MASTER EN SCIENCES MATHÉMATIQUES

Une méthode de points intérieurs non admissible de type primal-dual pour résoudre un problème de programmation convexe différentiable

Hennuy, Anne

Award date:
1994

Facultés Universitaires Notre-Dame de la Paix
Namur
Faculté des Sciences

Une méthode de points intérieurs non admissible de type primal-dual pour résoudre un problème de programmation convexe différentiable

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

Anne HENNUY

Promoteur : J.J. STRODIOT

Année académique 1993-1994

Que Monsieur Jean-Jacques STRODIOT veuille bien trouver ici l'expression de ma profonde gratitude pour son encadrement dynamique et les judicieux conseils dont il m'a fait bénéficier au cours de l'élaboration de ce mémoire.

Mes chaleureux remerciements vont aussi à Monsieur Eric CORNELIS qui n'a jamais hésité à me faire profiter de ses connaissances en LaTeX, ainsi qu'à Monsieur Hubert CLAES et Mademoiselle Laurence DUMORTIER pour leur précieux concours concernant la partie informatique.

Une méthode de points intérieurs non admissible de type primal-dual pour résoudre un problème de programmation convexe différentiable

Mémoire présenté par **Anne HENNUY** en juin 1994
Facultés Universitaires Notre-Dame de la Paix à Namur
Département des Sciences Mathématiques
Unité d'Optimisation
Promoteur : **J.J. STRODIOT**

Résumé

Dans ce mémoire, nous considérons une méthode de points intérieurs non admissible de type primal-dual introduite par Vial (1992-1993) pour résoudre un problème de programmation convexe différentiable. A chaque itération, une direction de recherche de type Newton est calculée et une recherche linéaire basée sur une fonction de mérite est effectuée. Sous des hypothèses classiques, nous établissons la convergence de l'algorithme. Nous avons aussi implémenté et testé la méthode sur quelques exemples numériques.

Abstract

In this work, we consider an infeasible primal-dual interior-point method introduced by Vial (1992-1993) for solving a smooth convex programming problem. On each iteration, a Newton-type search direction is computed and a line search based on a merit function is used. We show, under mild assumptions, that the algorithm is globally convergent. We also implemented and tested this method on a few numerical examples.

Table des matières

Introduction

En 1984, Karmarkar a introduit une nouvelle méthode permettant de résoudre des problèmes d'optimisation en programmation linéaire, la méthode des points intérieurs. Depuis, de nombreux auteurs ont étudié cette méthode dont l'idée de base peut se résumer de la façon suivante.

Il s'agit de suivre approximativement une trajectoire se trouvant à l'intérieur du domaine admissible pour arriver à la solution.

Récemment, des méthodes de points intérieurs, pour lesquelles cette trajectoire n'est plus strictement admissible, ont été mises au point. Ces dernières ont été généralisées au cas convexe différentiable.

Dans ce mémoire, nous avons étudié, en suivant les articles de Vial [5] et Anstreicher et Vial [1], une méthode de points intérieurs pour le cas convexe, différentiable et non admissible. Dans une première partie, nous présentons la convergence de la méthode. Dans une seconde, nous exposons quelques résultats numériques que nous avons obtenus après avoir implémenté la méthode.

Chapitre 1

Une méthode de points intérieurs primale-duale en programmation convexe différentiable

Dans ce premier chapitre, nous allons décrire une méthode de points intérieurs primale-duale en programmation convexe différentiable, ainsi qu'un algorithme associé à cette méthode, extension directe d'une méthode de points intérieurs primale-duale pour programmation linéaire.

1.1 Approche théorique

1.1.1 Définition du problème et équations de Karush-Kuhn-Tucker

Notre problème est de minimiser une fonction linéaire sous un ensemble de contraintes d'inégalités convexes, soit

$$\begin{cases} \min & c^T x \\ \text{s.c.} & g(x) \leq 0. \end{cases} \tag{1.1}$$

Nous supposons que chaque composante g_i de g est convexe.
Soit $s \in \mathbb{R}^m$, le vecteur des variables d'écart. Les contraintes d'inégalité de (1.1) sont donc remplacées par

$$g(x) + s = 0, \; s \geq 0.$$

Etant donné un paramètre $\mu > 0$, nous associons à (1.1) le problème barrière

$$\begin{cases} \min & c^T x - \mu \sum_{i=1}^{m} \ln(s_i) \\ \text{s.c.} & g(x) + s = 0 \\ & s > 0. \end{cases} \tag{1.2}$$

Nous supposons que la condition de Slater est vérifiée :

Hypothèse 1.1. *Il existe un $x \in \mathbb{R}^n$ tel que $g(x) < 0$.*

Nous supposons aussi

Hypothèse 1.2. *Pour tout $\theta \in \mathbb{R}$,*
l'ensemble $\{x \in \mathbb{R}^n \text{ t.q. } g(x) \leq 0 \text{ et } c^T x \leq \theta\}$ est borné.

Sous ces hypothèses, le problème (1.2) a une solution.

Introduisons deux notations souvent utilisées dans les méthodes de points intérieurs. La matrice diagonale dont les éléments diagonaux sont les composantes d'un vecteur x est noté X ou $\text{diag}(x)$ et le vecteur e est le vecteur, de dimension appropriée, pour lequel chaque composante vaut un.

Les conditions nécessaires et suffisantes d'optimalité, à savoir les équations de Karush-Kuhn-Tucker, ou équations de KKT, pour le problème (1.2) sont

$$Ys - \mu e = 0 \tag{1.3}$$

$$g(x) + s = 0 \tag{1.4}$$

$$\left(\frac{\partial g}{\partial x}\right)^T y + c = 0 \tag{1.5}$$

avec $s > 0$ et $y > 0$. Ici,

$$\frac{\partial g}{\partial x} = \left\{\frac{\partial g_i(x)}{\partial x_j}\right\}$$

représente la matrice Jacobienne de g et $y \in \mathbb{R}^m$ est le vecteur des variables duales. Pour la simplicité des notations, il est commode d'introduire le Lagrangien

$$L(y; s, x) = c^T x + y^T(g(x) + s). \tag{1.6}$$

Le système de KKT (1.3) - (1.5) peut être réécrit comme

$$Ys - \mu e = 0, \quad \frac{\partial L}{\partial x} = 0, \quad \text{et} \quad \frac{\partial L}{\partial y} = 0.$$

La notation suivante sera utilisée dans les prochains paragraphes. Soit

$$F : \mathbb{R}^m \times \mathbb{R}^m \times \mathbb{R}^n \longrightarrow \mathbb{R}^m \times \mathbb{R}^m \times \mathbb{R}^n,$$

l'application définie par

$$F(z) = \begin{pmatrix} F_c(z) \\ F_p(z) \\ F_d(z) \end{pmatrix} = \begin{pmatrix} Ys - \mu e \\ \left(\frac{\partial L}{\partial y}\right)^T \\ \left(\frac{\partial L}{\partial x}\right)^T \end{pmatrix}, \tag{1.7}$$

avec $z = (y, s, x)$. F dépend aussi du paramètre $\mu > 0$. Avec cette notation, le système de KKT s'écrit simplement $F(z) = 0$.

Remarque : $F_c = 0$ signifie que le point (y, s, x) se trouve sur la trajectoire centrale, $F_p = 0$ que (y, s, x) est primal-admissible et $F_d = 0$ que (y, s, x) est dual-admissible.

1.1.2 Trajectoire centrale

Notons $(y(\mu), s(\mu), x(\mu))$, la solution du système de KKT (1.3)-(1.5) pour chaque $\mu > 0$. La *trajectoire centrale primale-duale* est définie par

$$\{ (y(\mu), s(\mu), x(\mu)) \text{ t.q. } \mu > 0 \}.$$

Selon [2],

$$(y(\mu), s(\mu), x(\mu)) \rightarrow (y^*, s^*, x^*),$$

point solution du problème (1.1), lorsque $\mu \rightarrow 0$.

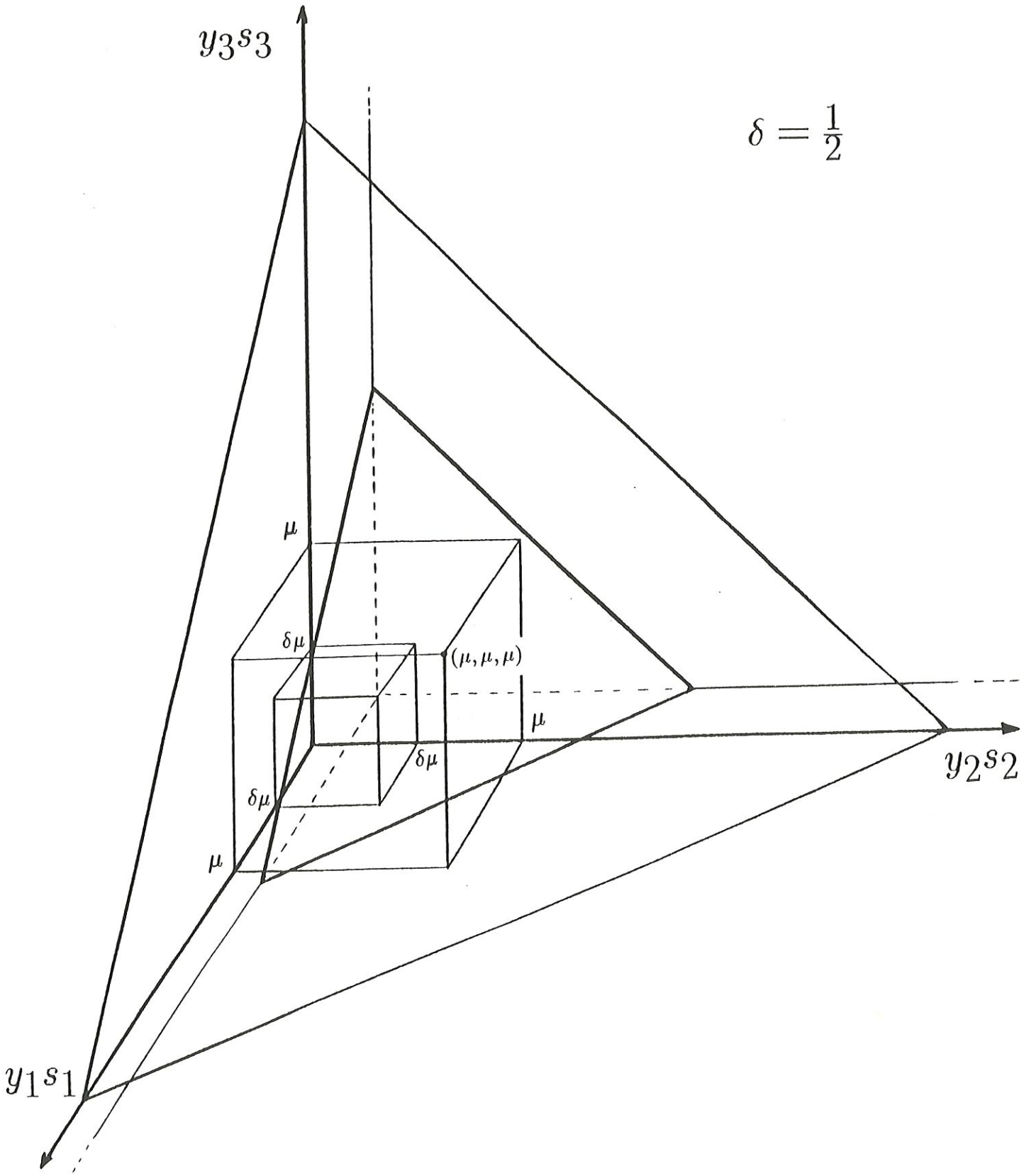
La méthode que nous allons étudier consiste à suivre la trajectoire centrale. Une première idée est de choisir un point de départ (y^0, s^0, x^0) et une valeur du paramètre $\mu > 0$, d'appliquer la méthode de Newton en partant de (y^0, s^0, x^0) pour arriver au point de la trajectoire correspondant à μ, soit $(y^1, s^1, x^1) = (y(\mu), s(\mu), x(\mu))$. Ensuite, on diminue μ et on recommence à partir de (y^1, s^1, x^1). Répétant ce procédé, on engendre une suite (y^k, s^k, x^k) qui convergera vers $z^* = (y^*, s^*, x^*)$ vérifiant $F(z^*) = 0$. Cependant, cet algorithme n'est pas implémentable car la méthode de Newton demande une infinité d'itérations pour obtenir le point de la trajectoire correspondant à μ. Une deuxième idée est alors de suivre la trajectoire "approximativement", c'est-à-dire non plus en cherchant à atteindre le point de la trajectoire centrale correspondant à μ, mais en restant dans un "voisinage" de la trajectoire centrale.

Un couple (y, s) strictement primal-dual admissible se trouve dans un *δ-voisinage de la trajectoire centrale* si

$$\min \{y_i s_i\} \geq \delta \frac{y^T s}{m},$$

avec $0 \leq \delta < 1$.

La page suivante illustre un $\frac{1}{2}$-voisinage de la trajectoire centrale où $\mu = \beta \frac{y^T s}{m}$ et $\beta = 1$.


$y_3 s_3$
$\delta = \frac{1}{2}$
μ
$\delta\mu$
(μ, μ, μ)
μ
$\delta\mu$
$y_2 s_2$
$\delta\mu$
μ
$y_1 s_1$

Une autre manière de définir un voisinage de la trajectoire centrale est possible grâce à l'introduction d'une *mesure de proximité* :

$$\delta(y, s; \mu) \;=\; \left\| \frac{Ys}{\mu} - e \right\|,$$

où (y, s) est un point strictement primal-dual admissible. Elle "mesure" la distance de (y, s) au point $(y(\mu), s(\mu))$ de la trajectoire.
Remarquons que lorsque le couple $(\bar{y}, \bar{s})$ se trouve sur la trajectoire centrale, nous avons

$$\delta(\bar{y}, \bar{s}; \mu) \;=\; 0.$$

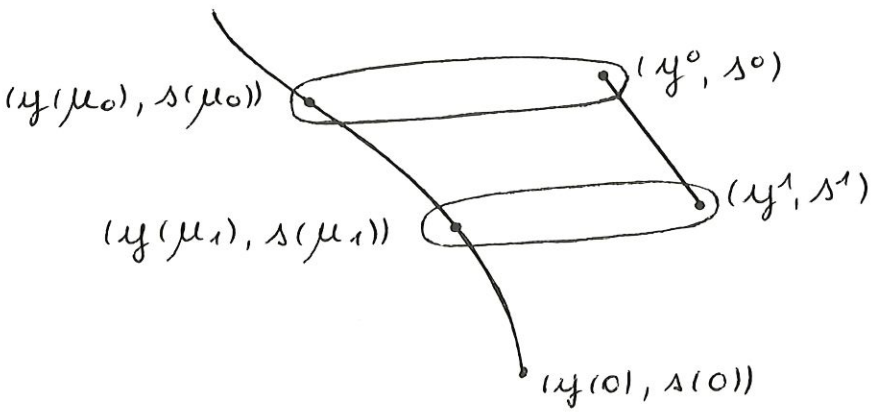
Le couple $(\bar{y}, \bar{s})$ est alors *dans un voisinage de la trajectoire centrale* si

$$\delta(\bar{y}, \bar{s}; \mu) \;\leq\; \xi \tag{1.8}$$

avec ξ fixé, pas trop grand, de sorte que le point $(\bar{y}, \bar{s})$ soit strictement primal-dual admissible.

Exposons maintenant deux stratégies pour suivre la trajectoire. Une première manière est la méthode "à pas courts" :

Le point de départ (y^0, s^0) n'est pas choisi trop loin de la trajectoire. Un pas de Newton envoie (y^0, s^0) sur (y^1, s^1) et les itérés (y^i, s^i) vérifient tous la condition (1.8). Pour cette méthode, un seul pas de Newton est effectué à chaque itération.



Algorithme de la méthode "à pas lents"

Données	Paramètres	ε	paramètre de précision
		$0 < \theta < 1$	facteur de réduction de μ
		ξ	paramètre de proximité

Entrées | (y^0, s^0), point strictement primal-dual admissible, et
$\mu_0 > 0$ tels que
$\delta(y^0, s^0; \mu_0) \leq \xi$

BEGIN

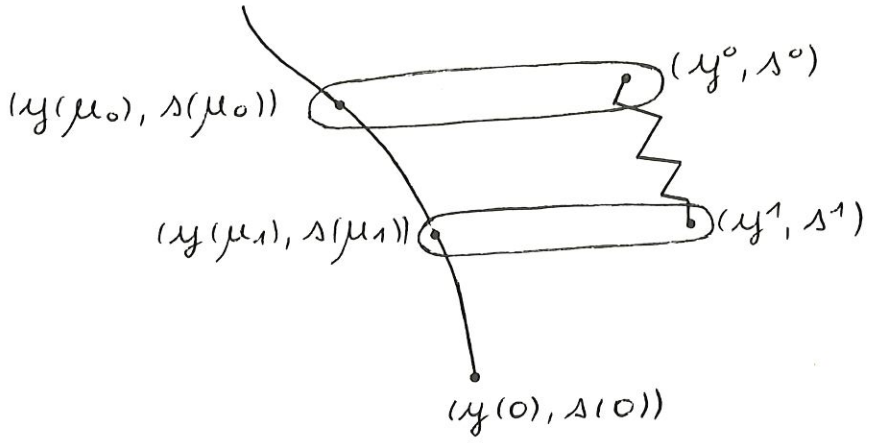
$y = y^0 \quad s = s^0 \quad x = x^0 \quad \mu = \mu_0$
WHILE $y^T s > \varepsilon$ DO

calculer dy, ds, dx
mettre à jour $y = y + dy$
$s = s + ds$
$\mu = (1 - \theta)\,\mu$

END

END.

Dans la méthode précédente, μ décroît lentement ($\mu = (1 - \theta)\,\mu$ à chaque itération). Le nombre d'itérations peut dès lors être grand. Une deuxième manière de procéder consiste à essayer de diminuer le nombre d'itérations. Pour ce faire, nous allons diminuer plus rapidement le paramètre μ avec la conséquence que, dans ce cas, l'itéré pourra être loin de $(y(\mu), s(\mu))$ et un seul pas de Newton ne sera plus suffisant. Nous aurons alors deux types d'itérations : les itérations externes (lorsque μ est changé) et les itérations internes (lorsque μ reste fixé).
De plus, il est possible qu'un pas de Newton conduise à un point non admissible. Une recherche linéaire avec une fonction de mérite, notée h, sera donc nécessaire.



Algorithme de la méthode "à pas longs"

Données	Paramètres	ε	paramètre de précision
		$0 < \theta < 1$	facteur de réduction de μ
		$\xi < \frac{1}{2}$	paramètre de proximité

Entrées	(y^0, s^0), point strictement primal-dual admissible, et $\mu_0 > 0$ tels que $\delta(y^0, s^0; \mu_0) \leq \xi$

BEGIN

$y = y^0 \quad s = s^0 \quad x = x^0 \quad \mu = \mu_0$

WHILE $y^T s > \varepsilon$ DO

$\mu = (1 - \theta)\,\mu$

WHILE $\delta(y, s; \mu) = \left\| \frac{Ys}{\mu} - e \right\| > \xi$ DO

calculer $dy,\ ds,\ dx$

trouver $\alpha > 0$ t.q. $h(y + \alpha\, dy, s + \alpha\, ds) < h(y, s)$

mettre à jour $y = y + \alpha\, dy$

$s = s + \alpha\, ds$

END

END

END.

Cette dernière méthode est numériquement meilleure que la précédente car elle est plus rapide. C'est donc en se basant sur celle-là que nous développerons la méthode du second chapitre.

1.1.3 Direction de recherche

Suivant la terminologie usuelle, le *point* $z = (y, s, x)$ est *intérieur* si $y > 0$ et $s > 0$. Nous n'exigeons pas qu'il soit primal ou dual admissible.
A un tel point, nous définissons la *direction de Newton* $dz = (dy, ds, dx)$ par

$$\frac{\partial F}{\partial z}\, dz + F = 0. \tag{1.9}$$

Remarquons que

$$\frac{\partial F}{\partial z} = \begin{pmatrix} S & Y & 0 \\ 0 & \frac{\partial^2 L}{\partial y \partial s} & \frac{\partial^2 L}{\partial y \partial x} \\ \frac{\partial^2 L}{\partial x \partial y} & 0 & \frac{\partial^2 L}{\partial x^2} \end{pmatrix}$$

avec $\frac{\partial^2 L}{\partial y \partial s} = I$, $\frac{\partial^2 L}{\partial y \partial x} = \frac{\partial g}{\partial x}$, $\frac{\partial^2 L}{\partial x \partial y} = \left(\frac{\partial g}{\partial x}\right)^T$, et $\frac{\partial^2 L}{\partial x^2} = \sum_{i=1}^{m} y_i \frac{\partial^2 g_i}{\partial x^2}$.
Etant donné que les g_i sont convexes, $\frac{\partial^2 L}{\partial x^2}$ est définie positive.
Faisons l'hypothèse supplémentaire :

Hypothèse 1.3. *Soit $y > 0$ et $s > 0$.*
La matrice $H := \frac{\partial^2 L}{\partial x^2} + \frac{\partial^2 L}{\partial x \partial y} Y S^{-1} \frac{\partial^2 L}{\partial y \partial x}$ *est définie positive.*

Remarque : On peut monter que si $\nabla^2_{x^2} L$ est définie positive ou
si $\nabla g(x)$ est de rang plein,
alors, H est définie positive.

