP
 1700       FORMAT(A1)
C
C     ------------------------------------------------------------------
C     La condition suivante teste si on veut afficher les resultats a l'ecran.
C     ------------------------------------------------------------------
C
            IF ((REP.EQ.'O').OR.(REP.EQ.'o')) THEN
               CALL AFFIC2(NBMAX,DIMMAX,DIMH,NBITER,NBCONT,SOLUTI,
     $                     XSOL,NBCOUF,NBCOUG)
            ENDIF
         ENDIF
       ELSE
         WRITE(*,1800)
 1800    FORMAT(X, 'Vous n''avez pas la solution du probleme avec la ',/,
     $X,'precision demandee. Le programme s''est arrete car nous avons',/,
     $X,'atteint le nombre maximum de contraintes que le programme ',/,
     $X,'gere.Si vous voulez avoir une solution precise, il vous faut',/,
     $X,'modifier MAX1 dans le fichier externe DONNEES.FOR .',/,
     $X,'Voulez-vous quand meme afficher les resultats a',/,
     $X,'l''ecran ? (O ou o =oui) : ')
         READ(*,1900) REP
 1900    FORMAT(A1)
C
C     ------------------------------------------------------------------
C     La condition suivante teste si on veut afficher les resultats a l'ecran.
C     ------------------------------------------------------------------
C
         IF ((REP.EQ.'O').OR.(REP.EQ.'o')) THEN
            CALL AFFIC2(NBMAX,DIMMAX,DIMH,NBITER,NBCONT,SOLUTI,XSOL,
     $                  NBCOUF,NBCOUG)
         ENDIF
       ENDIF
  1    END

C     ***********************************************************************
C     *                          FIN DU PROGRAMME PRINCIPAL                 *
C     ***********************************************************************
```

```
C     ************************************************************************
C     *                              ROUTINE TITRE                           *
C     ************************************************************************
C
C     ************************************************************************
C
      SUBROUTINE TITRE

      WRITE(*,2000)
 2000 FORMAT(5X,'Ce programme a pour but de resoudre des problemes',/,
     $5X,'de programmation semi-infinie convexe.',/,
     $5X,'La recherche de la solution se fera comme suit: il y aura ',/,
     $5X,'application immediate de l''algorithme de plan de coupe ',/,
     $5X,'central de K.O. KORTANEK et HOON NO qui determinera la ',/,
     $5X,'solution du probleme avec la precision demandee. Puis, ',/,
     $5X,'l''odinateur vous demandera si vous voulez afficher les',/,
     $5X,'resultats a l''ecran. En ce qui concerne la dimension ',/,
     $5X,'de H a laquelle on ajoute 2, la dimension de S,',/,
     $5X,'le nombre maximum d''iterations que l''ordinateur peut',/,
     $5X,'effectuer, le nombre maximum de contraintes que le',/,
     $5X,'programme peut gerer, le minimum et le maximum sur H de',/,
     $5X,'la fonction f, les bornes inferieure et superieure des',/,
     $5X,'variables de H, la borne superieure de la contrainte ',/,
     $5X,'generale du probleme lineaire initial SD0, la borne ',/,
     $5X,'inferieure et la longueur de chaque intervalle de S,',/,
     $5X,'la precision demandee pour la solution, la tolerance ',/,
     $5X,'a la frontiere de S, la valeur de BETA entre 0 et 1,',/,
     $5X,'les expressions des fonctions f et g ainsi que le ',/,
     $5X,'gradient de ces 2 fonctions, ils doivent se trouver ',/,
     $5X,'dans le fichier externe DONNEES.FOR.',/,
     $5X,'Taper RETURN pour continuer ...')
      READ(*,*)
      WRITE(*,*)
 2200 FORMAT(A1)
      END

C     ************************************************************************
C     *                              ROUTINE INTROD                          *
C     ************************************************************************
C
C     ************************************************************************
C     * I    : variable entiere de comptage.                                 *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * MXMF est une fonction double precision contenant le maximum de la    *
C     * fonction f sur H et qui est presente dans le fichier externe         *
C     * DONNEES.FOR .                                                        *
C     * MNMF est une fonction double precision contenant le minimum de la    *
C     * fonction f sur H et qui est presente dans le fichier externe         *
C     * DONNEES.FOR .                                                        *
C     * MAX1,MAX2,DIMEN1,DIMEN2 sont des fonctions entieres contenues dans le*
C     * fichier externe DONNEES.FOR qui contiennent respectivement le nombre *
C     * nombre maximum de contraintes, le nombre maximum d'iterations, la    *
C     * dimension de H a laquelle on ajoute 2 et finalement la dimension de S.*
C     * CONST est une routine qui contient la valeur des 3 variables BETA,   *
C     * TOLER et PRECIS.                                                     *
C     * BORLON est une routine qui contient la borne inferieur et la longueur*
C     * de chaque intervalle de S.                                           *
C     * BORNES est une routine qui contient les bornes inferieure et         *
C     * superieure de chaque variable de H et la borne superieure de la      *
C     * premiere contrainte generale du probleme de maximisation du PAS 0 de *
C     * l'algorithme.                                                        *
C     ************************************************************************
C
      SUBROUTINE INTROD(NBMAX,DIMMAX,DIMH,DIMS,NBCONT,BETA,TOLER,PRECIS,
     $                  LONGIN,BORNIN,PASCON,ITERA,BU,C,XLB,XUB,A,XSOL,
```

