Amélioration de la robustesse d'un simulateur pour le procédé d'extraction liquide-liquide.

Kevin HERILUS
Elève-ingénieur de la formation Mathématiques Appliquées et Calcul Scientifique

Stage réalisé du 14 avril au 30 septembre 2014
Encadré par Pascal Omnes, avec le concours d'Eli Laucoin
Ingénieurs CEA/DEN/DM2S/STMF

Table des matières

Liste des tableaux

Table des figures

Introduction

Dans le cadre de ma formation d'ingénieur en mathématiques appliquées, j'ai effectué un stage de fin d'études de cinq mois et demi. Les six premières semaines se sont déroulées au Laboratoire d'Analyse, Géométrie et Applications et ont été consacrées à un travail bibilographique. Le reste du travail a été effectué au sein du Service de Thermohydraulique et de Mécanique des Fluides du Département de Modélisation des Systèmes et Structures de la Direction de l'Energie Nucléaire du Commissariat à l'Énergie Atomique (CEA) de Saclay. Ce stage a eu pour but de travailler sur un problème de calcul scientifique en situation d'ingénieur au sein d'un laboratoire de recherche.

En chimie, notamment en cinétique chimique, l'on a besoin de déterminer l'évolution en fonction du temps des concentrations des différentes espèces dans un mélange. Une fois les équations discrétisées en temps, nous devons à chaque pas de temps résoudre un système algébrique non-linéaire. La stratégie de résolution choisie et appliquée pour tenter de résoudre le problème est issue d'une méthode plus générale explicitée dans le premier chapitre de ce rapport. J'y donnerai notamment un premier algorithme itératif permettant de résoudre des problèmes sans contrainte *a priori* sur les inconnues, après avoir introduit la notion de région de confiance. Je décrirai précisément la méthode employée pour le calcul du pas entre deux itérés, combinaison linéaire d'un pas dit *inexact* de Newton et d'une direction de descente. Je fournirai aussi un résultat de convergence sur cet algorithme.

Puis, dans le deuxième chapitre, je modifie l'algorithme pour imposer à tout instant des contraintes sur les inconnues du problème. Pour ce qui nous concerne, ce sont des contraintes de positivité sur certaines variables (concentrations, masses, volumes...). C'est un algorithme qui, à chaque itération, résout un problème de minimisation sous contrainte (explicité plus loin dans ce rapport). La modification ne concerne que le calcul du pas et tient compte à la fois des contraintes sur les inconnues et de la région de confiance. Lorsque ce sera nécessaire, je démontrerai des résultats pour la compréhension de l'algorithme. On y énoncera aussi un résultat important sur la convergence de la méthode numérique.

Dans le troisième chapitre, j'utiliserai les résultats et les algorithmes des deux chapitres précédents pour résoudre deux problèmes de mathématiques appliquées à la chimie, ce qui me permettra par ailleurs de présenter l'environnement informatique qui m'a permis d'implémenter des méthodes améliorant les résultats issus d'une première version du code. On testera le code sur deux exemples et on donnera le cas échéant les modifications effectuées.

Chapitre 1

Stratégie de résolution d'un système non linéaire d'équations

1.1 Objectif et notations

On souhaite résoudre le système d'équations non linéaires

$$F(x) = 0, \tag{1.1}$$

où F est une application de $\mathbb{R}^n$ à valeurs dans $\mathbb{R}^n$, continûment différentiable. On suppose l'existence d'une solution de (1.1). Une des stratégies de résolution consiste à minimiser la fonctionnelle

$$f(x) = \frac{1}{2} \|F(x)\|^2 ,$$

où $\|\cdot\|$ est la norme associée à un produit scalaire, qui n'est pas nécessairement le produit scalaire euclidien. Ayant supposé l'exsitence d'une solution de (1.1) et puisque $f \geq 0$, alors le minimum est 0 et est atteint en la solution de (1.1). Résoudre (1.1) ou minimiser f sont équivalents. En effet, si $x_\star$ réalise le minimum de f, alors $f(x_\star) = 0$ et $x_\star$ est une solution de (1.1). Réciproquement, si $x_\star$ est une solution de (1.1), par passage à la norme, $x_\star$ réalise le minimum de f.
On va utiliser une méthode itérative pour minimiser f. Ces itérés seront notés (x_k). On note $F_k = F(x_k)$ et on note $F'_k = F'(x_k)$, où F' est la jacobienne de F.

1.2 Méthodes de Newton

Afin de résoudre un système de type (1.1), on peut appliquer la méthode (classique) de Newton, méthode construisant une suite d'itérés de la manière suivante : on a $x_{k+1} = x_k + s_k$, où le pas s_k est la solution, si elle existe, de l'équation de Newton

$$F'_k s_k = -F_k. \tag{1.2}$$

Il existe aussi une classe de méthodes de Newton inexactes, où, à chaque itération k, on réduit la norme du modèle linéaire de F en x_k. Cette quantité est donnée par la

Définition 1 (Norme du modèle linéaire d'une application)**.** *Soit $F : \mathbb{R}^n \to \mathbb{R}^n$. On appelle norme du modèle linéaire de F en x, la quantité*

$$m(p) = \left\| F(x) + F'(x)p \right\| , \tag{1.3}$$

où $p \in \mathbb{R}^n$.