Sous l'hypothèse 1.3., $\frac{\partial F}{\partial z}$ est régulière en tout point intérieur. Donc,

$$dz = -\left(\frac{\partial F}{\partial z}\right)^{-1} F.$$

Ecrivons explicitement et résolvons le système (1.9) en dy, ds, dx :

$$S\,dy + Y\,ds + F_c = 0 \tag{1.10}$$

$$ds + \frac{\partial^2 L}{\partial y \partial x}\,dx + F_p = 0 \tag{1.11}$$

$$\frac{\partial^2 L}{\partial x \partial y}\,dy + \frac{\partial^2 L}{\partial x^2}\,dx + F_d = 0. \tag{1.12}$$

En ce qui concerne le coefficient de ds dans l'équation (1.11), nous avons pris en compte le fait que $\frac{\partial^2 L}{\partial y \partial s} = I$. La solution est

$$\begin{aligned} dx &= H^{-1}\left(\frac{\partial^2 L}{\partial x \partial y}\,S^{-1}(F_c - Y F_p) - F_d\right) \\ ds &= -F_p - \frac{\partial^2 L}{\partial y \partial x}\,dx \\ dy &= -S^{-1}F_c - Y S^{-1} ds. \end{aligned}$$

Un pas de Newton complet part de z et arrive en $z + dz$. Il y a peu de chances qu'un tel pas conserve la propriété d'intériorité, à savoir $y + dy \not> 0$ et/ou $s + ds \not> 0$. Pour maintenir cette propriété, il est nécessaire de choisir un pas de Newton amorti : $z + \alpha\,dz$, avec $0 \leq \alpha < \alpha_{\max}$, et

$$\alpha_{\max} = \max\{\alpha \geq 0 \ \text{ t.q. } \ y + \alpha\,dy \geq 0 \ \text{ et } \ s + \alpha\,ds \geq 0\}.$$

Notre objectif étant de diminuer le saut de dualité $y^T s$ et de rendre primal-dual admissible (y, s, x), nous allons montrer que la direction de Newton est une direction de descente pour la fonction de mérite

$$f(y, s, x) = y^T s + \frac{1}{2}\,\|F_p\|^2 + \frac{1}{2}\,\|F_d\|^2 - \mu \sum_{i=1}^{m} \ln(y_i s_i).$$

Notons

$$Df(z;dz) = \frac{\partial f}{\partial z}\, dz,$$

la dérivée directionnelle de f dans la direction dz.

Théorème 1.1. *Si dz satisfait $\frac{\partial F}{\partial z}\, dz + F = 0$,*

alors, $Df(z;dz) = -\|F_p\|^2 - \|F_d\|^2 - \|v - \mu v^{-1}\|^2$,

où $v = (YS)^{\frac{1}{2}}\, e$.

Preuve :

Par définition,

$$Df(z;dz) = \frac{\partial f}{\partial y}\, dy + \frac{\partial f}{\partial s}\, ds + \frac{\partial f}{\partial x}\, dx$$

avec

$$\begin{aligned}
\frac{\partial f}{\partial y} &= (s - \mu\, y^{-1})^T + F_p^T \frac{\partial F_p}{\partial y} + F_d^T \frac{\partial F_d}{\partial y} \\
\frac{\partial f}{\partial s} &= (y - \mu\, s^{-1})^T + F_p^T \frac{\partial F_p}{\partial s} + F_d^T \frac{\partial F_d}{\partial s} \\
\frac{\partial f}{\partial x} &= F_p^T \frac{\partial F_p}{\partial x} + F_d^T \frac{\partial F_d}{\partial x}.
\end{aligned}$$

Etant donné que

$$F_p^T \left(\frac{\partial F_p}{\partial y}\, dy + \frac{\partial F_p}{\partial s}\, ds + \frac{\partial F_p}{\partial x}\, dx \right) = F_p^T \frac{\partial F_p}{\partial z}\, dz = -\|F_p\|_2^2 \qquad (1)$$

et que

$$F_d^T \left(\frac{\partial F_d}{\partial y}\, dy + \frac{\partial F_d}{\partial s}\, ds + \frac{\partial F_d}{\partial x}\, dx \right) = F_d^T \frac{\partial F_d}{\partial z}\, dz = -\|F_d\|_2^2 \qquad (2)$$

car, par hypothèse,

$$\frac{\partial F}{\partial z} dz = -F,$$

nous obtenons

$$Df(z; dz) = -\|F_p\|^2 - \|F_d\|^2 + (s - \mu\, y^{-1})^T dy + (y - \mu\, s^{-1})^T ds. \tag{3}$$

En effet,

$$\begin{aligned} Df(z; dz) &= (s - \mu\, y^{-1})^T\, dy + F_p^T \frac{\partial F_p}{\partial y}\, dy + F_d^T \frac{\partial F_d}{\partial y}\, dy \\ &\quad + (y - \mu\, s^{-1})^T\, ds + F_p^T\, \frac{\partial F_p}{\partial s}\, ds + F_d^T\, \frac{\partial F_d}{\partial s}\, ds \\ &\quad + F_p^T\, \frac{\partial Fp}{\partial x}\, dx + F_d^T\, \frac{\partial F_d}{\partial x}\, dx \end{aligned}$$

donc, grâce à (1) et (2), on obtient directement (3).
Mais nous avons que

$$(s - \mu\, y^{-1})^T\, dy + (y - \mu\, s^{-1})^T\, ds = e^T (I - \mu\, Y^{-1} S^{-1})(S\, dy + Y ds)$$

car

$$\begin{aligned} e^T (I - \mu\, Y^{-1} S^{-1})(S\, dy + Y ds) &= (e^T - \mu\, (y^{-1})^T S^{-1})(S\, dy + Y ds) \\ &= e^T S\, dy + e^T Y ds - \mu\, (y^{-1})^T dy - \mu\, (y^{-1})^T S^{-1} Y ds \\ &= s^T dy + y^T ds - \mu\, (y^{-1})^T dy - \mu\, (s^{-1})^T ds \\ &= (s - \mu\, y^{-1})^T dy + (y - \mu\, s^{-1})^T ds, \end{aligned}$$

ainsi que

$$e^T (I - \mu\, Y^{-1} S^{-1})(S\, dy + Y ds) = e^T (I - \mu\, Y^{-1} S^{-1})(-Ys + \mu e),$$

grâce à la relation (1.10) et à la définition de F_c établie en (1.7).
Or,

$$e^T(I - \mu Y^{-1}S^{-1})(-Ys + \mu e) = -e^T YS\,(I - \mu Y^{-1}S^{-1})^2\, e.$$

En effet, d'une part,

$$\begin{aligned} e^T(I - \mu Y^{-1}S^{-1})(-Ys + \mu e) &= (e^T - \mu\,(y^{-1})^T S^{-1})(-Ys + \mu e) \\ &= (-y^T s + \mu\, m + \mu\,(y^{-1})^T S^{-1} Ys - \mu^2 (y^{-1})^T S^{-1} e) \\ &= -y^T s + \mu\, m + \mu\,(s^{-1})^T s - \mu^2 (y^{-1})^T S^{-1} e \\ &= -y^T s + 2\,\mu\, m - \mu^2 \sum_{i=1}^{m} \frac{1}{y_i s_i}, \end{aligned}$$

d'autre part,

$$\begin{aligned} -e^T YS\,(I - \mu Y^{-1}S^{-1})^2\, e &= (-e^T YS + \mu\, e^T YS\, Y^{-1}S^{-1})(I - \mu Y^{-1}S^{-1})\, e \\ &= (-y^T S + \mu\, y^T S\, Y^{-1}S^{-1} + \mu\, y^T S\, Y^{-1}S^{-1} \\ &\quad -\mu^2 y^T S\, Y^{-1}S^{-1}Y^{-1}S^{-1})\, e \\ &= -y^T s + \mu\, y^T S\, Y^{-1}s^{-1} + \mu\, y^T S\, Y^{-1}s^{-1} \\ &\quad -\mu^2 y^T S\, Y^{-1}S^{-1}Y^{-1}s^{-1} \\ &= -y^T s + \mu\, y^T y^{-1} + \mu\, y^T y^{-1} - \mu^2 \sum_{i=1}^{m} \frac{1}{y_i s_i} \end{aligned}$$

$$-e^T YS\,(I - \mu Y^{-1}S^{-1})^2 e = -y^T s + 2\,\mu\, m - \mu^2 \sum_{i=1}^{m} \frac{1}{y_i s_i}. \tag{4}$$

Donc, si nous posons

$$v = (YS)^{\frac{1}{2}}\, e,$$

Il s'ensuit que

$$e^T Y S\,(I - \mu\, Y^{-1} S^{-1})^2\, e = \|v - \mu v^{-1}\|_2^2.$$

En effet,

$$\begin{aligned}
\|v - \mu v^{-1}\|_2^2 &= (v - \mu v^{-1})^T (v - \mu v^{-1}) \\
&= v^T v - \mu\, v^T v^{-1} - \mu\, (v^{-1})^T v + \mu^2 (v^{-1})^T v^{-1} \\
&= \sum_{i=1}^{m} y_i s_i - 2\,\mu\, m + \mu^2 \sum_{i=1}^{m} \frac{1}{y_i s_i}
\end{aligned}$$

et nous concluons grâce à l'égalité suivante découlant de (4)

$$e^T Y S\,(I - \mu\, Y^{-1} S^{-1})^2\, e = \sum_{i=1}^{m} y_i s_i - 2\,\mu\, m + \mu^2 \sum_{i=1}^{m} \frac{1}{y_i s_i}$$

■

1.1.4 Cas linéaire

Pour mieux comprendre l'effet d'un pas de Newton amorti, en opposition avec un pas complet, nous considérons le cas particulier d'un problème linéaire :

$$\left\{ \begin{array}{ll} \min & c^T x \\ \text{s.c.} & A\,x + s = b \\ & s \geq 0. \end{array} \right. \tag{1.13}$$

où $A \in I\!R^{m \times n}$, c et $x \in I\!R^n$ et $b \in I\!R^m$. Il est facile de vérifier que

$$\left(\frac{\partial L}{\partial y}\right)^T = A\,x + s - b, \left(\frac{\partial L}{\partial x}\right)^T = c + A^T y, \tag{1.14}$$

$$\frac{\partial^2 L}{\partial x \partial y} = A, \ \frac{\partial^2 L}{\partial y \partial s} = I, \ \frac{\partial^2 L}{\partial y \partial x} = A^T, \ \text{ et } \ \frac{\partial^2 L}{\partial x^2} = 0. \tag{1.15}$$

Supposons que l'on prenne un pas de Newton amorti. Alors, par (1.7), et finalement par

(1.9), nous avons

$$\begin{aligned} F_p(z + \alpha\, dz) &= A\, x + s - b + \alpha\,(A\, dx + ds) \\ &= F_p(z) + \alpha \left(\frac{\partial F_p}{\partial z}\right) dz \\ &= (1 - \alpha)\, F_p(z). \end{aligned}$$

De manière semblable, nous avons

$$F_d(z + \alpha\, dz) = (1 - \alpha)\, F_d(z).$$

Finalement, par (1.7),

$$F_c(z + \alpha\, dz) = (Y + \alpha\, Dy)(s + \alpha\, ds) - \mu e \tag{1.16}$$

et la relation (1.10) ainsi que la définition de F_c établie en (1.7) nous donnent

$$F_c(z + \alpha\, dz) = (1 - \alpha)(Ys - \mu e) + \alpha^2 Dy\, ds, \tag{1.17}$$

où

$$Dx = \text{ diag } (dx).$$

De par la linéarité du problème, un pas de Newton partiel réduit la non-admissibilité primale et duale F_p et F_d par un même facteur $(1 - \alpha)$. S'il est admissible, la longueur du pas $\alpha = 1$ engendre l'admissibilité primale et duale en même temps. D'où le grand intérêt de prendre la longueur du pas de Newton complet $\alpha = 1$ chaque fois que c'est possible.

Analysons le terme complémentaire F_c. Si $F_c = 0$, c'est-à-dire si $Ys = \mu e$, nous dirons que le *couple* (y, s) est *centré*. Un couple centré est un *μ-centre* s'il est primal et dual admissible, à savoir si $F_p = 0$ et $F_d = 0$. Remarquons que pour un couple centré, le saut de dualité vaut

$$y^T s = m\mu.$$

Après un pas de Newton amorti, le saut de dualité est :

$$\begin{aligned}(y+\alpha\,dy)^T(s+\alpha\,ds) &= e^T(F_c(z+\alpha\,dz)+\mu e)\\ &= (1-\alpha)(y^Ts-m\mu)+\alpha^2 dy^T ds+m\mu,\end{aligned}$$

grâce aux relations (1.16) et (1.17).
Supposons que $F_p = 0$ et $F_d = 0$. Il s'ensuit, grâce à (1.9) et (1.15), que dy et ds sont orthogonaux. Si $\mu = \beta\frac{y^Ts}{m}$,

$$(y+\alpha\,dy)^T(s+\alpha\,ds) = (1-\alpha\,(1-\beta))\,y^Ts.$$

Les cas extrêmes sont $\beta = 1$ et $\beta = 0$. Si $\beta = 1$, le saut de dualité reste inchangé. L'effet du pas de Newton est alors un centrage pur, d'où le nom *pas de centrage* lorsque $\beta = 1$. Si $\beta < 1$, le saut de dualité décroît. La diminution maximale se produit quand $\beta = 0$. Un tel pas est appelé un *pas affin*. Nous appelons *pas normal* un pas associé à une valeur intermédiaire $0 < \beta < 1$.

Dans le cas non linéaire, il n'y a pas de relation simple entre la longueur du pas et le saut de dualité après un pas de Newton amorti.

1.2 L'algorithme

Cet algorithme est basé sur l'application directe de la méthode de Newton au système d'équations de KKT (1.3) - (1.5) associé au problème barrière (1.2) et est une généralisation du cas linéaire. Le paramètre barrière est adaptable. En ce qui concerne les pas de centrage, le paramètre barrière est $\mu = \frac{y^Ts}{m}$. Pour les pas normaux, il est $\mu = \beta\frac{y^Ts}{m}$, avec $0 < \beta < 1$.

Paramètres

n = nombre de variables libres
m = nombre de contraintes

ε = critère d'arrêt (valeur défaut : 10^{-6})
β = fraction du paramètre de centrage (valeur défaut : $\frac{1}{m}$)
γ = fraction du pas maximal (valeur défaut : $1 - 10^5$)
δ = seuil de centrage (valeur défaut : $1 - \gamma$)

Initialisation

Point initial : $y^0 > 0,\ s^0 > 0$ et $x^0 \in I\!R^n$.

Etape principale

Calcul des résidus

$$F_p := g(x) + s$$
$$F_d := \left(\frac{\partial g}{\partial x}\right)^T y + c$$

Test d'arrêt

relgap $:= \frac{y^T s}{(1+|b^T x|)}$, relp $:= \frac{\|F_p\|}{1+\|x\|}$, reld $:= \frac{\|F_d\|}{1+\|y\|}$
si max { relgap, relp, reld } $< \varepsilon$, STOP.

Choix de μ (c-à-d pas de centrage ou normal)

$$\mu = \begin{cases} \beta \frac{y^T s}{m} & \text{si } \min_i(y_i s_i) > \delta \frac{y^T s}{m} \text{ et relgap} > \varepsilon, \text{(PAS NORMAL)} \\ \frac{y^T s}{m} & \text{sinon, (PAS DE CENTRAGE)} \end{cases}$$

$$F_c := Ys - \mu e$$

Direction

$$H = \sum_{i=1}^{m} y_i \frac{\partial^2 g_i}{\partial x^2} + \frac{\partial^2 L}{\partial x \partial y} Y S^{-1} \frac{\partial^2 L}{\partial y \partial x}$$
$$dx = H^{-1}\left(\frac{\partial^2 L}{\partial x \partial y} S^{-1}(F_c - YF_p) - F_d\right)$$
$$ds = -F_p - \frac{\partial^2 L}{\partial y \partial x} dx$$
$$dy = -S^{-1}F_c - YS^{-1}ds$$

Pas maximal

$$\alpha_s := \max\{\alpha \geq 0 \text{ t.q. } s + \alpha\, ds \geq 0\}$$
$$\alpha_y := \max\{\alpha \geq 0 \text{ t.q. } y + \alpha\, dy \geq 0\}$$

Calcul pour un pas de Newton admissible

$$\alpha_s := \min\{\alpha_s, \tfrac{1}{\gamma}\}$$
$$\alpha_y := \min\{\alpha_y, \tfrac{1}{\gamma}\}$$
si PAS DE CENTRAGE,
alors, $\alpha_s := \min\{\alpha_y, \alpha_s\}$
$\alpha_y := \min\{\alpha_y, \alpha_s\}$

Mise à jour des variables

$$x := x + \gamma\,\alpha_s\,dx$$
$$s := s + \gamma\,\alpha_s\,ds$$
$$y := y + \gamma\,\alpha_y\,dy$$

Fin de l'étape principale

1.2.1 Initialisation de l'algorithme

Il est difficile de concevoir un choix raisonnable de x^0 pour un problème arbitraire. Une façon de faire consiste à tirer les composantes de x^0 à partir de la loi normale $\mathcal{N}(0, 10)$.

En ce qui concerne y^0 et s^0, différentes alternatives ont été proposées. Celle que nous avons retenue est la suivante :

$$\begin{array}{ll} \text{Si} & g_i(x^0) > 0 \quad \forall i = 1, ..., m, \\ \text{alors,} & s^0 := e, \\ \text{sinon,} & s_i^0 := \max\{-g_i(x^0), 0\} - \frac{1}{2}\min\{\min_k\{g_k(x^0)\}, -1\}. \end{array}$$

Ce choix assure que $s^0 > 0$. Pour y^0, nous prenons

$$y^0 := \frac{\|s^0 + g(x^0)\|}{m}\left(S^0\right)^{-1} e.$$

Le couple initial (y^0, s^0) est centré, étant donné que les couples complémentaires (y_i, s_i) valent tous $\frac{\|s^0+g(x^0)\|}{m}$. Donc, un pas normal est choisi avec

$$\mu \;=\; \beta\frac{(y^0)^T s^0}{m} \;=\; \frac{\beta}{m}\|s^0 + g(x^0)\| \;=\; \frac{\beta}{m}\|F_p\|.$$

1.2.2 Conclusion de l'algorithme

Bien qu'aucun résultat théorique de convergence n'ait été obtenu pour cet algorithme, la méthode implémentée dans cette forme présente une performance pratique remarquable (voir [5] pour plus de détails).

Chapitre 2

Sur la convergence d'une méthode de points intérieurs primal-dual non admissible en programmation convexe

Le but de ce chapitre est d'obtenir un résultat de convergence globale pour une version de l'algorithme primal-dual non admissible en programmation convexe différentiable du premier chapitre.

2.1 Approche théorique

2.1.1 L'algorithme primal-dual non admissible

Considérons le problème de programmation non linéaire, convexe et différentiable :

$$\begin{cases} \min & c(x) \\ \text{s.c.} & g(x) \leq 0, \end{cases} \tag{2.1}$$

où $x \in I\!R^n$, $c(\cdot) : I\!R^n \to I\!R^1$, $g(\cdot) : I\!R^n \to I\!R^m$, où $c(\cdot)$ et chaque $g_i(\cdot)$ sont des fonctions convexes et de classe C^2. Ajoutant des variables d'écart $s \geq 0$, les contraintes d'inégalité de (2.1) peuvent être réécrites

$$g(x) + s = 0, \;\; s \geq 0.$$

Pour un $\mu > 0$, fixé, considérons le problème barrière logarithmique :

$$\begin{cases} \min & c(x) - \mu \sum_{i=1}^{m} \ln(s_i) \\ \text{s.c.} & g(x) + s = 0 \\ & s > 0. \end{cases} \tag{2.2}$$

Supposons, partout dans la suite :

Hypothèse 2.1. *Il existe un $\bar{x} \in \mathbb{R}^n$ tel que $g(\bar{x}) < 0$;* et

Hypothèse 2.2. *L'ensemble admissible $\{x \in \mathbb{R}^n \text{ t.q. } g(x) \leq 0\}$ est borné.*

Sous les hypothèses 2.1 et 2.2, le problème (2.2) a une solution unique pour chaque $\mu > 0$, et les conditions de Karush-Kuhn-Tucker sont nécessaires et suffisantes pour l'optimalité. Les conditions de KKT pour (2.2) peuvent s'écrire

$$Ys - \mu e = 0 \tag{2.3}$$

$$g(x) + s = 0 \tag{2.4}$$

$$\nabla g(x)^T y + \nabla c(x)^T = 0 \tag{2.5}$$

où $y > 0$ est le vecteur des multiplicateurs de Lagrange, $Y = \text{diag}(y)$, $s > 0$, $e \in \mathbb{R}^m$ est le vecteur dont chaque composante vaut un, et $\nabla g(x)$ est la matrice Jacobienne

$$(\nabla g(x))_{ij} = \left(\frac{\partial g}{\partial x}\right)_{ij} = \frac{\partial g_i(x)}{\partial x_j}.$$

Notons $(y(\mu), s(\mu), x(\mu))$, la solution de ce système de KKT pour chaque $\mu > 0$.
La *trajectoire centrale* est définie par

$$\{ (y(\mu), s(\mu), x(\mu)) \text{ t.q. } \mu > 0 \}.$$

Comme dans le chapitre 1,

$$(y(\mu), s(\mu), x(\mu)) \to (y^*, s^*, x^*),$$

point solution du problème (2.1), lorsque $\mu \to 0$.