```
     $                    MAXITE,MAXCON,OK)
          INTEGER NBMAX,DIMMAX,DIMH,DIMS,I,MAXCON,MAXITE
          INTEGER NBCONT,PASCON(NBMAX)
          INTEGER ITERA(NBMAX),OK
          REAL BETA
          DOUBLE PRECISION BORNIN(DIMMAX),LONGIN(DIMMAX)
          DOUBLE PRECISION PRECIS,TOLER,BU(NBMAX),C(DIMMAX),XLB(DIMMAX)
          DOUBLE PRECISION XUB(DIMMAX),A(NBMAX,DIMMAX),XSOL(DIMMAX)

          INTEGER MAX1,MAX2,DIMEN1,DIMEN2
          DOUBLE PRECISION MXMF,MNMF
          EXTERNAL MXMF,MNMF,MAX1,MAX2,DIMEN1,DIMEN2,CONST,BORLON,BORNES

          MAXCON = MAX1()
C
C         ------------------------------------------------------------------
C         La condition suivante teste si le nombre de contraintes que nous avons
C         introduit dans le fichier externe ne depasse pas le nombre maximum de
C         contraintes que le programme peut gerer.
C         ------------------------------------------------------------------
C
          IF (MAXCON.GT.NBMAX) THEN
             WRITE(*,1000)
 1000        FORMAT(X,'Le nombre maximum de contraintes que le programme',/,
         $X,'doit gerer et que vous avez mis dans le fichier externe',/,
         $X,'DONNEES.FOR est trop grand. Vous devez le modifier si vous',/,
         $X,'si vous desirez utiliser ce programme.')
             OK=1
             GOTO 3
          ENDIF

          MAXITE = MAX2()
C
C         ------------------------------------------------------------------
C         La condition suivante teste si le nombre maximum d'iterations que nous
C         avons introduit dans le fichier externe ne depasse pas le nombre
C         maximum d'iterations que le programme peut effectuer.
C         ------------------------------------------------------------------
C
          IF (MAXITE.GT.NBMAX) THEN
             WRITE(*,1100)
 1100        FORMAT(X,'Le nombre maximum d''iterations que le programme',/,
         $X,'peut effectuer et que vous avez mis dans le fichier externe',/,
         $X,'DONNEES.FOR est trop grand. Vous devez le modifier si vous',/,
         $X,'desirez utiliser ce programme.')
             OK=1
             GOTO 3
          ENDIF

          DIMH = DIMEN1()
C
C         ------------------------------------------------------------------
C         La condition suivante teste si la dimension de H que nous avons
C         introduite dans le fichier externe ne depasse pas la dimension maximale
C         donnee dans le programme.
C         ------------------------------------------------------------------
C
          IF (DIMH.GT.DIMMAX) THEN
             WRITE(*,1200)
 1200        FORMAT(X,'La dimension de l''espace H que vous avez',/,
         $X,'introduite dans le fichier externe DONNEES.FOR est trop',/,
         $X,'grande. Vous devez modifier cette donnee si vous voulez',/,
         $X,'utiliser ce programme.')
             OK=1
             GOTO 3
          ENDIF

          DIMS = DIMEN2()
```

```
C
C     ------------------------------------------------------------------
C     La condition suivante teste si la dimension de S que nous avons
C     introduite dans le fichier externe ne depasse pas la dimension maximale
C     donnee dans le programme.
C     ------------------------------------------------------------------
C
      IF (DIMS.GT.DIMMAX) THEN
         WRITE(*,1300)
 1300    FORMAT(X,'La dimension de l''espace S que vous avez mise',/,
     $X,'dans le fichier externe DONNEES.FOR est trop grande.',/,
     $X,'Vous devez modifier cette donnee si vous voulez utiliser',/,
     $X,'ce programme.')
         OK=1
         GOTO 3
      ENDIF

      CALL CONST(BETA,PRECIS,TOLER)
C
C     ------------------------------------------------------------------
C     La condition suivante teste si la valeur de BETA qui a ete introduite est
C     bien comprise strictement entre 0 et 1.
C     ------------------------------------------------------------------
C
      IF ((BETA.LE.0).OR.(BETA.GE.1)) THEN
         WRITE(*,2100)
 2100    FORMAT(X,'Vous devez modifier la valeur de BETA dans le ',/,
     $X,'fichier externe car sa valeur n''est pas strictement ',/,
     $X,'comprise entre 0 et 1.')
         OK=1
         GOTO 3
      ENDIF
      CALL BORLON(DIMMAX,LONGIN,BORNIN)
      NBCONT=1
      PASCON(1)=1
      CALL BORNES(DIMMAX,NBMAX,BU,XUB,XLB)
C
C     ------------------------------------------------------------------
C     La boucle suivante place dans A les coefficients des variables
C     X(1)...X(n)de la contrainte generale du probleme de depart.
C     ------------------------------------------------------------------
C
      DO 40 I=1,DIMH-2
         A(1,I)=0.D01
 40   CONTINUE
      XLB(DIMH-1)=MNMF()
      XLB(DIMH)=0.
      XUB(DIMH-1)=MXMF()
      XUB(DIMH)=-1.0E30
      A(1,DIMH-1)=0.1D01
      A(1,DIMH)=0.1D01
C
C     ------------------------------------------------------------------
C     La boucle suivante introduit les coefficients des variables X(1)...X(n+1)
C     de la fonction objectif du probleme de maximisation.
C     ------------------------------------------------------------------
C
      DO 50 I=1,DIMH-1
         C(I)=0.D01
 50   CONTINUE
      C(DIMH)=-0.1D01
 3    END
```

```
C     ************************************************************************
C     *                              ROUTINE ADMISS                          *
C     ************************************************************************
C
C     ************************************************************************
C     * VAL : variable double precision contenant la valeur de la fonction f  *
C     *       en XSOL.                                                       *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * FONF est une fonction double precision contenant l'expression de f et *
C     * dont l'expression se trouve dans le fichier externe DONNEES.FOR .     *
C     ************************************************************************
C
      SUBROUTINE ADMISS(NBMAX,DIMMAX,DIMH,DIMS,NBCONT,TOLER,VALEUR,
     $                  REALIS,XSOL,LONGIN,BORNIN,SIGMA,A,BU,ITERA,
     $                  PASCON,NBITER)
        INTEGER NBMAX,DIMMAX,DIMH,DIMS,NBITER
        INTEGER NBCONT,PASCON(NBMAX)
        INTEGER ITERA(NBMAX),REALIS(2)
        DOUBLE PRECISION BORNIN(DIMMAX),LONGIN(DIMMAX)
        DOUBLE PRECISION TOLER,VALEUR(DIMMAX),XSOL(DIMMAX),SIGMA(NBMAX)
        DOUBLE PRECISION A(NBMAX,DIMMAX),BU(NBMAX),VAL

        DOUBLE PRECISION FONF
        EXTERNAL FONF

      VAL=FONF(XSOL,DIMMAX)
C
C     ------------------------------------------------------------------
C     La condition suivante teste si la fonction f a une valeur positive au
C     point XSOL donne par resolution du probleme de maximisation.
C     ------------------------------------------------------------------
C
      IF (VAL.GT.0) THEN
         REALIS(1)=1
      ELSE
         REALIS(1)=0
      ENDIF
      CALL TEST(DIMMAX,DIMH,DIMS,TOLER,VALEUR,REALIS,LONGIN,BORNIN,
     $          XSOL)
C
C     ------------------------------------------------------------------
C     La condition suivante teste si la solution XSOL donnee par resolution
C     du probleme de maximisation est admissible i.e. si f(XSOL) est negative
C     et si g(XSOL,t) est negative pour tout t dans S.
C     ------------------------------------------------------------------
C
      IF (REALIS(1).EQ.0) THEN
         IF(REALIS(2).EQ.0) THEN
            CALL SUPPR1(NBMAX,DIMMAX,DIMH,NBITER,A,BU,ITERA,PASCON,
     $                  NBCONT,SIGMA)
         ENDIF
      ENDIF
      END


C     ************************************************************************
C     *                              ROUTINE SUPPR2                          *
C     ************************************************************************
C
C     ************************************************************************
C     * I,J,K  : variables entieres de comptage.                             *
C     * VAL    : variable double precision contenant le produit BETA*SIGMA(I).*
C     * ACTIVE : variable double precision contenant la valeur de la         *
C     *          contrainte generale I ou les variables X(1)...X(n+1) et SIGMA*
C     *          ont ete remplacees par XSOL(1)...XSOL(DIMH).                *
C     ************************************************************************
```

```
C
      SUBROUTINE SUPPR2(NBMAX,DIMMAX,DIMH,NBITER,XSOL,A,BU,PASCON,
     $                  ITERA,SIGMA,BETA,NBCONT,PRECIS)
        INTEGER NBMAX,DIMMAX,DIMH,I,J,K,NBCONT
        INTEGER NBITER,PASCON(NBMAX),ITERA(NBMAX)
        DOUBLE PRECISION XSOL(DIMMAX),A(NBMAX,DIMMAX),BU(NBMAX)
        DOUBLE PRECISION SIGMA(NBMAX),VAL,ACTIVE,BETA,PRECIS