Soit $z = (y, s, x)$, où $y > 0$, $s > 0$. Nous ne supposons *pas* que z soit primal-dual admissible, c'est-à-dire que z satisfasse (2.4) et (2.5). Soit

$$F(z) = \begin{pmatrix} F_c(z) \\ F_p(z) \\ F_d(z) \end{pmatrix} = \begin{pmatrix} Ys - \mu e \\ g(x) + s \\ \nabla g(x)^T y + \nabla c(x)^T \end{pmatrix}. \tag{2.6}$$

Une direction de recherche $dz = (dy, ds, dx)$ est alors définie par le pas de Newton pour le système $F(z) = 0$:

$$\nabla F(z)\, dz + F(z) = 0. \tag{2.7}$$

Sous des hypothèses convenables (voir chapitre 1), les équations (2.7) sont régulières, et la direction de Newton dz est bien définie. L'algorithme, grâce à un pas (amorti), amène z sur un nouveau point $z + \alpha\, dz$, où $\alpha > 0$ est tel que $y + \alpha\, dy > 0$, $s + \alpha\, ds > 0$, et tout le processus est répété.

Pour plus de facilités, nous supposerons partout dans la suite que le pas de Newton est en fait bien défini, c'est-à-dire que l'hypothèse suivante est vérifiée :

Hypothèse 2.3. *Pour tout $z = (y, s, x)$ tel que $s > 0$ et $y > 0$,*
$\nabla F(z)$ est non singulière.

Pour obtenir notre résultat de convergence globale, nous considérons l'algorithme du premier chapitre dans une forme légèrement différente. En particulier, plutôt que d'employer une valeur implicite de μ comme précédemment – à savoir $\mu = \beta \frac{y^T s}{m}$ où $0 < \beta < 1$ aussi longtemps qu'une condition de faible centrage est satisfaite, et $\mu = \frac{y^T s}{m}$ sinon, ce dernier cas correspondant à un pas de centrage pur – nous considérons ici une approche "à pas longs" où une valeur de $\mu > 0$ est fixée et des pas sont pris jusqu'à ce qu'une solution approximative du problème barrière (2.2) soit obtenue. A ce point, μ est réduit et le processus est répété.

En ce qui concerne la longueur du pas, α, elle était déterminée en utilisant une règle de "fraction fixée à la frontière" (en les variables positives s et y), permettant toutefois

d'utiliser le pas de Newton pur ($\alpha = 1$) si c'est possible. L'algorithme permettait aussi des longueurs de pas différentes pour les variables du primal (x et s) et du dual (y) pour les pas de non-centrage. En effet, dans le cas des pas de centrage, nous avions $\alpha_s = \alpha_y$, puisque tous deux valaient $\min\{\alpha_y, \alpha_s\}$. Ici, notre choix de longueur de pas, décrit dans le paragraphe suivant, est basé sur des propriétés de la direction dz définie par (2.7).

2.1.2 Bases pour la Convergence Globale

Dans le premier chapitre, le théorème 1.1. nous indiquait une relation importante entre la direction dz définie par (2.7) et la fonction de mérite

$$f(z) = f(y, s, x) = y^T s + \frac{1}{2}\|F_p\|^2 + \frac{1}{2}\|F_d\|^2 - \mu \sum_{i=1}^{m} \ln(y_i s_i).$$

En effet, nous avons montré que si

$$Df(z; dz) = \nabla f(z)\, dz$$

est la dérivée directionnelle de $f(\cdot)$ dans la direction dz, alors,

$$Df(z; dz) = -\|F_p\|^2 - \|F_d\|^2 - \|v - \mu v^{-1}\|^2, \tag{2.8}$$

où $v = (YS)^{\frac{1}{2}} e$, $S =$ diag (s), et v^{-1} est le vecteur dont les composantes sont les inverses des composantes de v. Il s'ensuit que dz est une direction de descente pour $f(\cdot)$, sauf si z est une solution des conditions de KKT (2.3) - (2.5). Il est aussi facile de montrer que $f(\cdot)$ est minorée par la valeur $m\mu[1 - \ln(\mu)]$. Une stratégie naturelle, pour une valeur fixée de $\mu > 0$, est donc de choisir la longueur de pas α grâce à une sorte de critère de "descente suffisante" pour la fonction $f(\cdot)$. Par exemple, une possibilité serait de choisir α pour qu'il satisfasse les conditions de Goldstein-Armijo :

$$\lambda_1\, \alpha\, |Df(z; dz)| \leq f(z) - f(z + \alpha\, dz) \leq \lambda_2\, \alpha\, |Df(z; dz)|, \tag{2.9}$$

où $0 < \lambda_1 < \lambda_2 < 1$. Il est bien connu qu'un $\alpha > 0$ satisfaisant (2.9) existe toujours; voir par exemple [3]. Comme alternative, α pourrait être choisi en utilisant une stratégie "backtracking" de la forme $\alpha = \theta^p\, \alpha_{\max}$, où $\alpha_{\max}$ est le plus grand α tel que

$(y + \alpha\, dy, s + \alpha\, ds) \geq 0,\ 0 < \theta < 1,$ et p est le plus petit entier strictement positif tel que

$$f(z) - f(z + \theta^p\, \alpha_{\max}\, dz) \geq \lambda_1\, \theta^p\, \alpha_{\max}\, |Df(z; dz)|, \tag{2.10}$$

où $0 < \lambda_1 < 1$. Choisir α selon les conditions (2.9) ou (2.10) est suffisant pour obtenir la convergence de l'algorithme pour un μ fixé.

2.1.3 Algorithme pour un μ fixé

Initialisation	:	$\mu > 0$ fixé choisir le point de départ x^0 poser $k = 1$
Etape 1	:	calculer dz (en résolvant (2.7))
Etape 2	:	trouver $\alpha > 0$ satisfaisant les conditions (2.9) ou (2.10)
Etape 3	:	poser $z^{k+1} = z^k + \alpha\, dz$ $k = k + 1$ aller à l'Etape 1

2.1.4 Convergence

Dans le théorème suivant, nous prouvons la convergence de l'algorithme pour un μ fixé lorsque la condition (2.9) est utilisée à l'Etape 2. L'analyse basée sur (2.10) est similaire.

Théorème 2.1. *Soit* $\{z^k\}$, *une suite de points générée par l'algorithme pour une valeur fixée de* $\mu > 0$, *où, à chaque itération, la longueur du pas,* α, *est choisie pour satisfaire les conditions (2.9).*

alors, tout point d'accumulation z^* *de la suite* $\{z^k\}$ *est solution de l'équation* $F(z) = 0$.

Preuve :

Soit $\{z^k\}_{k\in K}$, une sous-suite de $\{z^k\}_{k\in I\!N}$ ($K \subset I\!N$) telle que

$$z^k \underset{k\in K}{\rightarrow} z^*,$$

alors, $\{z^k$ t.q. $k \in K\}$ est bornée.
Nous avons

$$\exists\, M_1 > 0 \text{ t.q. } \forall i = 1, ..., m, \forall k \in K, \quad s_i^k y_i^k \geq M_1 \tag{1}$$

En effet, selon (2.9),

$$\lambda_1\, \alpha\, |Df(z^k; dz^k)| \leq f(z^k) - f(z^k + \alpha\, dz^k),$$

ce qui revient à

$$\lambda_1\, \alpha\, |Df(z^k; dz^k)| \leq f(z^k) - f(z^{k+1}). \tag{2}$$

Le membre de gauche de (2) étant positif, nous avons

$$f(z^{k+1}) \leq f(z^k) \quad \forall k \in I\!N.$$

En particulier,

$$f(z^k) \leq f(z^1) \quad \forall k \in I\!N. \tag{3}$$

Par définition de $f(\cdot)$,

$$f(z^k) = \sum_{i=1}^{m} y_i^k s_i^k + \frac{1}{2}\|F_p^k\|^2 + \frac{1}{2}\|F_d^k\|^2 - \mu \sum_{i=1}^{m} \ln(s_i^k y_i^k) \qquad \forall k \in I\!N,$$

d'où,

$$f(z^k) \geq -\mu \sum_{i=1}^{m} \ln(s_i^k y_i^k), \quad \forall k \in I\!N,$$

et grâce à (3),

$$\sum_{i=1}^{m} \ln(s_i^k y_i^k) \geq \frac{-f(z^1)}{\mu} \quad \forall k \in I\!N,$$

et finalement,

$$\prod_{i=1}^{m} s_i^k y_i^k \geq e^{\frac{-f(z^1)}{\mu}} \overset{\text{not}}{\equiv} M_2 \quad \forall k \in I\!N. \tag{4}$$

D'autre part, $\{z^k\}_{k\in K}$ est bornée, et donc,

$$\exists\, M_3 > 0 \ \text{ t.q. } \forall i = 1, ..., m, \forall k \in K, \quad s_i^k y_i^k \leq M_3. \tag{5}$$

Par l'absurde, supposons que l'on n'ait pas (1), c-à-d

$$\forall M_1 > 0, \exists\, i,\ 1 \leq i \leq m, \exists\, k \in K \ \text{ t.q.} \quad s_i^k y_i^k < M_1. \tag{6}$$

Choisissons M_1 tel que

$$M_1 < \frac{M_2}{(m-1)M_3}. \tag{7}$$

Par l'hypothèse absurde (6),

$$\forall M_1 > 0, \exists\, i,\ 1 \leq i \leq m, \exists\, \bar{k} \in K \ \text{ t.q.} \quad s_i^{\bar{k}} y_i^{\bar{k}} < M_1,$$

et pour ce $\bar{k} \in K$, nous obtenons, par (5), (6), et finalement par (7),

$$\prod_{j=1}^{m} s_j^{\bar{k}} y_j^{\bar{k}} < M_2,$$

ce qui contredit (4) puisque nous avons donc

$$\exists\, \bar{k} \in K \subset I\!N \text{ t.q. } \prod_{i=1}^{m} s_i^{\bar{k}} y_i^{\bar{k}} < M_2.$$

Il s'ensuit que la relation (1) est bien vérifiée. De (1) découlent

$$\exists\, m_{12} > 0 \text{ t.q. } \forall i = 1, ..., m, \forall k \in K, \quad s_i^k \geq m_{12}$$

et

$$\exists\, m_{22} > 0 \text{ t.q. } \forall i = 1, ..., m, \forall k \in K, \quad y_i^k \geq m_{22},$$

avec $m_{12}.m_{22} = M_1$. Et donc,

$$y^* > 0 \quad \text{et} \quad s^* > 0. \tag{8}$$

Grâce à un résultat standard pour les conditions de Goldstein-Armijo (2.9) (voir par exemple l'Annexe 1), nous avons

$$\frac{Df(z^k; dz^k)}{\|dz^k\|} \underset{k \in K}{\rightarrow} 0. \tag{9}$$

Par l'Hypothèse 2.3. et (8), nous obtenons que

$$\nabla F(z^*) \text{ est régulière.} \tag{10}$$

Nous avons que

$$\|dz^k\| \text{ est bornée pour } k \in K. \tag{11}$$

En effet, grâce à (2.7),

$$\|dz^k\| \leq \|\nabla F(z^k)^{-1}\|.\|F(z^k)\|.$$

Il suffit donc de montrer que $\nabla F(z^k)^{-1}$ et $F(z^k)$ sont bornés. Le premier résultat découle du lemme de perturbation suivant

$$\textit{Soit } S, T \in L(X) \quad \textit{avec } S \textit{ inversible.}$$

$$\textit{Si} \quad \|S - T\|.\|S^{-1}\| < 1,$$

$$\textit{alors, } T \textit{ est inversible et}$$
$$\|T^{-1}\| \leq \frac{\|S^{-1}\|}{1-\|S-T\|.\|S^{-1}\|}.$$

où $S \equiv \nabla F(z^*)$, $T \equiv \nabla F(z^k)$, $\nabla F(z^*)$ est inversible d'après (10) et $\|\nabla F(z^*) - \nabla F(z^k)\| < \frac{1}{\|\nabla F(z^*)^{-1}\|}$. Le deuxième, du fait que $F(\cdot)$ soit continue et $\{z^k\}_{k\in K}$ bornée. D'où nous obtenons bien (11). Par (9) et (11), nous avons que

$$Df(z^k; dz^k) \underset{k\in K}{\rightarrow} 0. \tag{12}$$

Alors, $F(z^*) = 0$ se déduit aisément de (2.8), (12), et du fait que $\{z^k \text{ t.q. } k \in K\}$ soit borné. ■

Dans la preuve du théorème précédent, nous avons montré que $Df(z^k; dz^k) \underset{k\in K}{\rightarrow} 0$.
Dès lors, la condition

$$|Df(z^k; dz^k)| \leq \beta\mu, \quad 0 < \beta < 1, \tag{2.11}$$

est satisfaite après un nombre fini d'itérations. Cette condition (2.11) peut servir de critère d'arrêt pour l'algorithme avec $\mu > 0$ fixé.

2.1.5 Algorithme avec μ variable

Pour obtenir une solution du problème (2.1), nous avons vu qu'il fallait faire tendre μ vers zéro. Une façon de procéder est d'abord, de fixer μ, puis appliquer l'algorithme 2.1.3. pour cette valeur de μ jusqu'à ce que la condition (2.11) soit satisfaite et ensuite, de diminuer

μ et recommencer le procédé. Les itérations avec μ constant seront appelées *itérations internes* et celles où μ est changé, *itérations externes.*

Algorithme avec μ variable :

Initialisation	:	$\mu_1 > 0$ fixé $0 < \theta < 1$ choisir le point de départ x^0 poser $j = 1$
Etape 1	:	appliquer l'algorithme 2.1.3. pour $\mu = \mu_j$ avec le test d'arrêt (2.11) noter z^j le point obtenu
Etape 2	:	poser $\mu_{j+1} = (1-\theta)\mu_j$ $j = j + 1$ aller à l'Etape 1

Remarque : La suite $\{z^j\}$ est la suite des itérés externes.

Nous ne pouvons démontrer la convergence de cet algorithme à cause d'une difficulté technique. Pour chaque valeur de μ, nous devons supposer qu'il existe une sous-suite convergente pour les itérations internes correspondantes. Une condition suffisante pour l'existence d'une telle sous-suite convergente est que la suite $\{z^k\}$ engendrée par les itérations internes soit bornée. Grâce à l'Hypothèse 2.2., il est relativement facile de prouver que les suites $\{x^k\}$ et $\{s^k\}$ sont bornées. Cependant, il ne semble pas être possible de borner la suite des variables duales $\{y^k\}$ et, par conséquent, les itérés $\{z^k\}$.

2.1.6 Un algorithme globalement convergent

Nous allons décrire une légère modification de l'algorithme précédent qui garantira que la suite générée par les itérations internes est bornée et qui, par conséquent, donnera un résultat de convergence globale.

La fonction de mérite $f(\cdot)$ est remplacée par la fonction

$$\hat{f}(z) = \hat{f}(y, s, x) = f(y, s, x) - \hat{\mu} \sum_{i=1}^{m} \ln(g_i(x) + s_i), \tag{2.12}$$

où $\hat{\mu} > 0$ et où nous supposons que $g(x) + s > 0$.

Montrons dans le Lemme suivant que la relation entre la direction dz et la fonction de mérite $f(\cdot)$ établie dans le premier chapitre s'étend d'une manière remarquablement simple à la fonction $\hat{f}(\cdot)$.

Lemme 2.2. *Soit* $\left| \begin{array}{l} \hat{f}(\cdot), \text{ comme énoncée en (2.12), et} \\ dz, \text{ donnée par } \nabla F(z)\, dz + F(z) = 0, \end{array} \right.$

alors, $D\hat{f}(z; dz) = Df(z; dz) + m\hat{\mu}$.

Preuve :

Nous avons

$$D\hat{f}(z; dz) = Df(z; dz) - \hat{\mu} \frac{\partial}{\partial s} \left(\sum_{i=1}^{m} \ln(g_i(x) + s_i) \right) ds - \hat{\mu} \frac{\partial}{\partial x} \left(\sum_{i=1}^{m} \ln(g_i(x) + s_i) \right) dx. \tag{1}$$

Or,

$$\frac{\partial}{\partial s_j} \left(\sum_{i=1}^{m} \ln(g_i(x) + s_i) \right) = \frac{1}{g_j(x) + s_j},$$

Donc,

$$\frac{\partial}{\partial s} \left(\sum_{i=1}^{m} \ln(g_i(x) + s_i) \right) = F_p^{-T}, \tag{2}$$

où F_p^{-T} est le vecteur ligne ayant pour $i^{\text{ème}}$ composante $\frac{1}{g_i(x)+s_i}$. De plus,

$$\frac{\partial}{\partial x_j} \left(\sum_{i=1}^{m} \ln(g_i(x) + s_i) \right) = \sum_{i=1}^{m} \frac{1}{g_i(x) + s_i} \frac{\partial g_i(x)}{\partial x_j},$$

ce qui donne

$$\frac{\partial}{\partial x}\left(\sum_{i=1}^{m}\ln(g_i(x)+s_i)\right) = F_p^{-T}\,\nabla g(x). \tag{3}$$

Substituant (2) et (3) dans (1), nous obtenons

$$\begin{aligned} D\hat{f}(z;dz) &= Df(z;dz) - \hat{\mu}\,F_p^{-T}ds - \hat{\mu}\,F_p^{-T}\,\nabla g(x)\,dx \\ &= Df(z;dz) - \hat{\mu}\,F_p^{-T}\,(ds - \nabla g(x)\,dx) \end{aligned}$$

Cependant, les équations correspondant à F_p dans (2.6) et (2.7) impliquent que

$$\nabla g(x)\,dx + ds + F_p = 0,$$

et donc,

$$F_p^{-T}\,(ds + \nabla g(x)\,dx) = F_p^{-T}(-F_p) = -m,$$

d'où la thèse. ∎

Considérons le choix $\hat{\mu} = \frac{\beta\mu}{m}$, où $0 < \beta < \frac{1}{2}$. Remarquons que, grâce au Lemme 2.2. et au Théorème 1.1., le cas où dz n'est *pas* une direction de descente pour $\hat{f}(\cdot)$ est équivalent à

$$|Df(z;dz)| = \|F_p\|^2 + \|F_d\|^2 + \|v - \mu v^{-1}\|^2 \leq \beta\mu, \tag{2.13}$$

ce qui correspond au critère de centrage (2.11). Si on suppose que dz est une direction de descente pour $\hat{f}$, un critère évident pour terminer le processus itératif pour un $\mu > 0$ fixé est :

$$|D\hat{f}(z;dz)| \leq \beta\mu,$$

qui est équivalent à

$$\|F_p\|^2 + \|F_d\|^2 + \|v - \mu v^{-1}\|^2 \leq 2\,\beta\mu. \tag{2.14}$$

Nous pouvons donc considérer (2.14) comme étant le critère d'arrêt pour tous les cas. L'ensemble des points vérifiant cette condition (2.14) définit un voisinage autour de la trajectoire centrale.

Notre stratégie pour les itérations internes sera maintenant de baser la longueur de pas α sur une "diminution suffisante" de la fonction $\hat{f}(\cdot)$, plutôt que $f(\cdot)$. Comme auparavant, les conditions de la forme (2.9) ou (2.10) conviennent avec $f(\cdot)$ remplacée par $\hat{f}(\cdot)$. Dans le cas de la forme (2.10), $\alpha_{\max}$ est changé pour devenir le plus grand α tel que $y + \alpha\, dy \geq 0$, $s + \alpha\, ds \geq 0$, et $g(x + \alpha\, dx) + (s + \alpha\, ds) \geq 0$. Ou bien, le $\alpha_{\max}$ "originaire" basé uniquement sur les deux premières conditions peut être utilisé, avec α réduit par les puissances de θ jusqu'à ce que la troisième condition soit également satisfaite. Le théorème suivant est notre plus important résultat, étant donné qu'il assure la fin des itérations internes lorsque la longueur du pas est basée sur une diminution de $\hat{f}(\cdot)$. Tout comme dans le paragraphe précédent, nous donnons un résultat basé sur (2.9), mais un résultat analogue est aussi valable pour (2.10).

Théorème 2.2. *Soit* $0 < \mu \leq 1$, *fixé.*

Supposons que, à chaque itération de l'algorithme, la longueur du pas, α, satisfasse le critère de diminution suffisante (2.9), en substituant $\hat{f}(\cdot)$ à $f(\cdot)$.

Alors, le critère de terminaison (2.14) sera satisfait après un nombre fini d'itérations.

Preuve :

Etant donné que le critère de terminaison (2.14) est immédiatement satisfait si la direction dz n'est pas une direction de descente pour $\hat{f}(\cdot)$, nous pouvons supposer que toutes les étapes sont des étapes de descente pour $\hat{f}(\cdot)$. Supposons pour le moment que les itérés $\{z^k\}$ sont bornés. Alors, par un raisonnement analogue à celui fait au Théorème 2.1., nous obtenons que $D\hat{f}(z^k; dz^k) \underset{k \in K}{\rightarrow} 0$ pour toute sous-suite $\{z^k\}$ convergente, ce qui implique que (2.14) sera un jour satisfaite. Il nous reste donc à prouver que les itérés $\{z^k\}_{k \in N}$ sont

bien bornés.

Etant donné que $\hat{f}(\cdot)$ décroît de manière monotone, et que $t - \mu \ln(t)$ est minorée et approche l'infini lorsque $t \to 0$ ou $t \to \infty$, nous concluons qu'il existe des constantes $0 < \gamma_1 < \gamma_2 < \infty$ telles que, pour $k \geq 0$ et $1 \leq i \leq m$,

$$\gamma_1 \leq y_i^k s_i^k \leq \gamma_2, \tag{1}$$

$$\gamma_1 \leq g_i(x^k) + s_i^k \leq \gamma_2. \tag{2}$$

En effet, nous savons que $\hat{f}(\cdot)$ décroît de manière monotone; donc,

$$\hat{f}(y^k, s^k, x^k) \leq \hat{f}(y^1, s^1, x^1) \stackrel{\text{not}}{\equiv} M_1 \quad \forall k \geq 0. \tag{3}$$

Montrons que

$$\exists \gamma_1 > 0 \text{ t.q. } \forall k \geq 0, \forall i : 1 \leq i \leq m, \quad \gamma_1 \leq y_i^k s_i^k.$$

Par l'absurde, supposons que

$$\forall \gamma_1 > 0, \exists k_1 \geq 0, \exists i_1 : 1 \leq i_1 \leq m \text{ t.q. } 0 < y_{i_1}^{k_1} s_{i_1}^{k_1} < \gamma_1. \tag{4}$$

Par définition de $\hat{f}(\cdot)$, nous avons

$$\begin{aligned} \hat{f}(y^{k_1}, s^{k_1}, x^{k_1}) &= \sum_{j=1}^{m} y_j^{k_1} s_j^{k_1} + \frac{1}{2} \|F_p^{k_1}\|^2 + \frac{1}{2} \|F_d^{k_1}\|^2 \\ &\quad -\mu \sum_{j=1}^{m} \ln(s_j^{k_1} y_j^{k_1}) - \hat{\mu} \sum_{j=1}^{m} \ln(g_j(x^{k_1}) + s_j^{k_1}), \end{aligned}$$

c'est-à-dire

$$\begin{aligned} \hat{f}(y^{k_1}, s^{k_1}, x^{k_1}) &= \sum_{j=1, j \neq i_1}^{m} y_j^{k_1} s_j^{k_1} + y_{i_1}^{k_1} s_{i_1}^{k_1} + \frac{1}{2} \|F_p^{k_1}\|^2 + \frac{1}{2} \|F_d^{k_1}\|^2 \\ &\quad -\mu \sum_{j=1, j \neq i_1}^{m} \ln(s_j^{k_1} y_j^{k_1}) - \mu \ln(s_{i_1}^{k_1} y_{i_1}^{k_1}) - \hat{\mu} \sum_{j=1}^{m} \ln(g_j(x^{k_1}) + s_j^{k_1}). \end{aligned}$$

Or, nous savons que $t - \mu \ln(t)$ est minorée; notons c_1 la constante, et alors,

$$\begin{aligned} \hat{f}(y^{k_1}, s^{k_1}, x^{k_1}) \;\; \geq \;\; & (m-1)\, c_1 + y_{i_1}^{k_1} s_{i_1}^{k_1} + \frac{1}{2} \sum_{j=1}^{m} (g_j(x^{k_1}) + s_j^{k_1})^2 \\ & - \mu \ln(s_{i_1}^{k_1} y_{i_1}^{k_1}) - \hat{\mu} \sum_{i=1}^{m} \ln(g_j(x^{k_1}) + s_j^{k_1}). \end{aligned}$$

Or, $\frac{1}{2}\,(t_i - \hat{\mu} \ln(t_i)) \geq \frac{1}{2}\, c_2$ et donc, en posant $t_i = (g_i(x^{k_1}) + s_i^{k_1})^2$, nous obtenons

$$\hat{f}(y^{k_1}, s^{k_1}, x^{k_1}) \geq (m-1)\, c_1 + \frac{1}{2} m c_2 + y_{i_1}^{k_1} s_{i_1}^{k_1} - \mu \ln(s_{i_1}^{k_1} y_{i_1}^{k_1}). \tag{5}$$

Si nous montrons que

$$\hat{f}(y^{k_1}, s^{k_1}, x^{k_1}) > M_1, \tag{6}$$

nous aurons une contradiction avec (3) et nous aurons démontré l'inégalité de gauche de (1). Pour que (6) soit vérifiée, il nous faut choisir

$$y_{i_1}^{k_1} s_{i_1}^{k_1} - \mu \ln(s_{i_1}^{k_1} y_{i_1}^{k_1}) > M_1 - (m-1)\, c_1 - \frac{1}{2}\, m\, c_2. \tag{7}$$

Or, nous savons que

$$\lim_{t \to 0} (t - \mu \ln(t)) = +\infty,$$

ce qui est équivalent à

$$\forall b \in I\!R,\ \exists\, \delta > 0 \ \text{ t.q. } \forall t > 0, \ \ 0 < t < \delta \ \Rightarrow \ t - \mu \ln(t) > b,$$

et nous avons donc, grâce à (7),

$$\exists\, \gamma_1 > 0 \ \text{ t.q. } \forall t > 0, \ \ 0 < t < \gamma_1 \ \Rightarrow \ t - \mu \ln(t) > M_1 - (m-1)\, c_1 - \frac{1}{2}\, m\, c_2. \tag{8}$$

Considérons ce $\gamma_1 > 0$ et, par l'hypothèse absurde (4) et (8), nous obtenons

$$y_{i_1}^{k_1} s_{i_1}^{k_1} - \mu \ln(y_{i_1}^{k_1} s_{i_1}^{k_1}) > M_1 - (m-1)\, c_1 - \frac{1}{2}\, m\, c_2.$$

De cette dernière inégalité, ainsi que de (5), nous déduisons (6) et l'inégalité de gauche de (1) est ainsi démontrée. Celle de droite et (2) se démontrent de manière analogue.

D'après le théorème

Soit $f : \mathbb{R}^n \to \mathbb{R}$ *convexe.*

Si $\{x \ \ t.q. \ \ f(x) \leq \alpha\}$ *est borné et non vide pour un* α,

alors, $\{x \ \ t.q. \ \ f(x) \leq \beta\}$ *est borné pour tout* β.

de [4] (Corollary 8.7.1.) où $f \equiv g_i$, et grâce à l'Hypothèse 2.2., nous avons que

$$\{x^k \ \text{t.q.} \ \ g_i(x^k) \leq \beta\} \ \text{ est borné pour tout } \ \beta.$$

Prenons $\beta = \gamma_2$. Il s'ensuit que $\{x^k \ \ \text{t.q.} \ \ g_i(x^k) \leq \gamma_2\}$ est borné, et donc que $\{x^k\}$ est bornée.

Par l'inégalité de gauche de (2) et l'inégalité triangulaire, nous obtenons

$$|s_i^k| \leq |\gamma_2| + |g_i(x^k)|;$$

et puisque $\{g(x^k)\}$ est bornée, $g(\cdot)$ étant continue et $\{x^k\}$ bornée, il s'ensuit que $\{s^k\}$ est bornée.

Il nous reste maintenant à prouver que $\{y^k\}$ est bornée. Supposons qu'elle ne le soit pas. Il s'ensuit qu'il existe une sous-suite $\{y^k\}$ de $\{y^k\}$ telle que

$$\|y^k\|_2 \to +\infty. \tag{9}$$

Pour cette sous-suite, nous avons que

$$\|F_d^k\| = \|\nabla g(x^k)^T\, y^k + \nabla c(x^k)^T\| \text{ est bornée.} \tag{10}$$

En effet, par l'absurde, supposons qu'elle ne le soit pas. Alors,

$$\|\nabla g(x^k)^T y^k + \nabla c(x^k)^T\| \longrightarrow +\infty. \tag{11}$$

D'après (1),

$$-\mu \sum_{i=1}^{m} \ln(y_i^k s_i^k) \geq -\mu\, m \ln(\gamma_2), \tag{12}$$

et d'après (2),

$$-\hat{\mu} \sum_{i=1}^{m} \ln(g_i(x^k) + s_i^k) \geq -\hat{\mu}\, m \ln(\gamma_2). \tag{13}$$

Grâce à la définition de $\hat{f}(\cdot)$ vue en (2.12), (12) et (13), nous avons

$$\hat{f}(z^k) \geq \frac{1}{2} \|F_d^k\|^2 - m \ln(\gamma_2)\,(\mu + \hat{\mu}).$$

Par la définition de F_d dans (2.6) et (11),

$$\hat{f}(z^k) \to +\infty. \tag{14}$$

Comme $\hat{f}$ est décroissante, il est impossible que (14) soit vérifiée, et il en découle que l'affirmation (10) est démontrée. De (10) et du fait que $\{\nabla c(x^k)\}$ soit bornée, $c(\cdot)$ étant de classe C^2 et (x^k) bornée, nous déduisons que

$$\{\nabla g(x^k)^T y^k\} \text{ est bornée.} \tag{15}$$

Posant $d^k = \frac{y^k}{e^T y^k}$, il existe une sous-suite $\{d^k\}$ de $\{d^k\}$ telle que

$$d^k \to d^* \tag{16}$$

$$d^* \geq 0 \tag{17}$$

$$e^T d^* = 1 \tag{18}$$

$$\nabla g(x^*)^T d^* = 0. \tag{19}$$

(16) découle du fait que $\{d^k\}$ est bornée, et (17) et (18) sont évidentes. Prouvons (19) : nous savons, grâce à (15), que $\{\nabla g(x^k)^T y^k\}$ est bornée. $\frac{\nabla g(x^k)^T y^k}{e^T y^k} = \frac{\nabla g(x^k)^T y^k}{\|y^k\|_1}$ est donc bornée et elle admet alors une sous-suite qui converge, à savoir

$$\nabla g(x^k)^T \frac{y^k}{e^T y^k} \to \nabla g(x^*)^T d^*. \tag{20}$$

Or,

$$\frac{1}{e^T y^k} \to 0 \tag{21}$$

car

$$\|y^k\|_2 \leq e^T y^k = \|y^k\|_1,$$

d'où

$$\frac{1}{\|y^k\|_2} \geq \frac{1}{e^T y^k}. \tag{22}$$

Mais nous savons par (9) que $\|y^k\|_2 \to +\infty$, et donc que

$$\frac{1}{\|y^k\|_2} \to 0,$$

et grâce à (22), nous avons bien (21). Nous savons aussi, par (15), que $\{\nabla g(x^k)^T y^k\}$ est bornée, et il en est donc de même pour la sous-suite $\{\nabla g(x^k)^T y^k\}$. De (21) et de ce qui précède,

$$\nabla g(x^k)^T y^k \frac{1}{e^T y^k} \to 0, \tag{23}$$

et nous pouvons conclure (19) grâce à (20), (23), et par unicité de la limite.

(1) nous indique que $(s^k)^T y^k$ est bornée, et par un raisonnement analogue à celui de la preuve de (19), nous avons que

$$(s^*)^T d^* = 0. \tag{24}$$

Soit l'ensemble

$$I = \{i \text{ t.q. } s_i^* = 0\},$$

alors, par (2) et (24), nous obtenons directement

$$g_i(x^*) > 0, \quad i \in I, \tag{25}$$

et

$$d_i^* = 0, \quad i \notin I. \tag{26}$$

Il s'ensuit aisément que

$$\begin{aligned} d_I^* &\geq 0, \\ e^T d_I^* &= 1, \\ (d_I^*)^T \nabla g_I(x^*) &= 0, \end{aligned}$$

où la notation I en indice indique les composantes correspondant à $i \in I$. Le théorème d'alternative de Gordan

Soit $\left| \begin{array}{l} A \in I\!R^{n \times n}, \\ x \in I\!R^n \ \ et \\ u^T \in I\!R^m, \end{array} \right.$

alors, exactement une de ces conditions a lieu :

(1) $\exists\, x$ *t.q.* $A\,x < 0$

(2) $\exists\, u \neq 0$ *t.q.* $u\,A = 0 \quad u \geq 0,$

(2) étant vérifiée avec $u \equiv d_I^*$ et $A \equiv \nabla g_I(x^*) = 0$, implique alors qu'

il n'existe pas de vecteur p t.q. $\nabla g_I(x^*)\, p < 0.$

Cependant, ceci est une contradiction : étant donné que les fonctions $g_i(\cdot)$ sont convexes par hypothèse, que $g_I(x^*) > 0$ d'après (25), et que pour un point admissible $\bar{x}$, $g_I(\bar{x}) \leq 0$, nous avons, parce que le graphe d'une fonction convexe est toujours au-dessus de son plan tangent, que la direction $p = \bar{x} - x^*$ satisfait $\nabla g_I(x^*)p < 0$. Cette contradiction nous permet de conclure que $\{y^k\}$ est bien bornée, et de clore cette preuve. ■

Remarquons que la preuve du Théorème 2.2. n'utilise pas complètement l'Hypothèse 2.1.; elle demande seulement un point $\bar{x}$ admissible (comme opposé à intérieur). Par le Théorème 2.2., l'algorithme pour un $\mu_j > 0$ fixé produira finalement un itéré $z^k = \tilde{z}^j$ satisfaisant (2.14) où $\mu = \mu_j$. A ce point, la valeur de μ est réduite :

$$\mu_{j+1} = (1 - \theta)\mu_j, \qquad 0 < \theta < 1,$$

et le processus itératif peut continuer. Pour clore ce chapitre, il nous faut prouver que, lorsque $\mu_j \to 0$, la suite des $\{\tilde{z}^j\}$ admet un point d'accumulation z^* fournissant une solution optimale pour le problème (1.1) de départ. Ceci est le propos du théorème suivant, dont la preuve est similaire à celle du Théorème 2.2..

Théorème 2.3. *Soit* | *μ_j, $j \geq 0$, une suite de termes positifs telle que $\mu_j \to 0$,*
$\tilde{z}^j = (\tilde{y}^j, \tilde{s}^j, \tilde{x}^j)$, satisfaisant (2.14) où $\mu = \mu_j$, $j \geq 0$.

Alors, $\{\tilde{z}^j\}$ est bornée, et

si $z^* = (y^*, s^*, x^*)$ est un point d'accumulation de $\{\tilde{z}^j\}$,
alors, x^* est une solution optimale de (2.1).

Preuve :

Grâce à (2.14), la définition de F_p en (2.6), et le fait que $\mu_j \to 0$, nous avons immédiatement que

$$g(\tilde{x}^j) + \tilde{s}^j \to 0. \tag{1}$$

De même, en utilisant la définition de F_d en (2.6), nous obtenons

$$\nabla g(\tilde{x}^j)^T \tilde{y}^j + \nabla c(\tilde{x}^j)^T \to 0. \tag{2}$$

Par (1), la suite $\{\tilde{x}^j\}$ est bornée. La suite $\{\tilde{s}_j\}$ est également bornée. En effet, $g(\cdot)$ étant continue et $\{\tilde{x}^j\}$ bornée, $g(\tilde{x}^j)$ est bornée et $g(\tilde{x}^j) + \tilde{s}^j$ étant convergente, est bornée.

Etant donné que $\mu_j \to 0$, grâce à (2.14) et à la définition de v, nous obtenons facilement que

$$\tilde{s}_i^j \, \tilde{y}_i^j \to 0, \quad 1 \leq i \leq m. \tag{3}$$

Nous prétendons que la suite des variables duales $\{\tilde{y}^j\}$ est aussi bornée. Supposons qu'elle ne le soit pas. Il existe alors une sous-suite $\{\tilde{y}^j\}$ de $\{\tilde{y}^j\}$ telle que

$$\|\tilde{y}^j\|_2 \to +\infty.$$

Pour cette sous-suite,

$$\nabla g(\tilde{x}^j)^T \tilde{y}^j + \nabla c(\tilde{x}^j)^T \text{ est bornée,}$$

étant donné que, selon (2), $\nabla g(\tilde{x}^j)^T \tilde{y}^j + \nabla c(\tilde{x}^j)^T$ admet une sous-suite convergente. Il s'ensuit, par un raisonnement analogue à celui de la preuve du théorème 2.2., que

$$\text{les valeurs } \{\nabla g(\tilde{x}^j)^T \tilde{y}^j\} \text{ sont bornées.} \tag{4}$$

Posant $\tilde{d}^j = \frac{\tilde{y}^j}{e^T \tilde{y}^j}$, il existe une sous-suite $\{\tilde{d}^j\}$ de $\{\tilde{d}^j\}$ telle que

$$\begin{aligned} \tilde{d}^j &\to d^* \\ d^* &\geq 0 \\ e^T d^* &= 1 \\ \nabla g(x^*)^T d^* &= 0, \end{aligned} \tag{5}$$

par un raisonnement exactement analogue à celui de la preuve du théorème 2.2., et

$$(s^*)^T d^* = 0, \tag{6}$$

car $(\tilde{s}^j)^T \tilde{y}^j$ est bornée selon (3). Soit $I = \{i \text{ t.q. } s_i^* = 0\}$. Alors,

$$g_I(x^*) = 0, \tag{7}$$

$$d_I^* \geq 0, \tag{8}$$

$$e^T d_I^* = 1, \quad \text{et} \tag{9}$$

$$(d_I^*)^T \nabla g_I(x^*) = 0. \tag{10}$$

Les relations (8) et (10) sont évidentes; montrons (7) :
Etant donné que $\{\tilde{x}^j\}$ et $\{\tilde{s}^j\}$ sont bornées, il existe deux sous-suites telles que

$$(\tilde{x}^j, \tilde{s}^j) \to (x^*, s^*).$$

$g(\cdot)$ étant continue, nous obtenons

$$g(\tilde{x}^j) + \tilde{s}^j \to g(x^*) + s^*.$$

Nous avons aussi, grâce à (1) et par unicité de la limite, que

$$g(x^*) + s^* = 0,$$

et la définition de I nous permet de conclure (7). Nous savons que

$$d_{I^c}^* = 0. \tag{11}$$

En effet, la définition de I entraîne que

$$s_{I^c}^* > 0, \tag{12}$$

et (11), grâce à (6) et (12). (9) découle alors directement de (5) et (11). Le théorème

d'alternative de Gordan cité dans la preuve du théorème 2.2. implique alors que

$$\nexists\, p \text{ t.q. } \nabla g_I(x^*)\, p < 0.$$

Cependant, ceci est une contradiction, étant donné que les fonctions $g_i(\cdot)$ sont convexes, $g_I(x^*) = 0$, et que d'après l'Hypothèse 2.1., il existe un $\bar{x}$ tel que $g_I(\bar{x}) < 0$, car nous avons alors que la direction $p = \bar{x} - x^*$ doit satisfaire $\nabla g_I(x^*)\, p < 0$.

Puisque $\{\tilde{y}^j\}$ est bornée, $\{\tilde{z}^j\}$ est bornée aussi. Cette dernière admet donc une sous-suite $\{\tilde{z}^j\}$ convergeant vers le point $z^* = (y^*, s^*, x^*)$. Pour un tel z^*, grâce aux relations (3), (1) et (2), il est immédiat que

$$\begin{aligned} s_i^*\, y_i^* &= 0 \quad 1 \leq i \leq m, \\ g(x^*) + s^* &= 0, \\ \nabla g(x^*)^T\, y^* + \nabla c(x^*)^T &= 0, \end{aligned}$$

ce qui équivaut exactement à $F(z^*) = 0$, et donc, x^* satisfait les équations de KKT pour (2.1), qui sont suffisantes pour l'optimalité. ■

2.2 Approche expérimentale

Nous avons implémenté l'algorithme avec μ variable, qui n'est en fait rien d'autre que l'algorithme "à pas longs" de la page 11 avec, pour fonction de mérite, $h \equiv \hat{f}$. C'est en FORTRAN, sur VAX 6220/VMS version V5.5-2 des Facultés Universitaires Notre-Dame de la Paix à Namur que nous avons réalisé cette implémentation.
Un listing de notre programme est donné en annexe.

Le point de départ x^0 est introduit par l'utilisateur et les vecteurs y^0, s^0, ainsi que le paramètre μ_0, sont calculés selon la règle du paragraphe 1.2.1. Nous avons choisi comme suit les valeurs des paramètres :

$$
\begin{aligned}
\varepsilon &= 10^{-10} \\
\theta &= 0.5 \\
\lambda_1 &= 0.1 \\
\beta &= 0.25 \\
\hat{\mu} &= \frac{\beta\mu}{m}
\end{aligned}
$$

Comme vu précédemment, l'algorithme comporte des itérations internes et externes. Après expérimentation, nous avons dû limiter à 100 le nombre d'itérations internes. Nous avons utilisé la recherche linéaire caractérisée par la condition (2.10) rappelée ici :

α est choisi en utilisant une stratégie "backtracking" de la forme $\alpha = \theta^p \, \alpha_{\max}$, où $\alpha_{\max}$ est le plus grand α tel que $(y + \alpha\, dy, s + \alpha\, ds) \geq 0,\ 0 < \theta < 1,$ et p est le plus petit entier strictement positif tel que

$$
f(z) - f(z + \theta^p \, \alpha_{\max} \, dz) \;\geq\; \lambda_1 \, \theta^p \, \alpha_{\max} \, |Df(z; dz)|, \tag{2.10}
$$

où $0 < \lambda_1 < 1,$

et nous avons dû imposer une borne supérieure, en l'occurence 70, sur le nombre p. De plus, chaque fois que la condition de Newton est calculée, nous testons si celle-ci est bien une direction de descente pour $\hat{f}(\cdot)$. Si c'est le cas, on peut passer aux itérations internes et sinon, μ est diminué.

Comme pour l'algorithme du premier chapitre, nous avons choisi le test d'arrêt de l'algorithme de la façon suivante :

$$
\max \{ \text{ relgap } , \text{ relp } , \text{ reld } \} < \varepsilon
$$

où relgap $:= \frac{y^T s}{(1+|y^T s|)}$, relp $:= \frac{\|F_p\|}{1+\|x\|}$, reld $:= \frac{\|F_d\|}{1+\|y\|}$ indiquent la décroissance du saut de dualité et la convergence vers l'admissibilité primale et duale.