      ACTIVE=0.
      I=1
C
C     ------------------------------------------------------------------
C     Les trois conditions suivantes testent respectivement si la coupe a ete
C     generee a une iteration precedente, si la coupe a ete generee par les
C     PAS 3 (ii) ou 3 (iii) de l'algorithme et si elle verifie que
C     BETA*SIGMA(I)>=SIGMA(K) ou K est l'iteration a laquelle on est.
C     ------------------------------------------------------------------
C
 4    IF (ITERA(I).LT.NBITER) THEN
         IF (PASCON(I).EQ.2) THEN
            VAL=BETA*SIGMA(I)
            IF (VAL.GE.SIGMA(NBCONT)) THEN
C
C     ---------------------------------------------------------------
C     La boucle suivante calcule le membre de gauche de la contrainte
C     generale I ou chaque X(I) a ete remplace par XSOL(I).
C     ---------------------------------------------------------------
C
               DO 60 J=1,DIMH
                  ACTIVE=ACTIVE+A(I,J)*XSOL(J)
 60            CONTINUE
               ACTIVE=ACTIVE-BU(I)
C
C     ------------------------------------------------------------------
C     La condition suivante teste si la contrainte est active au point XSOL
C     donne par la resolution du probleme de maximisation.
C     ------------------------------------------------------------------
C
               IF ((ACTIVE.GE.(0.01)).OR.(ACTIVE.LE.(-0.01))) THEN
C
C     ------------------------------------------------------------------
C     La boucle suivante supprime tous les renseignements concernant la coupe
C     qui a ete faite a l'iteration I.
C     ------------------------------------------------------------------
C
                  DO 70 J=I,NBCONT-1
                     ITERA(J)=ITERA(J+1)
                     PASCON(J)=PASCON(J+1)
                     BU(J)=BU(J+1)
                     SIGMA(J)=SIGMA(J+1)
                     DO 80 K=1,DIMH
                        A(J,K)=A(J+1,K)
 80                  CONTINUE
 70               CONTINUE
                  NBCONT=NBCONT-1
               ELSE
                  I=I+1
                  GOTO 4
               ENDIF
            ELSE
               I=I+1
               GOTO 4
            ENDIF
         ELSE
            I=I+1
            GOTO 4
         ENDIF
      ENDIF
```

```
      END

C     ************************************************************************
C     *                              ROUTINE AJOUT2                          *
C     ************************************************************************
C
C     ************************************************************************
C     * DIM   : constante entiere contenant la dimension maximale du vecteur  *
C     *         GRADF.                                                         *
C     * I     : variable entiere de comptage.                                  *
C     * PROD  : variable double precision contenant le produit du gradient de *
C     *         la fonction f en XSOL avec la solution XSOL.                   *
C     * VAL   : variable double precision contenant la valeur de la fonction  *
C     *         f en XSOL.                                                     *
C     * NORME : variable double precision contenant la norme du gradient de la*
C     *         fonction f.                                                    *
C     * GRADF : vecteur double precision contenant le gradient de la fonction *
C     *         en XSOL et de dimension maximale DIM.                          *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * FONF est une fonction double precision contenant l'expression de f et *
C     * dont l'expression se trouve dans le fichier externe DONNEES.FOR .      *
C     * GRF est une subroutine contenant l'expression du gradient de f et qui *
C     * est presente dans le fichier externe DONNEES.FOR .                     *
C     ************************************************************************
C
      SUBROUTINE AJOUT2(NBMAX,DIMMAX,DIMH,XSOL,A,BU,PASCON,NBCONT,
     $                  NBCOUF)
        INTEGER DIM
        PARAMETER (DIM=10)
        INTEGER NBMAX,DIMMAX,DIMH,I,PASCON(NBMAX),NBCONT,NBCOUF
        DOUBLE PRECISION XSOL(DIMMAX),A(NBMAX,DIMMAX),BU(NBMAX)
        DOUBLE PRECISION PROD,VAL,NORME,GRADF(DIM)

        DOUBLE PRECISION FONF
        EXTERNAL FONF,GRF

      PROD=0.
      NORME=0.
      NBCONT=NBCONT+1
      PASCON(NBCONT)=2
      CALL GRF(XSOL,GRADF,DIMMAX,DIM)
C
C     ------------------------------------------------------------------------
C     La boucle suivante place les renseignements de la coupe faites suivant la
C     fonction f dans A.
C     ------------------------------------------------------------------------
C
      DO 90 I=1,DIMH-1
         A(NBCONT,I)=GRADF(I)
         NORME=NORME+GRADF(I)**2
         PROD=PROD+GRADF(I)*XSOL(I)
 90   CONTINUE
      NORME=DSQRT(NORME)
      A(NBCONT,DIMH)=NORME
      VAL=FONF(XSOL,DIMMAX)
      BU(NBCONT)=PROD-VAL
      NBCOUF=NBCOUF+1
      END


C     ************************************************************************
C     *                              ROUTINE AJOUT3                          *
C     ************************************************************************
C
C     ************************************************************************
```