2.3 Résultats numériques

2.3.1 Problème général

Le programme du paragraphe précédent permet de résoudre un problème de programmation convexe, non linéaire et différentiable du type :

$$\begin{cases} \min & c(x) \\ \text{s.c.} & g(x) \leq 0, \end{cases}$$

où $x \in \mathbb{R}^n$, $c(\cdot) : \mathbb{R}^n \to \mathbb{R}^1$, $g(\cdot) : \mathbb{R}^n \to \mathbb{R}^m$, où $c(\cdot)$ et chaque $g_i(\cdot)$ sont des fonctions convexes et de classe C^2.

2.3.2 Problème 1

Données :

$$
\begin{aligned}
n &= 2 \\
m &= 1 \\
c(x) &= 2\,x_1 + 3\,x_2 \\
g_1(x) &= x_1^2 + x_2^2 - 1 \\
\text{Point de départ } x^0 &= (10, 10)
\end{aligned}
$$

Résultats :

$$
\begin{aligned}
\text{Vecteur solution } x_1^* &= -0.55470020 \\
x_2^* &= -0.83205029 \\
\text{Valeur optimale } c(x^*) &= -3.605551275402202 \\
\text{Nombre d'itérations externes} &= 40 \\
\text{Nombre de directions calculées} &= 348 \\
\text{Temps CPU (sec.)} &= 27.98
\end{aligned}
$$

2.3.3 Problème 2

Données :

$$
\begin{aligned}
n &= 2 \\
m &= 2 \\
\\
c(x) &= x_1^2 - x_2 \\
\\
g_1(x) &= x_1^2 + x_2^2 - 1 \\
g_2(x) &= 0.5 - x_2 \\
\\
\text{Point de départ } x^0 &= (12, 15)
\end{aligned}
$$

Résultats :

$$
\begin{aligned}
\text{Vecteur solution } x_1^* &= 0.00000000000 \\
x_2^* &= 1.00000000000 \\
\\
\text{Valeur optimale } c(x^*) &= -1.00000000000 \\
\\
\text{Nombre d'itérations externes} &= 41 \\
\text{Nombre de directions calculées} &= 413 \\
\\
\text{Temps CPU (sec.)} &= 50.33
\end{aligned}
$$

2.3.4 Problème 3

Données :

$$
\begin{aligned}
n &= 2 \\
m &= 2 \\
\\
c(x) &= x_1^2 - x_2 \\
\\
g_1(x) &= x_1^2 + x_2^2 - 1 \\
g_2(x) &= (x_1 + 1)^2 + x_2^2 - 0.5 \\
\\
\text{Point de départ } x^0 &= (8, 8)
\end{aligned}
$$

Résultats :

$$
\begin{aligned}
\text{Vecteur solution } x_1^* &= -0.50000000000 \\
x_2^* &= 0.50000000000 \\
\\
\text{Valeur optimale } c(x^*) &= -0.2500000000000000 \\
\\
\text{Nombre d'itérations externes} &= 40 \\
\text{Nombre de directions calculées} &= 359 \\
\\
\text{Temps CPU (sec.)} &= 53.74
\end{aligned}
$$

2.3.5 Problème 4

Données :

$$
\begin{aligned}
n &= 2 \\
m &= 2 \\
\\
c(x) &= e^{x_1} + e^{x_2} \\
\\
g_1(x) &= (x_1 - 1)^2 + x_2^2 - 1 \\
g_2(x) &= (x_1 + 1)^2 + x_2^2 - 4 \\
\\
\text{Point de départ } x^0 &= (-5, -3)
\end{aligned}
$$

Résultats :

$$
\begin{aligned}
\text{Vecteur solution } x_1^* &= 0.12276952 \\
x_2^* &= -0.48006946 \\
\\
\text{Valeur optimale } c(x^*) &= 1.749364218299988 \\
\\
\text{Nombre d'itérations externes} &= 46 \\
\text{Nombre de directions calculées} &= 256 \\
\\
\text{Temps CPU (sec.)} &= 41.12
\end{aligned}
$$

2.3.6 Problème 5

Données :

n = 2
m = 1

$c(x)$ = $x_1^4 + 3\,x_2^2$

$g_1(x)$ = $x_1^2 - x_2$

Point de départ x^0 = $(-10, 10)$

Résultats :

Vecteur solution x_1^* = 0.00000000000
x_2^* = 0.36828992D−05

Valeur optimale $c(x^*)$ = 4.0691240191503669D−11

Nombre d'itérations externes = 39
Nombre de directions calculées = 416

Temps CPU (sec.) = 33.13

2.3.7 Problème 6

Données :

$$
\begin{aligned}
n &= 6 \\
m &= 3 \\
\\
c(x) &= x_1^4 + 2\,x_2^4 + 2\,x_3^4 + x_4^4 + x_5^4 + x_6^4 \\
\\
g_1(x) &= x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 - 1 \\
g_2(x) &= (x_1 + 1.5)^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 - 1 \\
g_3(x) &= (x_1 + 1)^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 - 1 \\
\\
\text{Point de départ } x^0 &= (2, 2, 2, 2, 2, 2)
\end{aligned}
$$

Résultats :

$$
\begin{aligned}
\text{Vecteur solution } x_1^* &= -0.50000000000 \\
x_2^* &= 0.00000000000 \\
x_3^* &= 0.00000000000 \\
x_4^* &= 0.00000000000 \\
x_5^* &= 0.00000000000 \\
x_6^* &= 0.00000000000 \\
\\
\text{Valeur optimale } c(x^*) &= 6.2500000000000000\text{D}-02 \\
\\
\text{Nombre d'itérations externes} &= 38 \\
\text{Nombre de directions calculées} &= 117 \\
\\
\text{Temps CPU (sec.)} &= 49.47
\end{aligned}
$$

Conclusion

Nous avons étudié une méthode de points intérieurs pour le cas convexe, différentiable et non admissible. Nous en avons établi la convergence et rapporté quelques résultats numériques obtenus avec notre implémentation de la méthode.

Il serait intéressant, dans une étude future, de tester de manière plus intensive cette méthode, notamment sur des exemples de plus grande taille.

Bibliographie

[1] ANSTREICHER K.M. and VIAL J.-P. On the convergence of an infeasible primal-dual interior-point method for convex programming. *Technical report 1993.16*, Department of Management Studies, University of Geneva (Geneva, Switzerland), 1993.

[2] FIACCO A.V. and MCCORMICK G.P. Nonlinear programming : Sequential unconstrained minimization techniques. John Wiley and Sons, New York, 1968.

[3] GOLDSTEIN A. and PRICE J. An effective algorithm for minimization. *Numer. Math. 10*, pages 184–189, 1967.

[4] ROCKAFELLAR R.T. Convex analysis. Princeton University Press, 1970.

[5] VIAL J.-P. Computational experience with a primal-dual interior-point method for smooth convex programming. *Technical report 1992.7*, Department of Management Studies, University of Geneva (Geneva, Switzerland), 1992.

Annexe 1

Dans cette annexe, nous fournissons, pour la facilité du lecteur, une preuve d'un résultat de convergence de base pour les conditions (2.9) de Goldstein-Armijo. Soit $f(\cdot) : Z \to \mathbb{R}$, continûment différentiable et minorée sur un ouvert $Z \subset \mathbb{R}^n$. Considérons un schéma itératif

$$z^{k+1} = z^k + \alpha^k \, dz^k,$$

où $Df(z^k; dz^k) < 0$, et α^k satisfait les conditions (2.9) de diminution suffisante. Nous nous référons à l'inégalité de gauche de (2.9), où $\alpha = \alpha^k$, comme étant la condition (I), et à celle de droite comme étant la condition (II).

Théorème A.1. *Supposons que $\{z^k\}$ ait une sous-suite convergente dans Z,*

alors, cette sous-suite, $\{z^k\}$, vérifie $\quad \frac{Df(z^k; dz^k)}{\|dz^k\|} \to 0.$

Preuve :

En modifiant les directions dz^k et les longueurs de pas α^k, sans perdre de généralité, on peut supposer que $\|dz^k\| = 1$ pour tout k, et montrer que dans ce cas,

$$Df(z^k; dz^k) \underset{k \in L}{\to} 0. \tag{1}$$

Etant donné que

$$\|dz^k\| = 1 \quad \forall k \in \mathbb{N},$$

de (1), nous pouvons déduire

$$\frac{Df(z^k; dz^k)}{\|dz^k\|} \underset{k\in L}{\rightarrow} 0. \tag{2}$$

Par l'absurde, supposons que la relation (1) ne soit pas vérifiée. Il existe alors une sous-suite $\{Df(z^k; dz^k)\}_{k\in K}$ de $\{Df(z^k; dz^k)\}_{k\in L}$, avec $K \subset L$, telle que

$$|Df(z^k; dz^k)| \geq \gamma > 0 \qquad \text{pour } k \in K. \tag{3}$$

Nous savons que

$$\alpha^k \underset{k\in K}{\rightarrow} 0. \tag{4}$$

En effet, par hypothèse,

$$\forall k \in L, \quad z^k \in Z.$$

Or, $K \subset L$, d'où

$$\forall k \in K, \quad z^k \in Z,$$

et donc, $f(\cdot)$ étant minorée sur Z,

$$\forall k \in K, \quad f(z^k) \geq m. \tag{5}$$

Par (I) et (3), nous avons

$$f(z^k) - f(z^{k+1}) \geq \lambda_1 \alpha^k \gamma > 0. \tag{6}$$

Si nous montrons que

$$\sum_{k=1}^{\infty} \alpha^k \text{ converge}, \tag{7}$$

nous aurons bien (4). Grâce à (6),

$$0 < \lambda_1 \gamma \sum_{k=1}^{n} \alpha^k \leq \sum_{k=1}^{n} \left(f(z^k) - f(z^{k+1}) \right),$$

ce qui revient à

$$0 < \lambda_1 \gamma \sum_{k=1}^{n} \alpha^k \leq f(z^1) - f(z^{n+1}),$$

et grâce à (5),

$$0 < \lambda_1 \gamma \sum_{k=1}^{n} \alpha^k \leq f(z^1) - m.$$

En faisant tendre n vers l'infini dans cette dernière inégalité, nous prouvons alors l'affirmation (7), et donc également (4). Cependant, pour tout k, nous avons

$$f(z^{k+1}) = f(z^k) + \alpha^k \left[\nabla f(z^k + \alpha^k \eta^k \, dz^k) \right] dz^k,$$

pour un η^k tel que $0 < \eta^k < 1$, ainsi,

$$\frac{f(z^{k+1}) - f(z^k)}{\alpha^k} = \left[\nabla f(z^k + \alpha^k \eta^k \, dz^k) \right] dz^k,$$

d'où

$$\frac{f(z^{k+1}) - f(z^k)}{\alpha^k} - Df(z^k; dz^k) = \left[\nabla f(z^k + \alpha^k \eta^k \, dz^k) - \nabla f(z^k) \right] dz^k \underset{k \in K}{\rightarrow} 0, \qquad (8)$$

étant donné que $f(\cdot)$ est de classe C^1, $\|dz^k\| = 1$, $\alpha^k \underset{k \in K}{\rightarrow} 0$, et que $\{z^k \text{ t.q. } k \in K\}$ converge. Cependant, d'après la condition (II), nous avons

$$f(z^k) - f(z^{k+1}) - \alpha^k \left| Df(z^k; dz^k) \right| \leq \lambda_2 \, \alpha^k \left| Df(z^k; dz^k) \right| - \alpha^k \left| Df(z^k; dz^k) \right|,$$

ce qui implique que

$$\frac{f(z^k) - f(z^{k+1})}{\alpha^k} - \left| Df(z^k; dz^k) \right| \leq (\lambda_2 - 1) \left| Df(z^k; dz^k) \right|. \qquad (9)$$

Mais ceci est une contradiction. En effet, d'une part, grâce à (8), nous savons que le membre de gauche de l'inégalité (9) converge vers 0 pour $k \in K$, et d'autre part, de (3), il résulte que le membre de droite de (9) est strictement négatif pour $k \in K$, étant donné que nous avons $\lambda_2 < 1$. ■