```
C     * DIM   : constante entiere contenant la dimension maximale du vecteur  *
C     *         GRADG.                                                         *
C     * I     : variable entiere de comptage.                                  *
C     * PROD  : variable double precision contenant le produit du gradient de *
C     *         la fonction g en XSOL et VALEUR avec la solution XSOL.         *
C     * VAL   : variable double precision contenant la valeur de la fonction g*
C     *         en XSOL et VALEUR.                                             *
C     * NORME : variable double precision contenant la norme du gradient de g *
C     *         au point XSOL et en VALEUR.                                    *
C     * GRADG : vecteur double precision contenant le gradient de la fonction *
C     *         g en XSOL et VALEUR et de dimension maximale DIM.              *
C     ***************************************************************************
C
C     ***************************************************************************
C     * FONG est une fonction double precision contenant l'expression de g et *
C     * dont l'expression est donnee dans le fichier externe DONNEES.FOR .     *
C     * GRG est une routine contenant l'expression du gradient de g et est     *
C     * present dans le fichier externe DONNEES.FOR .                          *
C     ***************************************************************************
C
      SUBROUTINE AJOUT3(NBMAX,DIMMAX,DIMS,DIMH,VALEUR,XSOL,A,BU,
     $                  PASCON,NBCONT,NBCOUG)
        INTEGER DIM
        PARAMETER (DIM=10)
        INTEGER NBMAX,DIMMAX,DIMS,DIMH,I,PASCON(NBMAX),NBCONT,NBCOUG
        DOUBLE PRECISION VALEUR(DIMMAX),XSOL(DIMMAX),A(NBMAX,DIMMAX)
        DOUBLE PRECISION BU(NBMAX),PROD,VAL,NORME,GRADG(DIM)

        DOUBLE PRECISION FONG
        EXTERNAL FONG,GRG

      PROD=0.
      NORME=0.
      NBCONT=NBCONT+1
      PASCON(NBCONT)=2
      CALL GRG(XSOL,DIMMAX,GRADG,VALEUR,DIM)
C
C     ---------------------------------------------------------------------
C     La boucle suivante place les renseignements de la coupe faite suivant
C     la fonction g dans A.
C     ---------------------------------------------------------------------
C
      DO 100 I=1,DIMH-1
         A(NBCONT,I)=GRADG(I)
         NORME=NORME+GRADG(I)**2
         PROD=PROD+GRADG(I)*XSOL(I)
  100 CONTINUE
      NORME=DSQRT(NORME)
      A(NBCONT,DIMH)=NORME
      VAL=FONG(XSOL,DIMMAX,VALEUR)
      BU(NBCONT)=PROD-VAL
      NBCOUG=NBCOUG+1
      END


C     ***************************************************************************
C     *                              ROUTINE AJOUT1                             *
C     ***************************************************************************
C
C     ***************************************************************************
C     * I : variable entiere de comptage.                                       *
C     ***************************************************************************
C
      SUBROUTINE AJOUT1(NBMAX,DIMMAX,DIMH,XSOL,A,BU,PASCON,NBCONT)
        INTEGER NBMAX,DIMMAX,DIMH,I,PASCON(NBMAX),NBCONT
        DOUBLE PRECISION XSOL(DIMMAX),A(NBMAX,DIMMAX),BU(NBMAX)

      NBCONT=NBCONT+1
```

```
      PASCON(NBCONT)=1
      DO 110 I=1,DIMH-2
         A(NBCONT,I)=0.
  110 CONTINUE
      A(NBCONT,DIMH-1)=1.
      A(NBCONT,DIMH)=1.
      BU(NBCONT)=XSOL(DIMH-1)
      END



C     ************************************************************************
C     *                              ROUTINE AFFIC2                          *
C     ************************************************************************
C
C     ************************************************************************
C     * I    : variable entiere de comptage.                                 *
C     * VAL  : variable double precision contenant la valeur de la fonction f *
C     *        au point optimal XSOL.                                        *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * FONF est une fonction double precision contenant l'expression de f et *
C     * dont l'expression se trouve dans le fichier externe DONNEES.FOR .    *
C     ************************************************************************
C
      SUBROUTINE AFFIC2(NBMAX,DIMMAX,DIMH,NBITER,NBCONT,SOLUTI,XSOL,
     $                  NBCOUF,NBCOUG)
        INTEGER NBMAX,DIMMAX,DIMH
        INTEGER NBITER,NBCONT,I,NBCOUF,NBCOUG
        DOUBLE PRECISION SOLUTI(NBMAX,DIMMAX)
        DOUBLE PRECISION VAL,XSOL(DIMMAX)

        DOUBLE PRECISION FONF
        EXTERNAL FONF

      WRITE(*,3200) NBITER
 3200 FORMAT(X,'Le nombre d''iterations effectuees est : ',I3)
      WRITE(*,3300)
 3300 FORMAT(X,'Voici la solution optimale du probleme: ')
C
C     ------------------------------------------------------------------
C     La boucle suivante affiche a l'ecran la solution optimale recherchee.
C     ------------------------------------------------------------------
C
      DO 140 I=1,DIMH-2
         WRITE(*,3400) I,SOLUTI(NBITER,I)
 3400    FORMAT(X,'X(',I2,')=',E15.8,/)
  140 CONTINUE
      VAL= FONF(XSOL,DIMMAX)+XSOL(DIMH-1)
      WRITE(*,3500)
 3500 FORMAT(X,'Voici la valeur de la fonction f au point optimal: ')
      WRITE(*,3600) VAL
 3600 FORMAT(X,'F(XSOL)= ',E15.8)
      WRITE(*,3700) NBCOUF,NBCOUG
 3700 FORMAT(X,'Le nombre de contraintes suivant la fonction f est: ',
     $I3,/,
     $X,'Le nombre de contraintes suivant la fonction g est: ',I3)
      END