Annexe 2 : Programme FORTRAN

```
      PROGRAM OPTI
C
C     ************************************************************************
C     ************************************************************************
C     **                                                                    **
C     **                                                                    **
C     **   Ce programme permet de résoudre un problème de programmation     **
C     **                          convexe, non linéaire, différentiable :   **
C     **                                                                    **
C     **                                                                    **
C     **                          MIN  c(x)                                 **
C     **                          S.C. g(x) <= 0                            **
C     **                                                                    **
C     **                                                                    **
C     **                       où .x appartient à Rn                        **
C     **                          .c(.) : Rn --> R                          **
C     **                          .g(.) : Rn --> Rm                         **
C     **                                                                    **
C     **  .c(.) et chaque gi(.) sont des fonctions convexes et de classe C2 **
C     **                                                                    **
C     **                                                                    **
C     ************************************************************************
C     ************************************************************************
C
C
C     1. DECLARATIONS
C     ***************
C
C
C     1.1. DECLARATION DES PARAMETRES
C     *******************************
C
C     EPS  : (double précision) - paramètre de précision
C     TETA : (double précision) - facteur de réduction de mu
C     BETA : (double précision) -
C     LBD1 : (double précision) -
C
C     DOUBLE PRECISION EPS, TETA, LBD1
C     PARAMETER (EPS = 1.D-10, TETA = 0.5D0, LBD1 = 0.1D0)
C
C
C     1.2. DECLARATION DES VARIABLES
C     ******************************
C
C     mmax    : (integer) - dimension maximale des vecteurs y et s
C     nmax    : (integer) - dimension maximale du vecteur x
C     rmax    : (integer) - dimension maximale des vecteurs z et dz
C     NBitEXT : (integer) - nombre d'itérations externes
C     NBitINT : (integer) - nombre d'itérations internes
C
C     EP      : (logique) - (1) = .TRUE. si l'utilisateur désire que les
C                                 résultats intermédiaires apparaissent
C                                 à l'écran
C                           (2) = .TRUE. si l'utilisateur désire que les
C                                 résultats intermédiaires soient stockés
C                                 dans un fichier
C     FICHIER : (logique) - (1) = .TRUE. si l'utilisateur désire que les
C                                 données soient stockées dans un fichier
C                           (2) = .TRUE. si l'utilisateur désire que les
C                                 résultats intermédiaires soient stockés
C                                 dans un fichier
C     IMPR    : (logique) - cette variable = .TRUE. si l'utilisateur désire
C                           afficher/imprimer les résultats intermédiaires
C
C     MU      : (double précision) -
C     compteur : (entier) ENTREE et SORTIE - compteur du nombre de
C                résolutions du système linéaire dont la solution est dz
C
C
      INTEGER mmax, nmax, rmax
      PARAMETER (mmax = 100, nmax = 100, rmax = 2*mmax+nmax)
C
      INTEGER I,NBitEXT,NBitINT,dim1,dim2,dim3
      LOGICAL FICHIER(2)
      DOUBLE PRECISION zW(rmax),zWW(rmax),fchapzW
C
C     INTRINSIC DBLE
C
      INTEGER m, n, r
      COMMON/DIM/m, n, r
C
      DOUBLE PRECISION y, s, x, z
      COMMON/VECTz/y(mmax), s(mmax), x(nmax), z(rmax)
C
      DOUBLE PRECISION dz
      COMMON/VECTdz/dz(rmax)
C
      LOGICAL IMPR, EP
      COMMON/LOGI/IMPR(8),EP(2)
C
      INTEGER UL
      COMMON/LU/UL
C
      DOUBLE PRECISION MU
      COMMON/DP/MU
C
      DOUBLE PRECISION BETA,yTs,nrmFp2,nrmFd2
      COMMON/BLOC/BETA,yTs,nrmFp2,nrmFd2
C
      INTEGER compteur
      COMMON/PLUS/compteur
C
C
C
C     2. IMPLEMENTATION
C     *****************
C
C
C
C     2.1. LECTURE DES DONNEES
C     ************************
C
C
      CALL DONNEES
C
C      -------------------------
C     |                  T      |
C     |                (y0).s0  |
C     |   MU = BETA . -------   |
C     |                  m      |
C     |                         |
C      -------------------------
C
      BETA = 0.25D0
      MU   = ((BETA * DDOT(m,y,1,s,1))/DBLE(m))
C
C
C
C     2.2. IMPRESSION DES DONNEES
```

```
C       ***************************
C
C
        CALL IMPRDONN (EPS,TETA,LBD1,FICHIER)
C
C
C
C       2.3. RESOLUTION DU PROBLEME    MIN  c(x)
C       ***************************    S.C. g(x)<=0
C
        CALL INTERMED (FICHIER)
        CALL RESOLVE (EPS,TETA,LBD1,NBitEXT,NBitINT)
C
C
C
C       2.4. IMPRESSION DES RESULTATS
C       *****************************
C
        CALL IMPRRESU (FICHIER)
C
C
C       2.5. SOUS-ROUTINES UTILISEES
C       ****************************
C
C
C        ------------------------------------------------
C       |                                                |
C       |   REMARQUE : Les sous-routines EVALc et EVALg  |
C       |   --------   se trouvent toutes deux dans le  |
C       |              fichier TESTi.FOR                 |
C       |                                                |
C        ------------------------------------------------
C
C
C        ------------------------------------------
C       |                                          |
C       |   Sous-routines dans l'ordre : 1) DONNEES |
C       |   --------------------------   2) IMPRDONN |
C       |                                3) INTERMED |
C       |                                4) IMPRRESU |
C       |                                5) RESOLVE  |
C       |                                6) EVALfcha |
C       |                                7) DERIVDIR |
C       |                                8) RECHLIN  |
C       |                                9) EVALF    |
C       |                                          |
C        ------------------------------------------
C
C
        END
C
C       ************************************************************************
C       ************************************************************************
```

```
C       1111111111111111111111111111111111111111111111111111111111111111111111111111
C
        SUBROUTINE DONNEES
C
C       ************************************************************************
C       *                                                                      *
C       *                                                                      *
C       *   1) Affichage à l'écran de ce que le programme OPTI.FOR résout      *
C       *   2) Introduction des valeurs de m, n et x0 par l'utilisateur        *
C       *   3) Calcul de y0 et s0                                              *
C       *                                                                      *
C       *                                                                      *
C       ************************************************************************
C
C
C       1. DECLARATION DES VARIABLES LOCALES
C       ************************************
C
C
C
        INTEGER mmax, nmax, rmax
        PARAMETER (mmax = 100, nmax = 100, rmax = 2*mmax+nmax)
C
        INTEGER I,INDICEk
        LOGICAL PTI
        CHARACTER*1 rep
        DOUBLE PRECISION gx(mmax),MINkgkx0,Fp(mmax)
C
C
        INTEGER m, n, r
        COMMON/DIM/m, n, r
C
        DOUBLE PRECISION y, s, x, z
        COMMON/VECTz/y(mmax), s(mmax), x(nmax), z(rmax)
C
        INTEGER UL
        COMMON/LU/UL
C
C
C
C
C
C       2. IMPLEMENTATION
C       *****************
C
C
C       2.1. COMMENTAIRE
C       ****************
C
        WRITE (*,11111)
11111   FORMAT(1X,20(/))
        WRITE (*,*) 'Ce programme permet de résoudre un problème de ',
     *'programmation convexe, non linéaire, différentiable :          ',
     *'                                                               ',
     *'                       MIN  c(x)                               ',
     *'                       S.C. g(x) <= 0                          ',
     *'                                                               ',
     *'                                                               ',
     *'                   où .x appartient à Rn                       ',
     *'                      .c(.) : Rn --> R                         ',
     *'                      .g(.) : Rn --> Rm                        ',
     *'                                                               ',
     *' .chaque gi(.) est une fonction convexe et différentiable      ',
     *' .c(.) est une fonction convexe, différentiable et de classe C2',
     *'                                                               ',
     *'                                                               ',
     *'                                                               ',
```

```
     *'                                                                  ',
     *'Votre fonction objectif c(x) et votre vecteur des                 ',
     *'containtes g(x) se trouvent dans le fichier TESTi.FOR,            ',
     *'respectivement dans les SOUS-ROUTINES EVALc et EVALg.             ',
     *'                                                                  ',
     *'                                                                  ',
C
C
C       2.2. ENTREES DE m, n et x0
C       **************************
C
        WRITE (*,11112)
11112   FORMAT(1X,'UL (2 : fichier; 5 : écran) = ',$)
        READ  (*,*) UL
        WRITE (*,11113)
11113   FORMAT(1X,'Nombre de contraintes (max. 100) m = ',$)
        READ  (*,11115) m
        WRITE (*,11114)
11114   FORMAT(1X,'Nombre de variables   (max. 100) n = ',$)
        READ  (*,11115) n
11115   FORMAT(I3)
        r = 2 * m + n
        DO 111 I = 1, n
           WRITE (*,11116) I
11116      FORMAT(1X,'Votre point de départ    x0(',I3,')   = ',$)
           READ  (*,*) x(I)
111     CONTINUE
        WRITE (UL,*)
        WRITE (UL,*)
C
C
C       2.3. CALCUL DE s0
C       *****************
C
C
C        ------------------------------------------------------------------
C       |                                                                  |
C       |   SI     gi(x0) > 0    pour tout i,                              |
C       |                                                                  |
C       |   ALORS, s0 = e        (c-à-d  s0(i) = 1   1<=i<=m)              |
C       |                                                                  |
C       |   SINON, s0(i) = MAX {-gi(x0),0} - 0.5 MIN {  MIN  {gk(x0)},-1}  |
C       |                                              1<=k<=m             |
C       |                                                                  |
C        ------------------------------------------------------------------
C
C                                              PTI = Pour Tout I
C                                                  = .FALSE. s'il existe i
C                                                     tel que gi(x0) <= 0
C
        I = 1
        PTI = .TRUE.
        CALL EVALg (m,n,x,gx)
1111    IF ((I .LE. m) .AND. (PTI .EQ. .TRUE.)) THEN
           IF (gx(I) .LE. 0.0D0) PTI = .FALSE.
           I = I + 1
           GOTO 1111
        ENDIF
C                                              MINkgkx0 =  MIN {gk(x0)}
C                                                        1<=k<=m
        INDICEk  = IDMIN (m,gx,1)
        MINkgkx0 = gx(INDICEk)
        IF (PTI .EQ. .TRUE.) THEN
           DO 1 I = 1, m
              s(I) = 1.0D0
1          CONTINUE
        ELSE
           DO 2 I = 1, m
             s(I) = DMAX1(-gx(I),0.0D0)
              s(I) = s(I) - (0.5D0 * DMIN1(MINkgkx0,-1.0D0))
2          CONTINUE
        ENDIF
C
C
C       2.4. CALCUL DE y0
C       *****************
C
C
C        ---------------------------------------
C       |                                       |
C       |         || s0 + g(x0) ||        -1     |
C       |   y0 = ----------------- . (S0) . e   |
C       |                 m                     |
C       |                                       |
C        ---------------------------------------
C
C                                                    Fp = s0 + g(x0)
        DO 3 I = 1, m
           Fp(I) = s(I) + gx(I)
3       CONTINUE
        DO 4 I = 1, m
           y(I) = (DNRM2 (m,Fp,1)/(DBLE(m) * s(I)))
4       CONTINUE
        DO 5 I = 1, m
           z(I) = y(I)
           z(m + I) = s(I)
5       CONTINUE
        DO 6 I = 1, n
           z(2*m+I) = x(I)
6       CONTINUE
C
C
        RETURN
        END
```

```
C       22222222222222222222222222222222222222222222222222222222222222222222222222222222
C
        SUBROUTINE IMPRDONN (EPS,TETA,LBD1,FICHIER)
C
C       ****************************************************************
C       *                                                              *
C       *                                                              *
C       *   Affichage des données à l'écran et                         *
C       *   écriture dans un fichier selon le choix de l'utilisateur   *
C       *                                                              *
C       *                                                              *
C       ****************************************************************
C
C
C       1. DECLARATION DES VARIABLES
C       ****************************
C
C
C       1.1. SPECIFICATION DES ARGUMENTS DE LA SOUS-ROUTINE
C       ***************************************************
C
C       EPS     : (double précision) ENTREE - paramètre de précision
C       TETA    : (double précision) ENTREE - facteur de réduction de mu
C       BETA    : (double précision) ENTREE -
C       LBD1    : (double précision) ENTREE -
C       MU      : (double précision) ENTREE -
C       FICHIER : (logique) (1) SORTIE, (2) RESTE INCONNU -
C                           (1) = .TRUE. si l'utilisateur désire que les
C                                 données soient stockées dans un fichier
C                           (2) = .TRUE. si l'utilisateur désire que les
C                                 résultats intermédiaires soient stockés
C                                 dans un fichier
C
C
        LOGICAL FICHIER(2)
        DOUBLE PRECISION EPS,TETA,LBD1
C
C
C       1.2. DECLARATION DES VARIABLES LOCALES
C       **************************************
C
C
C
        INTEGER mmax, nmax, rmax
        PARAMETER (mmax = 100, nmax = 100, rmax = 2*mmax+nmax)
C
        INTEGER I
        CHARACTER*1 rep
        CHARACTER*14 nomfich
        LOGICAL PASSAGE
C
C
        INTEGER m, n, r
        COMMON/DIM/m, n, r
C
        DOUBLE PRECISION y, s, x, z
        COMMON/VECTz/y(mmax), s(mmax), x(nmax), z(rmax)
C
        INTEGER UL
        COMMON/LU/UL
C
        DOUBLE PRECISION MU
        COMMON/DP/MU
C
        DOUBLE PRECISION BETA,yTs,nrmFp2,nrmFd2
        COMMON/BLOC/BETA,yTs,nrmFp2,nrmFd2
C
C
C
C       2. IMPLEMENTATION
C       *****************
C
        PASSAGE = .FALSE.
        UL      = 5
C
        WRITE (UL,*) 'DONNEES DU PROBLEME :                              '
        WRITE (UL,*) '-------------------                                '
        WRITE (UL,*)
1000    WRITE (UL,11117) EPS
11117   FORMAT(1X,'EPSILON = ',D15.8)
        WRITE (UL,11118) TETA
11118   FORMAT(1X,'TETA    = ',D15.8)
        WRITE (UL,11119) BETA
11119   FORMAT(1X,'BETA    = ',D15.8)
        WRITE (UL,11120) LBD1
11120   FORMAT(1X,'LAMBDA1 = ',D15.8)
        WRITE (UL,11121) MU
11121   FORMAT(1X,'MU      = ',D15.8)
        WRITE (UL,*)
        DO 7 I = 1, n
           WRITE (UL,11122) I,x(I)
11122      FORMAT(1X,'x0(',I3,') = ',D15.8)
7       CONTINUE
        DO 8 I = 1, m
           WRITE (UL,11123) I,y(I)
11123      FORMAT(1X,'y0(',I3,') = ',D15.8)
8       CONTINUE
        DO 9 I = 1, m
           WRITE (UL,11124) I,s(I)
11124      FORMAT(1X,'s0('I3,') = ',D15.8)
9       CONTINUE
        WRITE (UL,*)
C
        IF (.NOT. PASSAGE) THEN
           PASSAGE = .TRUE.
           WRITE (*,*) 'Désirez-vous que ces données soient écrites ',
     *                 'dans un fichier (T si oui) ?'
           READ  (*,11125) rep
11125      FORMAT(A1)
           IF ((rep .EQ. 'T') .OR. (rep .EQ. 't')) THEN
              FICHIER(1) = .TRUE.
              UL = 2
              WRITE (*,*) 'Nom du fichier dans lequel vous désirez ',
     *                    'stocker vos données (RESULTATSi conseillé)',
     *                    '(ce sera le même fichier pour les résultats',
     *                    ' intermédiaires ainsi que les résultats) : '
              READ  (*,11126) nomfich
11126         FORMAT(A10)
              OPEN (UNIT = 2, FILE = nomfich, STATUS = 'NEW')
              GOTO 1000
           ENDIF
        ENDIF
C
C
        RETURN
        END
```

```
C     3333333333333333333333333333333333333333333333333333333333333333333333333333333333
C
      SUBROUTINE INTERMED (FICHIER)

C     ****************************************************************
C     *                                                              *
C     *                                                              *
C     *   Affichage des résultats intermédiaires à l'écran et        *
C     *   écriture dans un fichier selon le choix de l'utilisateur   *
C     *                                                              *
C     *                                                              *
C     ****************************************************************


C     1. DECLARTION DES VARIABLES
C     ***************************


C     1.1. SPECIFICATION DES ARGUMENTS DE LA SOUS-ROUTINE
C     ***************************************************

C     FICHIER : (logique) (1) ENTREE, (2) SORTIE -
C                         (1) = .TRUE. si l'utilisateur désire que les
C                               données soient stockées dans un fichier
C                         (2) = .TRUE. si l'utilisateur désire que les
C                               résultats intermédiaires soient stockés
C                               dans un fichier
C     IMPR    : (logique) (1),..., (8) SORTIE -
C                         ces variables = .TRUE. si l'utilisateur désire
C                         afficher/imprimer les résultats intermédiaires
C     EP      : (logique) (1) SORTIE, (2) SORTIE -
C                         (1) = .TRUE. si l'utilisateur désire que les
C                               résultats intermédiaires apparaissent
C                               à l'écran
C                         (2) = .TRUE. si l'utilisateur désire que les
C                               résultats intermédiaires soient stockés
C                               dans un fichier


      LOGICAL FICHIER(2)


C     1.2. DECLARATION DES VARIABLES LOCALES
C     **************************************



      INTEGER I
      CHARACTER*1 rep
      CHARACTER*14 nomfich

C
C
      LOGICAL IMPR, EP
      COMMON/LOGI/IMPR(8),EP(2)

C     2. IMPLEMENTATION
C     *****************


C     2.1. CHOIX DES RESULTATS INTERMEDIAIRES A IMPRIMER
C     **************************************************

      WRITE (*,*)
      WRITE (*,*)
      WRITE (*,*) 'Voici ce que IMPR permet d''imprimer :          ',
     *            '-------------------------------------          ',
     *            '                                               '
     *          '1 : fchap(zW), chaque fois qu''on appelle EVALfcha  ',
     *          '2 : FJAC, la matrice Jacobienne de F,""" DERIVDIR   ',
     *          '3 : dz, chaque fois qu''on appelle DERIVDIR         ',
     *          '4 : Gfchap, le gradient de f chapeau, "" DERIVDIR   ',
     *          '5 : ALPHA, solution de la recherche linéaire        ',
     *          '6 : gJAC, la matrice Jacobienne de g, "" EVALF      ',
     *          '7 : trans(Jg(x)).y, chaque fois qu''on appelle EVALF',
     *          '8 : Gc, le gradient de c, """""""""""""""""""" EVALF '
C
      WRITE (*,*)
      WRITE (*,*) 'Les impressions ne se feront que si vous ',
     *            'introduisez un T (et puis RETURN) après le ',
     *            'chiffre correspondant! '
      WRITE (*,*)
C
      DO 11 I = 1, 8
         IMPR(I) = .FALSE.
11    CONTINUE
      DO 12 I = 1, 8
         WRITE (*,11127) I
11127    FORMAT(1X,'IMPRESSION DE ',I1,' : ',$)
         READ  (*,11129) rep
         IF ((rep .EQ. 'T') .OR. (rep .EQ. 't')) IMPR(I) = .TRUE.
12    CONTINUE
      WRITE (*,*)
C
C
C     2.2. CHOIX D'AFFICHAGE A L'ECRAN ET/OU D'ECRITURE DANS UN FICHIER
C          POUR LES RESULTATS INTERMEDIAIRES
C     ********************************************************************
C
      EP(1) = .FALSE.
      WRITE (*,*) 'Désirez-vous que les résultats intermédiaires ',
     *            'apparaissent à l''écran (T si oui) ? '
      READ  (*,11129) rep
      IF ((rep .EQ. 'T') .OR. (rep .EQ. 't')) EP(1) = .TRUE.
C
      EP(2) = .FALSE.
      WRITE (*,*) 'Désirez-vous que les résultats intermédiaires ',
     *            'soient stockés dans un fichier (T si oui) ?   '
      READ  (*,11129) rep
      IF ((rep .EQ. 'T') .OR. (rep .EQ. 't')) THEN
         FICHIER(2) = .TRUE.
         EP(2)      = .TRUE.
         IF (.NOT. FICHIER(1)) THEN
            WRITE (*,*) 'Nom du fichier dans lequel vous désirez ',
     *                  'stocker vos résultats intermédiaires ',
     *                  '(RESULTATSi conseillé)                ',
     *            '(ce sera le même fichier pour les résultats) : '
            READ  (*,11128) nomfich
11128       FORMAT(A10)
            OPEN (UNIT = 2, FILE = nomfich, STATUS = 'NEW')
         ENDIF
      ENDIF
C
11129 FORMAT (A1)
C
C
      RETURN
      END
```

```
C       444444444444444444444444444444444444444444444444444444444444444444444444444444
C
        SUBROUTINE IMPRRESU (FICHIER)
C
C       ****************************************************************
C       *                                                              *
C       *                                                              *
C       *   Affichage des résultats à l'écran et                       *
C       *   écriture dans un fichier selon le choix d l'utilisateur    *
C       *                                                              *
C       *                                                              *
C       ****************************************************************
C
C
C       1. DECLARATION DES VARIABLES
C       ****************************
C
C
C       1.1. SPECIFICATION DE L'ARGUMENT DE LA SOUS-ROUTINE
C       ***************************************************
C
C       FICHIER : (logique) (1) ENTREE, (2) ENTREE -
C                           (1) = .TRUE. si l'utilisateur désire que les
C                                 données soient stockées dans un fichier
C                           (2) = .TRUE. si l'utilisateur désire que les
C                                 résultats intermédiaires soient stockés
C                                 dans un fichier
C
C
        LOGICAL FICHIER(2)
C
C
C       1.2. DECLARATION DES VARIABLES LOCALES
C       **************************************
C
C       compteur : (entier) ENTREE et SORTIE - compteur du nombre de
C                  résolutions du système linéaire dont la solution est dz
C
C
        INTEGER mmax, nmax, rmax
        PARAMETER (mmax = 100, nmax = 100, rmax = 2*mmax+nmax)
C
        INTEGER I
        LOGICAL PASSAGE
        CHARACTER*1 rep
        CHARACTER*14 nomfich
C
C
        INTEGER m, n, r
        COMMON/DIM/m, n, r
C
        DOUBLE PRECISION y, s, x, z
        COMMON/VECTz/y(mmax), s(mmax), x(nmax), z(rmax)
C
        INTEGER compteur
        COMMON/PLUS/compteur
C
        INTEGER UL
        COMMON/LU/UL
C
C
C
C       2. IMPLEMENTATION
C       *****************
C
C
C
        PASSAGE = .FALSE.
        UL      = 5
C
2000    DO 13 I = 1, m
           IF (((y(I) .GE. 0.0D0) .AND. (y(I) .LE. 1.0D-9)) .OR.
     &          ((y(I) .LE.0.0D0) .AND. (y(I) .GE. (-1.0D-9)))) THEN
              y(I) = 0.0D0
           ENDIF
           WRITE (UL,11130) I,y(I)
11130      FORMAT(1X,'y(',I3,') = ',D15.8)
13      CONTINUE
        DO 14 I = 1, m
           IF (((s(I) .GE. 0.0D0) .AND. (s(I) .LE. 1.0D-9)) .OR.
     &          ((s(I) .LE.0.0D0) .AND. (s(I) .GE. (-1.0D-9)))) THEN
              s(I) = 0.0D0
           ENDIF
           WRITE (UL,11131) I,s(I)
11131      FORMAT(1X,'s(',I3,') = ',D15.8)
14      CONTINUE
        DO 15 I = 1, n
           IF (((x(I) .GE. 0.0D0) .AND. (x(I) .LE. 1.0D-9)) .OR.
     &          ((x(I) .LE.0.0D0) .AND. (x(I) .GE. (-1.0D-9)))) THEN
              x(I) = 0.0D0
           ENDIF
           WRITE (UL,11132) I,x(I)
11132      FORMAT(1X,'x('I3,') = ',D15.8)
15      CONTINUE
        WRITE (UL,*)
        WRITE (UL,36367) compteur
36367   FORMAT(1X,'Nombre de résolutions du système linéaire
     &  de solution dz =',I5)
        WRITE (UL,*)
C
        IF (.NOT. PASSAGE) THEN
           PASSAGE = .TRUE.
           WRITE (*,*) 'Désirez-vous que ces résultats soient écrits ',
     *                 'dans un fichier (T si oui) ?'
           READ  (*,11133) rep
11133      FORMAT(A1)
           IF ((rep .EQ. 'T') .OR. (rep .EQ. 't')) THEN
              UL = 2
              IF ((.NOT. FICHIER(1)) .AND. (.NOT. FICHIER(2))) THEN
                 WRITE (*,*) 'Nom du fichier dans lequel vous désirez ',
     *                       'stocker vos résultats (RESULTATSi ',
     *                       'conseillé) : '
                 READ  (*,11134) nomfich
11134            FORMAT(A10)
                 OPEN (UNIT = 2, FILE = nomfich, STATUS = 'NEW')
              ENDIF
              GOTO 2000
           ENDIF
        ENDIF
C
        CLOSE (UNIT = 2)
C
        RETURN
        END
```

```
C     5555555555555555555555555555555555555555555555555555555555555555555555555555
C
      SUBROUTINE RESOLVE (EPS,TETA,LBD1,NBitEXT,NBitINT)

C     ************************************************************************
C     *                                                                      *
C     *                                                                      *
C     *                Résolution de MIN  c(x)                               *
C     *                                   S.C. g(x)<=0                       *
C     *                                                                      *
C     *                                                                      *
C     ************************************************************************


C     1. DECLARATION DES VARIABLES
C     ****************************


C     1.1. SPECIFICATION DES ARGUMENTS DE LA SOUS-ROUTINE
C     ***************************************************

C     NBitEXT : (entier) SORTIE - compteur d'itérations externes
C     NBitINT : (entier) SORTIE - compteur d'itérations internes

C     IMPR    : (logique) (1),..., (8) ENTREE -
C                         ces variables = .TRUE. si l'utilisateur désire
C                         afficher/imprimer les résultats intermédiaires
C     EP      : (logique) (1) ENTREE, (2) ENTREE -
C                         (1) = .TRUE. si l'utilisateur désire que les
C                               résultats intermédiaires apparaissent
C                               à l'écran
C                         (2) = .TRUE. si l'utilisateur désire que les
C                               résultats intermédiaires soient stockés
C                               dans un fichier

C     BETA : (double précision) ENTREE -
C     MU   : (double précision) ENTREE et SORTIE -
C     EPS  : (double précision) ENTREE - paramètre de précision
C     TETA : (double précision) ENTREE - facteur de réduction de mu
C     LBD1 : (double précision) ENTREE -


      INTEGER NBitEXT,NBitINT
      DOUBLE PRECISION EPS,TETA,LBD1

C     1.2. DECLARATION DES VARIABLES LOCALES
C     **************************************

C     compteur : (entier) ENTREE et SORTIE - compteur du nombre de
C                résolutions du système linéaire dont la solution est dz


      INTEGER mmax, nmax, rmax
      PARAMETER (mmax = 100, nmax = 100, rmax = 2*mmax+nmax)

      DOUBLE PRECISION INFINI
      PARAMETER (INFINI = 1.0D18)

      INTEGER I
      DOUBLE PRECISION fchapz, relgap, relp, reld, valc,
     *                 maximumW, maximum, GfchaXdz, AbsDfcha,
     *                 ALPHA, NORMx, NORMy


      INTEGER m, n, r
      COMMON/DIM/m, n, r

      DOUBLE PRECISION y, s, x, z
      COMMON/VECTz/y(mmax), s(mmax), x(nmax), z(rmax)

      DOUBLE PRECISION dz
      COMMON/VECTdz/dz(rmax)

      LOGICAL IMPR, EP
      COMMON/LOGI/IMPR(8),EP(2)

      INTEGER UL
      COMMON/LU/UL

      DOUBLE PRECISION MU
      COMMON/DP/MU

      DOUBLE PRECISION BETA,yTs,nrmFp2,nrmFd2
      COMMON/BLOC/BETA,yTs,nrmFp2,nrmFd2

      INTEGER compteur
      COMMON/PLUS/compteur


C     2. IMPLEMENTATION
C     *****************


      compteur = 0
      NBitEXT  = 0
3000  NBitINT  = 0
      CALL EVALfcha (r,z,fchapz)

C      ------------------------------------------------------------------------
C     |                                                                        |
C     |                yTs                  ||Fp||                 ||Fd||     |
C     |    relgap = --------- ; relp = --------2- ; reld = --------2-          |
C     |             1 + |yTs|            1 + ||x||                1 + ||y||    |
C     |                                        2                       2       |
C      ------------------------------------------------------------------------


      relgap = yTs / (1.0D0 + DABS(yTs))
      NORMx = 0.0D0
      DO 96 I = 1, n
         IF (DABS(x(I)) .GE. INFINI) THEN
             NORMx = INFINI
             GOTO 17000
         ENDIF
96    CONTINUE
      NORMx  = DNRM2 (n,x,1)
17000 relp   = nrmFp2 / (1.0D0 + NORMx)
      NORMy = 0.0D0
      DO 97 I = 1, m
         IF (DABS(y(I)) .GE. INFINI) THEN
             NORMy = INFINI
             GOTO 20000
         ENDIF
97    CONTINUE
      NORMy  = DNRM2 (m,y,1)
20000 reld   = nrmFd2 / (1.0D0 + NORMy)
C
      maximumW = DMAX1 (relgap,relp)
      maximum  = reld
      IF (maximumW .GE. maximum) THEN
         maximum  = maximumW
      ENDIF
      IF (maximum .GE. EPS) THEN
```

```
            NBitEXT = NBitEXT + 1
            MU = (1.0D0 - TETA) * MU
C
C            ------------------------------------------------------
C           |                                                      |
C           |    Calcul de |Dfchap(z;dz)|, appelée AbsDfcha      |
C           |                                                      |
C            ------------------------------------------------------
C
4000        CALL DERIVDIR (GfchaXdz,AbsDfcha)
            IF ((AbsDfcha .GT. (BETA * MU)) .AND.
     &          (GfchaXdz .LT. 0.0D0)) THEN
               IF (NBitINT .EQ. 10) THEN
                   GOTO 3000
               ENDIF
               NBitINT = NBitINT + 1
C
C               ----------------------------------------
C              |                                        |
C              |    Recherche linéaire sur f chapeau    |
C              |    Solution : ALPHA                    |
C              |                                        |
C               ----------------------------------------
C
               CALL RECHLIN (TETA,LBD1,AbsDfcha,ALPHA)
               IF (ALPHA .EQ. 0.0D0) THEN
                   GOTO 3000
               ENDIF
C
C               ------------------------------------------------------
C              |                                                      |
C              |    Mise à jour des variables :                       |
C              |    ---------------------------                       |
C              |                                                      |
C              |    1) z = z + ALPHA dz    (c-à-d  y = y + ALPHA.dy   |
C              |                                   s = s + ALPHA.ds   |
C              |    2) (y,s,x) = z                 x = x + ALPHA.dx)  |
C              |                                                      |
C               ------------------------------------------------------
C
               CALL DAXPY (r,ALPHA,dz,1,z,1)
C
               DO 25 I = 1, m
                  y(I) = z(I)
                  s(I) = z(m+I)
25             CONTINUE
               DO 26 I = 1, n
                  x(I) = z(2*m+I)
26             CONTINUE
               GOTO 4000
            ENDIF
            GOTO 3000
         ENDIF
C
C
         RETURN
         END
```

```
C        6666666666666666666666666666666666666666666666666666666666666666666666666666666666
C
         SUBROUTINE EVALfcha (dim3,zW,fchapzW)
C
C        ************************************************************************
C        *                                                                      *
C        *                                                                      *
C        *  Calcul de f chapeau en zW :                                         *
C        *  ***************************                                         *
C        *                                                                      *
C        *                                        2             2             *
C        *  fchap (y,s,x) = yTs + (1/2) ||Fp|| + (1/2) ||Fd||                   *
C        *                                                                      *
C        *                  - MU . { SUM(i=1,m) ln[s(i).y(i)] }                 *
C        *                                                                      *
C        *                  - MUchapeau . { SUM(i=1,m) ln[gi(x)+s(i)] }         *
C        *                                                                      *
C        *               = fchap (zW)                                           *
C        *                                                                      *
C        *                                                                      *
C        ************************************************************************
C
C
C        1. DECLARATION DES VARIABLES
C        ****************************
C
C
C        1.1. SPECIFICATION DES ARGUMENTS DE LA SOUS-ROUTINE
C        ***************************************************
C
C        zW      : (double précision) ENTREE - variable de travail
C        fchapzW : (double précision) SORTIE - valeur de f chapeau évaluée en zW
C
C
         INTEGER mmax, nmax, rmax
         PARAMETER (mmax = 100, nmax = 100, rmax = 2*mmax+nmax)
C
         INTEGER dim3
         DOUBLE PRECISION zW(rmax),fchapzW
C
C
C        1.2. DECLARARTION DES VARIABLES LOCALES
C        ***************************************
C
C        IMPR    : (logique) (1),..., (8) ENTREE -
C                            ces variables = .TRUE. si l'utilisateur désire
C                            afficher/imprimer les résultats intermédiaires
C        EP      : (logique) (1) ENTREE, (2) ENTREE -
C                            (1) = .TRUE. si l'utilisateur désire que les
C                                  résultats intermédiaires apparaissent
C                                  à l'écran
C                            (2) = .TRUE. si l'utilisateur désire que les
C                                  résultats intermédiaires soient stockés
C                                  dans un fichier
C
C        BETA    : (double précision) ENTREE -
C        MU      : (double précision) ENTREE -
C        yTs     : (double précision) SORTIE - saut de dualité
C        nrmFp2  : (double précision) SORTIE - norme 2 de Fp
C        nrmFd2  : (double précision) SORTIE - norme 2 de Fd
C
C
         DOUBLE PRECISION INFINI
         PARAMETER (INFINI = 1.0D18)
C
         INTEGER I,dim1,dim2
         LOGICAL PASSAGE
         DOUBLE PRECISION y(mmax),s(mmax),x(nmax),FzW(rmax),Fp(mmax),nrmFp4,
```

```
     *                  Fd(nmax),nrmFd4,smlnsy,MUchap,gx(mmax),smlnFp
C
C
      INTEGER m, n, r
      COMMON/DIM/m, n, r
C
      LOGICAL IMPR, EP
      COMMON/LOGI/IMPR(8),EP(2)
C
      INTEGER UL
      COMMON/LU/UL
C
      DOUBLE PRECISION MU
      COMMON/DP/MU
C
      DOUBLE PRECISION BETA,yTs,nrmFp2,nrmFd2
      COMMON/BLOC/BETA,yTs,nrmFp2,nrmFd2
C
C
C
C     2. IMPLEMENTATION
C     *****************
C
C
C
C      ---------------------------------------------
C     |                                             |
C     |  (y,s,x) = zW   pour facilités d'écriture   |
C     |                                             |
C      ---------------------------------------------
C
      DO 30 I = 1, m
         y(I) = zW(I)
         s(I) = zW(m+I)
30    CONTINUE
      DO 31 I = 1, n
         x(I) = zW(2*m+I)
31    CONTINUE
C
C      -------------------
C     |                   |
C     |   Calcul de yTs   |
C     |                   |
C      -------------------
C
      yTs = 0.0D0
      DO 89 I = 1, m
         IF (DABS(y(I)) .GE. INFINI) THEN
            IF ((((y(I) .GE. INFINI) .AND. (s(I) .GT. 0.0D0)) .OR.
     &           ((y(I) .LE. (-INFINI)) .AND. (s(I) .LT. 0.0D0))) THEN
               yTs = yTs + INFINI
            ELSE
               yTs = yTs - INFINI
            ENDIF
         ELSE
           IF (((y(I) .GT. 0.0D0) .AND. (s(I) .GE. INFINI)) .OR.
     &         ((y(I) .LT. 0.0D0) .AND. (s(I) .LE. (-INFINI)))) THEN
              yTs = yTs + INFINI
           ELSE
              IF (DABS(s(I)) .GE. INFINI) THEN
                 yTs = yTs - INFINI
              ELSE
                 yTs = yTs + (y(I) * s(I))
                 IF (yTs .GE. INFINI) THEN
                    yTs = INFINI
                 ELSE
                    IF (yTs .LE. (-INFINI)) THEN
                       yTs = (-INFINI)
                    ENDIF
                 ENDIF
              ENDIF
           ENDIF
         ENDIF
      ENDIF
89    CONTINUE
C
C      ------------------------------------------------------------
C     |                  2                                    2    |
C     |     nrmFp4 = ||Fp||       et       nrmFd4 = ||Fd||       |
C     |                  2                                    2    |
C      ------------------------------------------------------------
C
      CALL EVALF (r,r,zW,FzW)
C                                                  Fp = FzW(m+1,2m)
      DO 32 I = 1, m
         Fp(I) = FzW(m+I)
         IF (DABS(Fp(I)) .GE. INFINI) THEN
            nrmFp2 = INFINI
            nrmFp4 = INFINI
            GOTO 18000
         ENDIF
32    CONTINUE
      nrmFp2 = DNRM2 (m,Fp,1)
C
      IF (nrmFp2 .GE. INFINI) THEN
         nrmFp4 = INFINI
         GOTO 18000
      ENDIF
      nrmFp4 = (nrmFp2 ** 2)
C                                                  Fd = FzW(2m+1,2m+n)
18000 DO 33 I = 1, n
         Fd(I) = FzW(2*m+I)
         IF (DABS(Fd(I)) .GE. INFINI) THEN
            nrmFd2 = INFINI
            nrmFd4 = INFINI
            GOTO 19000
         ENDIF
33    CONTINUE
      nrmFd2 = DNRM2 (n,Fd,1)
      IF (nrmFd2 .GE. INFINI) THEN
         nrmFd4 = INFINI
         GOTO 19000
      ENDIF
      nrmFd4 = (nrmFd2 ** 2)
C
C      ------------------------------------------
C     |                                          |
C     |   smlnsy = { SUM(i=1,m) ln[s(i).y(i)] }  |
C     |                                          |
C      ------------------------------------------
C
19000 smlnsy = 0.0D0
      DO 34 I = 1, m
         smlnsy = smlnsy + DLOG (s(I) * y(I))
34    CONTINUE
C
C      -------------------------
C     |                         |
C     |   Calcul de MUchapeau   |
C     |                         |
C      -------------------------
C
      MUchap = ((BETA * MU)/DBLE(m))
C
C      ------------------------------------------
C     |                                          |
C     |   smlnFp = { SUM(i=1,m) ln[gi(x)+s(i)] } |
C
```

```
C       |                                                |
C        ------------------------------------------------
C
        smlnFp = 0.0D0
        CALL EVALg (m,n,x,gx)
        DO 37 I = 1, m
           IF ((gx(I) + s(I)) .LE. 0.0D0) THEN
               smlnFp = smlnFp - INFINI
           ELSE
               smlnFp = smlnFp + DLOG (gx(I) + s(I))
           ENDIF
37      CONTINUE
C
C        ----------------------------
C       |                            |
C       |   Calcul de fchapeau(zW)   |
C       |                            |
C        ----------------------------
C
        fchapzW = yTs + (DBLE (0.5) * nrmFp4) + (DBLE (0.5) * nrmFd4)
        fchapzW = fchapzW - (MU * smlnsy) - (MUchap * smlnFp)
        IF (fchapzW .GE. INFINI) THEN
            fchapzW = INFINI
        ELSE
            IF (fchapzW .LE. (-INFINI)) THEN
                fchapzW = (-INFINI)
            ENDIF
        ENDIF
C
C        --------------------------------
C       |                                |
C       |   Impression de fchapeau(zW)   |
C       |                                |
C        --------------------------------
C
        PASSAGE = .FALSE.
        IF (EP(1) .EQ. .TRUE.) THEN
           UL = 5
        ELSE
           IF (EP(2) .EQ. .TRUE.) THEN
              UL = 2
              PASSAGE = .TRUE.
           ENDIF
        ENDIF
C
5000    IF (IMPR(1) .EQ. .TRUE.) THEN
           WRITE (UL,11136) fchapzW
11136      FORMAT (1X,'fchap(zW) = ',D23.16)
        ENDIF
C
        IF ((EP(2) .EQ. .TRUE.) .AND. (PASSAGE .EQ. .FALSE.)) THEN
           UL = 2
           PASSAGE = .TRUE.
           GOTO 5000
        ENDIF
C
C
        RETURN
        END
```

```
C       77777777777777777777777777777777777777777777777777777777777777777777777777777777
C
        SUBROUTINE DERIVDIR (GfchaXdz,AbsDfcha)
C
C       ************************************************************
C       *                                                          *
C       *                                                          *
C       *       Calcul de la valeur absolue de la dérivée          *
C       *       directionnelle de f chapeau, notée AbsDfcha        *
C       *                                                          *
C       *                                                          *
C       ************************************************************
C
C
C       1. DECLARATION DES VARIABLES
C       ****************************
C
C
C       1.1. SPECIFICATION DES ARGUMENTS DE LA SOUS-ROUTINE
C       ***************************************************
C
C       IMPR    : (logique) (1),..., (8) ENTREE -
C                           ces variables = .TRUE. si l'utilisateur désire
C                           afficher/imprimer les résultats intermédiaires
C       EP      : (logique) (1) ENTREE, (2) ENTREE -
C                           (1) = .TRUE. si l'utilisateur désire que les
C                                 résultats intermédiaires apparaissent
C                                 à l'écran
C                           (2) = .TRUE. si l'utilisateur désire que les
C                                 résultats intermédiaires soient stockés
C                                 dans un fichier
C       MU       : (double précision) ENTREE -
C       AbsDfcha : (double précision) SORTIE - valeur de | Dfchap (z;dz)|
C       compteur : (entier) ENTREE et SORTIE - compteur du nombre de
C                  résolutions du système linéaire dont la solution est dz
C
C
        DOUBLE PRECISION GfchaXdz, AbsDfcha
C
C
C       1.2. DECLARATION DES VARIABLES LOCALES
C       **************************************
C
C
C
        INTEGER mmax, nmax, rmax
        PARAMETER (mmax = 100, nmax = 100, rmax = 2*mmax+nmax)
C
        DOUBLE PRECISION INFINI
        PARAMETER (INFINI = 1.0D18)
C
        INTEGER I,J,K,dim1,dim2
        CHARACTER*1 stopECRAN
        LOGICAL PASSAGE
        DOUBLE PRECISION Fz(rmax),EPSFCN,zSCALE(rmax),FJAC(rmax,rmax),
     *                   gJACx(mmax,nmax),TgJACx(nmax,mmax),
     *                   gx(mmax),BLOC33(nmax,nmax),JgJg(mmax,mmax),
     *                   INTER(mmax),
     *                   INTER2(nmax),HESgENx(nmax,nmax,mmax),
     *                   HEScENx(nmax,nmax),INTER3(nmax,nmax),
     *                   INTER4(nmax),Gcx(nmax),INT(mmax),MUchap,
     *                   MFz(rmax),fchapz,Gfchap(rmax)
C
C
        INTEGER m, n, r
        COMMON/DIM/m, n, r
C
        DOUBLE PRECISION y, s, x, z
```

```
      COMMON/VECTz/y(mmax), s(mmax), x(nmax), z(rmax)
C
      DOUBLE PRECISION dz
      COMMON/VECTdz/dz(rmax)
C
      LOGICAL IMPR, EP
      COMMON/LOGI/IMPR(8),EP(2)
C
      INTEGER UL
      COMMON/LU/UL
C
      DOUBLE PRECISION MU
      COMMON/DP/MU
C
      DOUBLE PRECISION BETA,yTs,nrmFp2,nrmFd2
      COMMON/BLOC/BETA,yTs,nrmFp2,nrmFd2
C
      INTEGER compteur
      COMMON/PLUS/compteur
C
C
C     2. IMPLEMENTATION
C     *****************
C
C
C     2.1. CALCUL DE LA DIRECTION dz
C     ******************************
C
C
C     ------------------------------------------------------------
C    |                                                            |
C    |   Calcul de la matrice Jacobienne de F, évaluée en z       |
C    |                                                            |
C     ------------------------------------------------------------
C
      DO 112 I = 1, r
         DO 113 J = 1, r
            FJAC(I,J) = 0.0D0
113      CONTINUE
112   CONTINUE
C
      DO 114 I = 1, m
         FJAC(I,I) = s(I)
114   CONTINUE
C
      DO 115 I = 1, m
         FJAC(I,m+I) = y(I)
115   CONTINUE
C
      DO 116 I = 1, m
         FJAC(m+I,m+I) = 1.0D0
116   CONTINUE
C
      CALL MATJACg (m,n,x,gJACx)
      DO 117 J = 1, n
         DO 118 I = 1, m
            DO 150 K = 1, m
               FJAC(m+I,2*m+J) = gJACx(I,J)
               TgJACx(J,I)     = gJACx(I,J)
150         CONTINUE
118      CONTINUE
117   CONTINUE
C
      DO 130 J = 1, m
         DO 119 I = 1, n
            FJAC(2*m+I,J) = gJACx(J,I)
119      CONTINUE
130   CONTINUE
C
      CALL HESSIENg (m,n,x,HESgENx)
      CALL HESSIENc (n,x,HEScENx)
      DO 131 I = 1, n
         DO 120 J = 1, n
            BLOC33(I,J) = HEScENx(I,J)
120      CONTINUE
131   CONTINUE
      DO 121 I = 1, m
         DO 122 J = 1, n
            DO 123 K = 1, n
               BLOC33(J,K) = BLOC33(J,K) + y(I) * HESgENx(J,K,m)
123         CONTINUE
122      CONTINUE
121   CONTINUE
      DO 124 I = 1, n
         DO 125 J = 1, n
            FJAC(2*m+I,2*m+J) = BLOC33(I,J)
125      CONTINUE
124   CONTINUE
C
C     ------------------------------------------------------------
C    |                                                            |
C    |   Impression de la matrice Jacobienne de F                 |
C    |                                                            |
C     ------------------------------------------------------------
C
      PASSAGE = .FALSE.
      IF (EP(1) .EQ. .TRUE.) THEN
         UL = 5
      ELSE
         IF (EP(2) .EQ. .TRUE.) THEN
             UL = 2
             PASSAGE = .TRUE.
         ENDIF
      ENDIF
C
6000  IF (IMPR(2) .EQ. .TRUE.) THEN
         DO 1127 I = 1, r
            DO 1128 J = 1, r
               WRITE (UL,1129) I, J, FJAC(I,J)
1129           FORMAT(1X,'FJAC(',I3,',',I3,') : ',D15.8)
1128        CONTINUE
1127     CONTINUE
      ENDIF
C
      IF ((EP(2) .EQ. .TRUE.) .AND. (PASSAGE .EQ. .FALSE.)) THEN
         UL = 2
         PASSAGE = .TRUE.
         GOTO 6000
      ENDIF
C
C     ------------------------------------------------------------
C    |                                                            |
C    |   Résolution du système  FJAC * dz = -Fz                   |
C    |   Solution : dz                                            |
C    |                                                            |
C     ------------------------------------------------------------