C     ************************************************************************
C     *                              ROUTINE TEST                            *
C     ************************************************************************
C
C     ************************************************************************
C     * PROD,I,J : variables double precision servant a discretiser S.       *
```

```
C     ************************************************************************
C
      SUBROUTINE TEST(DIMMAX,DIMH,DIMS,TOLER,VALEUR,REALIS,LONGIN,
     $                BORNIN,XSOL)
         INTEGER DIMMAX,DIMH,DIMS,REALIS(2)
         DOUBLE PRECISION BORNIN(DIMMAX),LONGIN(DIMMAX)
         DOUBLE PRECISION TOLER,XSOL(DIMMAX),VALEUR(DIMMAX),PROD
         DOUBLE PRECISION I,J

C
C     ------------------------------------------------------
C     La condition suivante teste si la dimension de S est 1.
C     ------------------------------------------------------
C
      IF (DIMS.EQ.1) THEN
         CALL DISCRE(DIMMAX,DIMH,DIMS,TOLER,VALEUR,LONGIN,BORNIN,
     $               XSOL,REALIS)
      ELSE
         I=0.
 5       J=1.
 6       IF(J.LE.(2**I)) THEN
            PROD=((2*J-1)*LONGIN(2))*(2**(-(I+1)))
C
C     ---------------------------------------------------------------------
C     La condition suivante teste si on a verifie la condition g(x,t)<=0 pour
C     toutes les valeurs de l'intervalle a une distance TOLER de la frontiere
C     de S.
C     ---------------------------------------------------------------------
C
            IF(PROD.GT.TOLER) THEN
               VALEUR(2)=BORNIN(2)+PROD
               CALL DISCRE(DIMMAX,DIMH,DIMS,TOLER,VALEUR,LONGIN,
     $                     BORNIN,XSOL,REALIS)
C
C     ---------------------------------------------------------------------
C     La condition suivante teste si on a deja trouve une valeur dans S qui
C     verifie la condition g(x,t)>0.
C     ---------------------------------------------------------------------
C
               IF (REALIS(2).EQ.0) THEN
                  J=J+1
                  GOTO 6
               ENDIF
            ELSE
               REALIS(2)=0
            ENDIF
         ELSE
            I=I+1
            GOTO 5
         ENDIF
      ENDIF
      END



C     ************************************************************************
C     *                                ROUTINE SUPPR1                        *
C     ************************************************************************
C
C     ************************************************************************
C     * I,J,K : variables entieres de comptage.                              *
C     ************************************************************************
C
      SUBROUTINE SUPPR1(NBMAX,DIMMAX,DIMH,NBITER,A,BU,ITERA,PASCON,
     $                  NBCONT,SIGMA)
         INTEGER NBMAX,DIMMAX,DIMH,NBITER,ITERA(NBMAX),PASCON(NBMAX)
         INTEGER NBCONT,I,J,K
         DOUBLE PRECISION A(NBMAX,DIMMAX),BU(NBMAX),SIGMA(NBMAX)
```

```
      I=1
C
C     ------------------------------------------------------------------
C     Les deux conditions suivantes testent respectivement si nous sommes a
C     une iteration precedente et si la coupe a ete engendree par le PAS 3 (i)
C     de l'algorithme.
C     ------------------------------------------------------------------
C
 7    IF (ITERA(I).LT.NBITER) THEN
         IF (PASCON(I).EQ.1) THEN
C
C     ------------------------------------------------------------------
C     La boucle suivante supprime tous les renseignements de la coupe faite a
C     l'iteration I.
C     ------------------------------------------------------------------
C
            DO 120 J=I,NBCONT-1
               ITERA(J)=ITERA(J+1)
               PASCON(J)=PASCON(J+1)
               BU(J)=BU(J+1)
               SIGMA(J)=SIGMA(J+1)
               DO 130 K=1,DIMH
                  A(J,K)=A(J+1,K)
130            CONTINUE
120         CONTINUE
         ELSE
            I=I+1
            GOTO 7
         ENDIF
         NBCONT=NBCONT-1
      ENDIF
      END


C     ************************************************************************
C     *                                ROUTINE DISCRE                        *
C     ************************************************************************
C
C     ************************************************************************
C     * PROD,I,J : variables double precision servant a discretiser S.       *
C     * NB       : variable double precision contenant la valeur de la       *
C     *            fonction g en XSOL et VALEUR.                             *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * FONG est une fonction double precision contenant l'expression de g et *
C     * dont l'expression a ete introduite dans le fichier externe           *
C     * DONNEES.FOR.                                                         *
C     ************************************************************************
C
      SUBROUTINE DISCRE(DIMMAX,DIMH,DIMS,TOLER,VALEUR,LONGIN,BORNIN,
     $                  XSOL,REALIS)
         INTEGER DIMMAX,DIMH,DIMS,REALIS(2)
         DOUBLE PRECISION BORNIN(DIMMAX),LONGIN(DIMMAX)
         DOUBLE PRECISION TOLER,XSOL(DIMMAX),VALEUR(DIMMAX),PROD,NB
         DOUBLE PRECISION I,J