C
      CALL EVALF (r,r,z,Fz)
C                                                  MFz(r) = - Fz(r)
      DO 38 I = 1, r
         MFz(I) = - Fz(I)
38    CONTINUE
      compteur = compteur + 1
C                                                  RESOLUTION DU SYSTEME
C                                                  SOLUTION : dz
```

```
      CALL DLSARG (r,FJAC,rmax,MFz,1,dz)
C
C      ------------------------------------
C      |                                  |
C      |   Impression de la direction dz  |
C      |                                  |
C      ------------------------------------
C
      PASSAGE = .FALSE.
      IF (EP(1) .EQ. .TRUE.) THEN
         UL = 5
      ELSE
         IF (EP(2) .EQ. .TRUE.) THEN
            UL = 2
            PASSAGE = .TRUE.
         ENDIF
      ENDIF
C
7000  IF (IMPR(3) .EQ. .TRUE.) THEN
         DO 127 I = 1, m
            WRITE (UL,1131) I, dz(I)
1131        FORMAT(1X,'dy(',I3,') = ',D15.8)
127      CONTINUE
         DO 128 I = 1, m
            WRITE (UL,1132) I, dz(m+I)
1132        FORMAT(1X,'ds(',I3,') = ',D15.8)
128      CONTINUE
         DO 129 I = 1, n
            WRITE (UL,1133) I, dz(2*m+I)
1133        FORMAT(1X,'dx(',I3,') = ',D15.8)
129      CONTINUE
      ENDIF
C
      IF ((EP(2) .EQ. .TRUE.) .AND. (PASSAGE .EQ. .FALSE.)) THEN
         UL = 2
         PASSAGE = .TRUE.
         GOTO 7000
      ENDIF
C
C
C     2.2. CALCUL DE | Dfchap (z;dz)|
C     ********************************
C
C
C      ------------------------------------------------
C      |                                              |
C      |   Calcul du gradient de f chapeau, évalué en z   |
C      |                                              |
C      ------------------------------------------------
C
      CALL GRADc (n,x,Gcx)
      CALL EVALg (m,n,x,gx)
C
C     **** DERIVEE DE f chapeau PAR RAPPORT A y ****
C
C                                          JgJg = Jg(x).(Jg(x))T
      CALL DMXTXF (n,m,TgJACx,nmax,m,JgJg,mmax)
C                                          INTER = JgJg.y
      CALL DMURRV (m,m,JgJg,mmax,m,y,1,m,INTER)
C                                          Gfchap(1...m) = INTER + s
      DO 39 I = 1, m
         Gfchap(I) = s(I) + INTER(I)
39    CONTINUE
C                                          INTER = Jg(x).Gc(x)
      CALL DMURRV (m,n,gJACx,m,n,Gcx,1,m,INTER)
C                                          INIT Gfchap(1...m)
      DO 40 I = 1, m
         IF (DABS(y(I)) .LE. 1.D-6) THEN
            Gfchap(I) = Gfchap(I) + INTER(I) - INFINI
         ELSE
            Gfchap(I) = Gfchap(I) + INTER(I) - (MU/y(I))
         ENDIF
         IF (Gfchap(I) .GE. INFINI) THEN
            Gfchap(I) = INFINI
         ENDIF
         IF (Gfchap(I) .LE. (- INFINI)) THEN
            Gfchap(I) = - INFINI
         ENDIF
40    CONTINUE
C
C     **** DERIVEE DE f chapeau PAR RAPPORT A s ****
C
      MUchap = ((BETA * MU)/DBLE(m))
      DO 41 I = 1, m
         IF (DABS(s(I)) .LE. 1.D-6) THEN
            Gfchap(m+I) = y(I) + gx(I) + s(I) - INFINI
         ELSE
            Gfchap(m+I) = y(I) + gx(I) + s(I) - (MU/s(I))
         ENDIF
         IF (Gfchap(m+I) .GE. INFINI) THEN
            Gfchap(m+I) = INFINI
         ENDIF
         IF (Gfchap(m+I) .LE. (- INFINI)) THEN
            Gfchap(m+I) = - INFINI
         ENDIF
         IF (DABS(gx(I) + s(I)) .LE. 1.D-6) THEN
            Gfchap(m+I) = Gfchap(m+I) - INFINI
         ELSE
            Gfchap(m+I) = Gfchap(m+I) - (MUchap/(gx(I) + s(I)))
         ENDIF
         IF (Gfchap(m+I) .GE. INFINI) THEN
            Gfchap(m+I) = INFINI
         ENDIF
         IF (Gfchap(m+I) .LE. (- INFINI)) THEN
            Gfchap(m+I) = - INFINI
         ENDIF
41    CONTINUE
C
C     **** DERIVEE DE f chapeau PAR RAPPORT A x ****
C
C                                          INTER2 = (Jg(x))T.g(x)
      CALL DMURRV (n,m,TgJACx,nmax,m,gx,1,n,INTER2)
C                                          Gfchap(2m...2m+n) = INTER2
      DO 42 I = 1, n
         Gfchap(2*m+I) = INTER2(I)
42    CONTINUE
C                                          INTER2 = (Jg(x))T.s
      CALL DMURRV (n,m,TgJACx,nmax,m,s,1,n,INTER2)
C                                          Gfchap = Gfchap + INTER2
      DO 43 I = 1, n
         Gfchap(2*m+I) = Gfchap(2*m+I) + INTER2(I)
43    CONTINUE
C                                          INTER2 = (Jg(x))T / (g(x) + s)
      DO 44 I = 1, m
         IF (DABS(gx(I) + s(I)) .LE. 1.D-6) THEN
            INTER(I) = INFINI
         ELSE
            INTER(I) = 1.0D0/(gx(I) + s(I))
         ENDIF
         IF (INTER(I) .GE. INFINI) THEN
            INTER(I) = INFINI
         ENDIF
         IF (INTER(I) .LE. (- INFINI)) THEN
            INTER(I) = - INFINI
         ENDIF
44    CONTINUE
```

```
      CALL DMURRV (n,m,TgJACx,nmax,m,INTER,1,n,INTER2)
C                                          Gfchap = Gfchap - MUchap*INTER2
      DO 45 I = 1, n
         Gfchap(2*m+I) = Gfchap(2*m+I) - (MUchap * INTER2(I))
45    CONTINUE
C                                          INTER3 = SUM (yi.Hg(x)) + Hc(x)
      DO 46 I = 1, n
         DO 47 j = 1, n
            INTER3(I,J) = 0.0D0
47       CONTINUE
46    CONTINUE
      DO 48 I = 1, n
         DO 49 J = 1, n
            DO 50 K = 1, m
               INTER3(I,J) = (INTER3(I,J) + (HESgENx(I,J,K) * y(K)))
50          CONTINUE
            INTER3(I,J) = (INTER3(I,J) + HEScENx(I,J))
49       CONTINUE
48    CONTINUE
C                                          INTER2 = (Jg(x))T.y + Gc(x)
      CALL DMURRV (n,m,TgJACx,nmax,m,y,1,n,INTER2)
      DO 51 I = 1, n
         INTER2(I) = (INTER2(I) + Gcx(I))
51    CONTINUE
C                                          INTER4 = INTER3 * INTER2
      CALL DMURRV (n,n,INTER3,nmax,n,INTER2,1,n,INTER4)
C                                          mise à jour de Gfchap
      DO 52 I = 1, n
         Gfchap(2*m+I) = (Gfchap(2*m+I) + INTER4(I))
52    CONTINUE
C
C
C      ----------------------------------------
C     |                                        |
C     |   Impression du gradient de f chapeau  |
C     |                                        |
C      ----------------------------------------
C
      PASSAGE = .FALSE.
      IF (EP(1) .EQ. .TRUE.) THEN
         UL = 5
      ELSE
         IF (EP(2) .EQ. .TRUE.) THEN
             UL = 2
             PASSAGE = .TRUE.
         ENDIF
      ENDIF
C
8000  IF (IMPR(4) .EQ. .TRUE.) THEN
         DO 53 I = 1,r
            WRITE (UL,1134) I, Gfchap(I)
1134        FORMAT(1X,'Gfchap(',I3,') =',D18.11)
53       CONTINUE
      ENDIF
C
      IF ((EP(2) .EQ. .TRUE.) .AND. (PASSAGE .EQ. .FALSE.)) THEN
         UL = 2
         PASSAGE = .TRUE.
         GOTO 8000
      ENDIF
C
C      --------------------------------
C     |                                |
C     |   Calcul de | Dfchap (z;dz)|   |
C     |                                |
C      --------------------------------
C
C                                          GfchaXdz = Gfchap.dz
      GfchaXdz = DDOT (r,Gfchap,1,dz,1)
C                                                  AbsDfcha = | Dfchap (z;dz)|
      AbsDfcha = DABS (GfchaXdz)
C
C
      RETURN
      END
```

```
C       8888888888888888888888888888888888888888888888888888888888888888888888888888888888888888
C
        SUBROUTINE RECHLIN (TETA,LBD1,AbsDfcha,ALPHA)
C
C       ***********************************************************************
C       *                                                                     *
C       *                                                                     *
C       *                 Recherche linéaire sur f chapeau                    *
C       *                 le long de la direction dz                          *
C       *                                                                     *
C       *                                                                     *
C       ***********************************************************************
C
C
C       1. DECLARATION DES VARIABLES
C       ****************************
C
C
C       1.1. SPECIFICATION DES ARGUMENTS DE LA SOUS-ROUTINE
C       ***************************************************
C
C       IMPR    : (logique) (1),..., (8) ENTREE -
C                           ces variables = .TRUE. si l'utilisateur désire
C                           afficher/imprimer les résultats intermédiaires
C       EP      : (logique) (1) ENTREE, (2) ENTREE -
C                           (1) = .TRUE. si l'utilisateur désire que les
C                                 résultats intermédiaires apparaissent
C                                 à l'écran
C                           (2) = .TRUE. si l'utilisateur désire que les
C                                 résultats intermédiaires soient stockés
C                                 dans un fichier
C       TETA    : (double précision) ENTREE - facteur de réduction de mu
C       LBD1    : (double précision) ENTREE -
C       AbsDfcha : (double précision) ENTREE - valeur de | Dfchap (z;dz)|
C       ALPHA   : (double précision) SORTIE -
C
C
        INTEGER mmax, nmax, rmax
        PARAMETER (mmax = 100, nmax = 100, rmax = 2*mmax+nmax)
C
        DOUBLE PRECISION TETA,LBD1,ALPHA
C
C
C       1.2. DECLARATION DES VARIABLES LOCALES
C       **************************************
C
C
C
        DOUBLE PRECISION INFINI
        PARAMETER (INFINI = 1.0D18)
C
        INTEGER I,p,icomptor
        LOGICAL PASSAGE
        DOUBLE PRECISION ALPHAmax,dy(mmax),ds(mmax),zMzDIR,
     *                   fchapz,zDIR(rmax),fchazDIR,AbsDfcha,morceau
C
C
        INTEGER m, n, r
        COMMON/DIM/m, n, r
C
        DOUBLE PRECISION y, s, x, z
        COMMON/VECTz/y(mmax), s(mmax), x(nmax), z(rmax)
C
        DOUBLE PRECISION dz
        COMMON/VECTdz/dz(rmax)
C
        LOGICAL IMPR, EP
        COMMON/LOGI/IMPR(8),EP(2)
C
        INTEGER UL
        COMMON/LU/UL
C
        DOUBLE PRECISION MU
        COMMON/DP/MU
C
        DOUBLE PRECISION BETA,yTs,nrmFp2,nrmFd2
        COMMON/BLOC/BETA,yTs,nrmFp2,nrmFd2
C
C
C
C       2. IMPLEMENTATION
C       *****************
C
C
C       2.1. RECHERCHE DE ALPHAmax
C       **************************
C
C
C        ------------------------------------------------------------------
C       |                                                                  |
C       |   ALPHAmax = MAX { ALPHA tq (y + ALPHA dy, s + ALPHA ds) >= (0,0) }  |
C       |                                                                  |
C        ------------------------------------------------------------------
C
        ALPHAmax = INFINI
        DO 54 I = 1, m
           dy(I) = dz(I)
           ds(I) = dz(m+I)
54      CONTINUE
        DO 55 I = 1, m
           IF (dy(I) .LT. 0.0D0) THEN
              ALPHAmax = DMIN1 (ALPHAmax,(-y(I)/dy(I)))
           ENDIF
           IF (ds(I) .LT. 0.0D0) THEN
              ALPHAmax = DMIN1 (ALPHAmax,(-s(I)/ds(I)))
           ENDIF
55      CONTINUE
C
C
C       2.2. RECHERCHE DE p
C       *******************
C
C
C        ------------------------------------------------------------------
C       |                                                                  |
C       |   p = le plus petit entier strictement positif tel que           |
C       |       la condition (10) soit réalisée, c-à-d tel que             |
C       |                                                                  |
C       |                                                                  |
C       |                                    p                             |
C       |       [ fchap(z) - fchap(z + (TETA).ALPHAmax.dz) ]  >=           |
C       |                                                                  |
C       |                                        p                         |
C       |                          [ LAMBDA1.(TETA).ALPHAmax.|Dfchap(z;dz)| ]  |
C       |                                                                  |
C        ------------------------------------------------------------------
C
        p = 1
        CALL EVALfcha (r,z,fchapz)
9000    DO 57 I = 1, r
           zDIR(I) = z(I) + ((TETA ** p) * ALPHAmax * dz(I))
57      CONTINUE
        CALL EVALfcha (r,zDIR,fchazDIR)
        morceau = (LBD1 * (TETA ** p) * ALPHAmax * AbsDfcha)
        IF (morceau .GE. INFINI) THEN
            morceau = INFINI
```

```
        ELSE
            IF (morceau .LE. (-INFINI)) THEN
                morceau = (-INFINI)
            ENDIF
        ENDIF
        zMzDIR = fchapz - fchazDIR
        IF (zMzDIR .GE. INFINI) THEN
            zMzDIR = INFINI
        ELSE
            IF (zMzDIR .LE. (-INFINI)) THEN
                zMzDIR = (- INFINI)
            ENDIF
        ENDIF
C
        IF (zMzDIR .LT. morceau) THEN
           IF (p .LT. 70) THEN
               p = p + 1
               GOTO 9000
           ELSE
               ALPHA = 0.0D0
               GOTO 25000
           ENDIF
        ENDIF
C
CCC     2.3. CALCUL DE ALPHA
CCC     ********************
C
        ALPHA = ((TETA ** p) * ALPHAmax)
C
CCC      ---------------------------
CCC     |                           |
CCC     |    Impression de ALPHA    |
CCC     |                           |
CCC      ---------------------------
C
        PASSAGE = .FALSE.
        IF (EP(1) .EQ. .TRUE.) THEN
           UL = 5
        ELSE
            IF (EP(2) .EQ. .TRUE.) THEN
                UL = 2
                PASSAGE = .TRUE.
           ENDIF
        ENDIF
C
10000   IF (IMPR(5) .EQ. .TRUE.) THEN
           WRITE (UL,11137) ALPHA
11137      FORMAT(1X,'ALPHA',D15.8)
        ENDIF
C
        IF ((EP(2) .EQ. .TRUE.) .AND. (PASSAGE .EQ. .FALSE.)) THEN
           UL = 2
           PASSAGE = .TRUE.
           GOTO 10000
        ENDIF
C
C
25000   RETURN
        END
```

```
C       99999999999999999999999999999999999999999999999999999999999999999999999999999999
C
        SUBROUTINE EVALF (dim1,dim2,zWW,F)
C
C       ********************************************************************
C       *                                                                  *
C       *                                                                  *
C       *                    ( Fc )   (           Y.s - MU.e          )   *
C       *                                                                  *
C       *   Calcul de F(z) = ( Fp ) = (           g(x) + s            )   *
C       *                                                                  *
C       *                    ( Fd )   ( trans(Jg(x)).y + Gc(x)        )   *
C       *                                                                  *
C       *                                                                  *
C       ********************************************************************
C
C
C       1. DECLARATION DES VARIABLES
C       ****************************
C
C
C       1.1. SPECIFICATION DES ARGUMENTS DE LA SOUS-ROUTINE
C       ***************************************************
C
C       IMPR    : (logique) (1),..., (8) ENTREE -
C                           ces variables = .TRUE. si l'utilisateur désire
C                           afficher/imprimer les résultats intermédiaires
C       EP      : (logique) (1) ENTREE, (2) ENTREE -
C                           (1) = .TRUE. si l'utilisateur désire que les
C                                 résultats intermédiaires apparaissent
C                                 à l'écran
C                           (2) = .TRUE. si l'utilisateur désire que les
C                                 résultats intermédiaires soient stockés
C                                 dans un fichier
C       MU   : (double précision) ENTREE -
C       zWW  : (double précision) ENTREE - vecteur en lequel F sera évaluée
C       F    : (double précision) SORTIE - valeur de F(zWW)
C
C
        INTEGER mmax, nmax, rmax
        PARAMETER (mmax = 100, nmax = 100, rmax = 2*mmax+nmax)
C
        INTEGER dim1,dim2
        DOUBLE PRECISION zWW(rmax),F(rmax)
C
C
C       1.2. DECLARATION DES VARIABLES LOCALES
C       **************************************
C
C
C
        DOUBLE PRECISION INFINI
        PARAMETER (INFINI = 1.0D18)
C
        INTEGER I,J
        LOGICAL PASSAGE
        DOUBLE PRECISION y(mmax),s(mmax),x(nmax),Fc(mmax),gx(mmax),
     *                   Fp(mmax),xSCALE(nmax),EPSFCN,gJAC(mmax,nmax),
     *                   TgJAC(nmax,mmax),Fd(nmax),Gc(nmax),cx
C
C
        INTEGER m, n, r
        COMMON/DIM/m, n, r
C
        LOGICAL IMPR, EP
        COMMON/LOGI/IMPR(8),EP(2)
C
        INTEGER UL
```

```
      COMMON/LU/UL
C
      DOUBLE PRECISION MU
      COMMON/DP/MU
C
C
C
C     2. IMPLEMENTATION
C     *****************
C
C
      dim1 = r
      dim2 = r
C
C      ------------------------------------------------
C     |                                                |
C     |   (y,s,x) = zWW   pour facilités d'écriture    |
C     |                                                |
C      ------------------------------------------------
C
      DO 58 I = 1, m
         y(I) = zWW(I)
         s(I) = zWW(m+I)
58    CONTINUE
      DO 59 I = 1, n
         x(I) = zWW(2*m+I)
59    CONTINUE
C
C
C     2.1. Fc = Y.s - MU.e
C     ********************
C
C                                              Fc(i) = y(i).s(i)
      DO 90 I = 1, m
         Fc(I) = 0.0D0
90    CONTINUE
      DO 91 I = 1, m
         IF (DABS(y(I)) .GE. INFINI) THEN
            IF (((y(I) .GE. INFINI) .AND. (s(I) .GT. 0.0D0)) .OR.
     &          ((y(I) .LE. (-INFINI)) .AND. (s(I) .LT. 0.0D0))) THEN
               Fc(I) = INFINI
            ELSE
               Fc(I) = (- INFINI)
            ENDIF
         ELSE
           IF (((y(I) .GT. 0.0D0) .AND. (s(I) .GE. INFINI)) .OR.
     &         ((y(I) .LT. 0.0D0) .AND. (s(I) .LE. (-INFINI)))) THEN
              Fc(I) = INFINI
           ELSE
               IF (DABS(s(I)) .GE. INFINI) THEN
                   Fc(I) = (- INFINI)
               ELSE
                   Fc(I) = (y(I) * s(I))
                   IF (Fc(I) .GE. INFINI) THEN
                      Fc(I) = INFINI
                   ELSE
                       IF (Fc(I) .LE. (-INFINI)) THEN
                          Fc(I) = (-INFINI)
                       ENDIF
                   ENDIF
               ENDIF
           ENDIF
         ENDIF
91    CONTINUE
C                                              Fc(i) = Fc(i) - MU
      CALL DADD (m,-MU,Fc,1)
C
C
```

```
C     2.2. Fp = g(x) + s
C     ******************
C
      CALL EVALg (m,n,x,gx)
C                                              Fp(i) = gx(i), c-à-d:
C                                              Fp(i) = gi(x)
      CALL DCOPY (m,gx,1,Fp,1)
C                                              Fp(i) = gx(i) + s(i)
      CALL DAXPY (m,1.0D0,s,1,Fp,1)
      DO 65 I = 1, m
         IF (Fp(I) .GE. INFINI) THEN
             Fp(I) = INFINI
         ELSE
             IF (Fp(I) .LE. (-INFINI)) THEN
                 Fp(I) = (-INFINI)
             ENDIF
         ENDIF
65    CONTINUE
C
C
C     2.3. Fd = trans(Jg(x)).y + trans(Gc(x))
C     ***************************************
C
C
C      ------------------------------------------------------
C     |                                                      |
C     |   Calcul de la matrice Jacobienne de g, évaluée en x |
C     |                                                      |
C      ------------------------------------------------------
C
      CALL MATJACg (m,n,x,gJAC)
C
C      ----------------------------------------------
C     |                                              |
C     |   Impression de la matrice Jacobienne de g   |
C     |                                              |
C      ----------------------------------------------
C
      PASSAGE = .FALSE.
      IF (EP(1) .EQ. .TRUE.) THEN
        UL = 5
      ELSE
         IF (EP(2) .EQ. .TRUE.) THEN
             UL = 2
             PASSAGE = .TRUE.
         ENDIF
      ENDIF
C
11000 IF (IMPR(6) .EQ. .TRUE.) THEN
         DO 67 I = 1, m
            DO 68 J = 1, n
               WRITE (UL,1146) I, J, gJAC(I,J)
1146           FORMAT(1X,'gJAC(',I3,',',I3,') : ',D15.8)
68          CONTINUE
67       CONTINUE
      ENDIF
C
      IF ((EP(2) .EQ. .TRUE.) .AND. (PASSAGE .EQ. .FALSE.)) THEN
         UL = 2
         PASSAGE = .TRUE.
         GOTO 11000
      ENDIF
C
C      -------------------------
C     |                         |
C     |   Fd = trans(Jg(x)).y   |
C     |                         |
C      -------------------------
```

```
C
C                                                  TgJAC = trans(Jg(x))
C
      DO 69 I = 1, m
         DO 70 J = 1, n
            TgJAC(J,I) = gJAC(I,J)
70       CONTINUE
69    CONTINUE
C                                                  Fd = trans(Jg(x)).y
      DO 92 I = 1, n
         Fd(I) = 0.0D0
92    CONTINUE
      DO 93 I = 1, n
         DO 94 J = 1, m
            IF (DABS(TgJAC(I,J)) .GE. INFINI) THEN
             IF(((TgJAC(I,J) .GE. INFINI) .AND. (y(J) .GT. 0.0D0)).OR.
     &       ((TgJAC(I,J) .LE. (-INFINI)) .AND. (y(J) .LT. 0.0D0)))THEN
                  Fd(I) = Fd(I) + INFINI
               ELSE
                  Fd(I) = Fd(I) - INFINI
               ENDIF
            ELSE
             IF(((TgJAC(I,J) .GT. 0.0D0) .AND. (y(J) .GE. INFINI)).OR.
     &       ((TgJAC(I,J) .LT. 0.0D0) .AND. (y(J) .LE. (-INFINI))))THEN
                  Fd(I) = Fd(I) + INFINI
              ELSE
                  IF (DABS(y(J)) .GE. INFINI) THEN
                      Fd(I) = Fd(I) - INFINI
                  ELSE
                      Fd(I) = Fd(I) + (TgJAC(I,J) * y(J))
                      IF (Fd(I) .GE. INFINI) THEN
                          Fd(I) = INFINI
                      ELSE
                          IF (Fd(I) .LE. (-INFINI)) THEN
                              Fd(I) = (-INFINI)
                          ENDIF
                       ENDIF
                      ENDIF
               ENDIF
            ENDIF
94          CONTINUE
93    CONTINUE
C
C     ----------------------------------
C    |                                  |
C    |   Impression de trans(Jg(x)).y   |
C    |                                  |
C     ----------------------------------
C
      PASSAGE = .FALSE.
      IF (EP(1) .EQ. .TRUE.) THEN
         UL = 5
      ELSE
          IF (EP(2) .EQ. .TRUE.) THEN
             UL = 2
             PASSAGE = .TRUE.
         ENDIF
      ENDIF
C
12000 IF (IMPR(7) .EQ. .TRUE.) THEN
         DO 73 I = 1, n
            WRITE (UL,1147) I, Fd(I)
1147        FORMAT(1X,'[trans(Jg(x)).y](',I3,') : ',D15.8)
73       CONTINUE
      ENDIF
C
      IF ((EP(2) .EQ. .TRUE.) .AND. (PASSAGE .EQ. .FALSE.)) THEN
         UL = 2
         PASSAGE = .TRUE.
         GOTO 12000
      ENDIF
C
C     ----------------------------------------
C    |                                        |
C    |   Calcul du gradient de c, évalué en x |
C    |                                        |
C     ----------------------------------------
C
      CALL GRADc (n,x,Gc)
C
C     ---------------------------------
C    |                                 |
C    |   Impression du gradient de c   |
C    |                                 |
C     ---------------------------------
C
      PASSAGE = .FALSE.
      IF (EP(1) .EQ. .TRUE.) THEN
         UL = 5
      ELSE
         IF (EP(2) .EQ. .TRUE.) THEN
             UL = 2
             PASSAGE = .TRUE.
         ENDIF
      ENDIF
C
13000 IF (IMPR(8) .EQ. .TRUE.) THEN
         DO 75 I= 1, n
            WRITE (UL,1149) I, Gc(I)
1149        FORMAT(1X,'Gc(',I3,') = ',D15.8)
75       CONTINUE
      ENDIF
C
      IF ((EP(2) .EQ. .TRUE.) .AND. (PASSAGE .EQ. .FALSE.)) THEN
         UL = 2
         PASSAGE = .TRUE.
         GOTO 13000
      ENDIF
C
C     ---------------------------------
C    |                                 |
C    |   Fd = trans(Jg(x)).y + Gc(x)   |
C    |                                 |
C     ---------------------------------
C
      CALL DAXPY (n,1.0D0,Gc,1,Fd,1)
C
C
C     2.4. F(z) = (Fc,Fp,Fd)
C     **********************
C
      DO 77 I = 1, m
         F(I)   = Fc(I)
         F(m+I) = Fp(I)
77    CONTINUE
      DO 78 I = 1, n
         F(2*m+I) = Fd(I)
78    CONTINUE
C
C
      RETURN
      END
```