         DOUBLE PRECISION FONG
         EXTERNAL FONG

         I=0.
 8       J=1.
 9       IF (J.LE.(2**I)) THEN
            PROD=((2*J-1)*LONGIN(1))*(2**(-(I+1)))
C
C     ------------------------------------------------------------------
C     La condition suivante teste si on a verifie la condition g(x,t)<=0 pour
```

```
C     toutes les valeurs de l'intervalle a une distance TOLER de la frontiere
C     de S.
C     ------------------------------------------------------------------
C
           IF (PROD.GT.TOLER) THEN
              VALEUR(1)=BORNIN(1)+PROD
              NB=FONG(XSOL,DIMMAX,VALEUR)
C
C     ---------------------------------------------------------------
C     La condition suivante teste si g(XSOL,VALEUR)<=0 est verifiee.
C     ---------------------------------------------------------------
C
              IF (NB.LE.(0.D01)) THEN
                 J=J+1
                 GOTO 9
              ELSE
                 REALIS(2)=1
              ENDIF
           ELSE
              REALIS(2)=0
           ENDIF
        ELSE
           I=I+1
           GOTO 8
        ENDIF
        END
```

A.4 Listing du fichier externe

```
C     ************************************************************************
C     *                              FONCTION MAX1                           *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * Cette fonction entiere contient le nombre maximum de contraintes que *
C     * le programme MEM.FOR peut gerer.                                     *
C     ************************************************************************
C
      INTEGER FUNCTION MAX1()

      MAX1=200

      END


C     ************************************************************************
C     *                              FONCTION MAX2                           *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * Cette fonction entiere contient le nombre maximum d'iterations que le *
C     * programme MEM.FOR peut effectuer.                                    *
C     ************************************************************************
C
      INTEGER FUNCTION MAX2()

      MAX2=200

      END


C     ************************************************************************
C     *                              FONCTION DIMEN1                         *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * Cette fonction entiere contient la dimension de l'espace H a laquelle *
C     * on a ajoute deux unites.                                             *
C     ************************************************************************
C
      INTEGER FUNCTION DIMEN1()

      DIMEN1=4

      END


C     ************************************************************************
C     *                              FONCTION DIMEN2                         *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * Cette fonction entiere contient la dimension de l'ensemble S.        *
C     ************************************************************************
C
      INTEGER FUNCTION DIMEN2()

      DIMEN2=1

      END
```

```
C     ******************************************************************
C     *                          FONCTION MNMF                         *
C     ******************************************************************
C     *                                                                *
C     ******************************************************************
C     * Cette fonction double precision contient le minimum sur l'espace H de *
C     * la fonction f0 du probleme de programmation semi-infinie convexe.     *
C     ******************************************************************
C
      DOUBLE PRECISION  FUNCTION MNMF()

      MNMF=1.

      END

C     ******************************************************************
C     *                          FONCTION MXMF                         *
C     ******************************************************************
C     *                                                                *
C     ******************************************************************
C     * Cette fonction double precision contient le maximum sur l'espace H de *
C     * la fonction f0 du probleme de programmation semi-infinie convexe.     *
C     ******************************************************************
C
      DOUBLE PRECISION FUNCTION MXMF()

      MXMF=9.04

      END


C     ******************************************************************
C     *                          ROUTINE CONST                         *
C     ******************************************************************
C     *                                                                *
C     ******************************************************************
C     * Cette routine contient les valeurs des trois variables suivantes: la  *
C     * precision demandee pour la solution, la tolerance a la frontiere de S *
C     * et la valeur reelle BETA strictement comprise entre 0 et 1.           *
C     ******************************************************************
C
      SUBROUTINE CONST(BETA,PRECIS,TOLER)
        REAL BETA
        DOUBLE PRECISION TOLER,PRECIS

      BETA=0.75
      PRECIS=0.1D-09
      TOLER=0.01
      END


C     ******************************************************************
C     *                          ROUTINE BORLON                        *
C     ******************************************************************
C     *                                                                *
C     ******************************************************************
C     * Cette routine contient la longueur et la borne inferieure de chaque   *
C     * intervalle de l'ensemble S.                                           *
C     ******************************************************************
C
      SUBROUTINE BORLON(DIMMAX,LONGIN,BORNIN)
        INTEGER DIMMAX
        DOUBLE PRECISION BORNIN(DIMMAX),LONGIN(DIMMAX)

      BORNIN(1)=0.
      LONGIN(1)=8.
      END
```

```
C     ************************************************************************
C     *                              ROUTINE BORNES                          *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * Cette routine contient les bornes des variables de l'ensemble H ainsi *
C     * que la borne superieure de la contrainte generale du probleme lineaire*
C     * initial SD0.                                                         *
C     ************************************************************************
C
      SUBROUTINE BORNES(DIMMAX,NBMAX,BU,XUB,XLB)
        INTEGER DIMMAX,NBMAX
        DOUBLE PRECISION BU(NBMAX),XUB(DIMMAX),XLB(DIMMAX)

      BU(1)=4.
      XLB(1)=-1.
      XUB(1)=1.
      XLB(2)=0.
      XUB(2)=0.2
      END


C     ************************************************************************
C     *                              FONCTION FONF                           *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * Cette fonction double precision contient l'expression de la fonction f*
C     ************************************************************************
C
      DOUBLE PRECISION FUNCTION FONF(X,DIMMAX)
        INTEGER DIMMAX
        DOUBLE PRECISION X(DIMMAX)
      FONF= (X(1)-2)**2 + (X(2)-0.2)**2 - X(3)
      END


C     ************************************************************************
C     *                              FONCTION FONG                           *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * Cette fonction double precision contient l'expression de la fonction g*
C     *                                                                      *
C     * NUM : variable double precision contenant l'expression du numerateur *
C     *       du coefficient de la variable X(1).                            *
C     * PI  : variable double precision contenant la valeur de la contante pi.*
C     ************************************************************************
C
      DOUBLE PRECISION FUNCTION FONG(X,DIMMAX,T)
        INTEGER DIMMAX
        DOUBLE PRECISION X(DIMMAX),T(DIMMAX),NUM,PI
        INTRINSIC SIN,ATAN
      PI=4*ATAN(1.)
      NUM= 5*SIN(PI*DSQRT(T(1)))
      FONG= (NUM/(1+T(1)**2))*(X(1)**2) - X(2)
      END
```

```
C     ************************************************************************
C     *                              ROUTINE GRF                             *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * Cette routine contient l'expression du gradient de la fonction f.    *
C     ************************************************************************
C
      SUBROUTINE GRF(X,GRADF,DIMMAX,DIM)
        INTEGER DIMMAX,DIM
        DOUBLE PRECISION X(DIMMAX),GRADF(DIM)
      GRADF(1)= 2*X(1)-4
      GRADF(2)= 2*X(2)-0.4
      GRADF(3)= -1
      END


C     ************************************************************************
C     *                              ROUTINE GRG                             *
C     ************************************************************************
C     *                                                                      *
C     ************************************************************************
C     * Cette routine contient l'expression du gradient de la fonction g.    *
C     *                                                                      *
C     * NUM : variable double precision contenant l'expression du numerateur *
C     *       du coefficient de la variable X(1) dans GRADG(1).              *
C     * PI  : variable double precision contenant la valeur de la contante pi.*
C     ************************************************************************
C
      SUBROUTINE GRG(X,DIMMAX,GRADG,T,DIM)
        INTEGER DIMMAX,DIM
        DOUBLE PRECISION X(DIMMAX),T(DIMMAX),GRADG(DIM),NUM,PI
        INTRINSIC SIN,ATAN
      PI=4*ATAN(1.)
      NUM= 5*SIN(PI*DSQRT(T(1)))
      GRADG(1)= 2*(NUM/(1+T(1)**2))*X(1)
      GRADG(2)= -1
      GRADG(3)= 0
      END
```

Conclusion

Nous avons exposé dans ce mémoire un algorithme qui permet de résoudre les problèmes de programmation semi-infinie convexe. Cet algorithme dû à Kortanek et No est basé sur la construction de plan de coupe passant par le centre sphérique du domaine contenant les solutions optimales. D'autres algorithmes existent pour résoudre de tels problèmes (voir par exemple l'article de R. Hettich et K.O. Kortanek [2]). Un travail ultérieur consisterait à comparer les méthodes avec celles développées dans ce mémoire.

La rapidité d'exécution de la méthode de Kortanek et No dépend très fort de la façon dont l'ensemble S est discrétisé. Dans ce mémoire, nous nous sommes limités à des rectangles de dimension deux. Il serait intéressant d'examiner des ensembles S de dimension supérieure à deux.

Enfin, le programme que nous avons implémenté pourrait être amélioré afin qu'il résolve des problèmes provenant de la théorie des approximations complexes.

Bibliographie

[1] K. O. KORTANEK and HOON NO. A central cutting plane algorithme for convex semi-infinite programming problems. SIAM J. Optimisation, 3: 901-918, 1993.

[2] R. HETTICH and K. O. KORTANEK. Semi-infinite programming: Theory, methods and applications. SIAM REVIEW, 35: 380-429, 1993.

[3] R. TICHATSCHKE and V. NEBELING. A cutting plane method for quadratic semi-infinite programming problems. Optimisation, 19: 803-817, 1988.

[4] Y. TANAKA, M. TUKUSHIMA and T. IBARAKI. A comparative study of several semi-infinite nonlinear programming algorithms. European J.Oper.Res., 36: 92-100, 1988.

[5] E. POLAK and L. HE. A unified steerable Phase I- Phase II method of feasible directions for semi-infinite optimization. Dept. electrical engineering and computer sciences, 1990.

[6] MATH/LIBRARY IMSL. Fortran subroutines for mathematical applications. 1987.

[7] M. LEDOUX. Les centres analytique et elliptique d'un polytope et leurs applications aux problèmes d'optimisation convexe. FUNDP, 1994.