On choisit alors le pas inexact de Newton s_k^{IN} de sorte que $m_k(s_k^{IN}) \leq \eta_k \|F_k\|$, où m_k est la norme du modèle linéaire de F_k en x_k et $\eta_k \in]0, 1]$ un *terme de forçage*.

Pour obtenir la convergence de la méthode, on "complète" ces méthodes inexactes de Newton avec d'autres procédures. Il existe deux types de procédures :

– le *backtracking* : où l'on règle la longueur du pas (le plus souvent, on diminue sa longueur). Cette méthode est a priori facile à implémenter, mais cette direction de descente peut s'avérer faible, surtout si la matrice jacobienne de F est mal conditionnée ;
– la *méthode de la région de confiance*, où le pas minimise la norme du modèle linéaire local dans un ouvert (usuellement une boule) appelé région de confiance. C'est ce type de méthode qui est choisi pour notre étude. Cette méthode présente l'avantage d'obtenir de meilleures directions de descente que le pas d'essai initial s_k^{IN}. En revanche c'est son implémentation qui est délicate et il n'est pas toujours aisé de calculer le pas optimal dans la région de confiance, et on est alors amené à chercher des approximations de ce pas. L'une des méthodes les plus populaires pour cela est la méthode *dogleg* décrite ci-dessous. Nous verrons plus loin dans ce rapport l'avantage de la programmation orientée objet par rapport à l'implémentation de ce type de méthode.

1.3 Algortihme INDL (Inexact Newton DogLeg)

Les méthodes de type *dogleg* sont les méthodes les plus utilisées pour le calcul du pas satisfaisant le critère de la région de confiance. Il y a quatre points clés dans cet algorithme :
– le calcul d'un pas de Newton ;
– le calcul de la direction de descente de la plus grande pente ;
– le calcul du pas de dogleg ;
– le test qui rejette ou accepte ce pas.

1.3.1 Calcul du pas de Newton

Ce que l'on appelle "pas de Newton" dans cette sous-section correspond ici au pas *inexact* de Newton noté s_k^{IN}. (*cf.* §1.2).

1.3.2 Calcul de la direction de descente approximative

On rappelle le résultat suivant :

Théorème 1 (admis)**.** *Soit $< \cdot, \cdot >$ un produit scalaire et $< \cdot, \cdot >_2$ le produit scalaire euclidien (en notant $\|\cdot\|$ et $\|\cdot\|_2$ les normes associées). Il existe une unique matrice symétrique définie positive telle S que pour tous $u, v \in \mathbb{R}^n$, $< u, v >=< u, Sv >_2$.*

Soit $w \in \mathbb{R}^n$, pour évaluer le gradient de la fonctionnelle f dans la direction w, on calcule

$$\begin{aligned}
f(x+w) - f(x) &= \frac{1}{2}\left(\|F(x+w)\|^2 - \|F(x)\|^2\right) \\
&= \frac{1}{2}\left(\left\|F(x) + F'(x)w\right\|^2 - \|F(x)\|^2\right) + o(\|w\|) \\
&= \frac{1}{2}\left(< F(x) + F'(x)w, S(F(x) + F'(x)w) >_2 - < F(x), SF(x) >_2\right) + o(\|w\|) \\
&= \frac{1}{2}\left(< F'(x)w, SF(x) >_2 + < F(x), SF'(x)w >_2\right) + o(\|w\|) \\
&= < F'(x)^T SF(x), w >_2 + o(\|w\|) \\
&= < S^{-1}F'(x)^T SF(x), w > + o(\|w\|).
\end{aligned}$$

On en déduit que le gradient de la fonctionnelle (par rapport au produit scalaire $< \cdot, \cdot >$ s'écrit

$$\nabla f(x) = S^{-1}F'(x)^T SF(x), \tag{1.4}$$

puis que la direction de descente de la plus grande pente associée au produit scalaire $< \cdot, \cdot >$ est

$$d(x) = -S^{-1}F'(x)^T SF(x). \tag{1.5}$$

1.3.3 Calcul du pas de Cauchy

Le calcul exact de la direction de descente peut éventuellement être très difficile parce que l'on doit effectuer des multiplications par la matrice $F'(x)^T$. On calcule donc une direction de descente *approximative*. Soit $\hat{d_k} \approx d(x_k)$ cette direction. On peut alors déterminer le pas de Cauchy $\hat{s_k}^{CP}$ défini par

$$\hat{s_k}^{CP} = \text{argmin}\{\left\|F(x_k) + F'(x_k)s\right\|, s = \lambda \hat{d_k}, \lambda \in \mathbb{R}\}. \tag{1.6}$$

(1.6) signifie que le pas de Cauchy minimise la norme du modèle linéaire *local* de F le long d'une direction de descente approximative.

1.3.4 Calcul du pas de dogleg

On construit un pas d'essai sur lequel on teste une condition de rejet ou non du pas (voir §1.3.5). La procédure adoptée est la suivante :

Si ($\left\|s_k^{IN}\right\| \leq \delta$) **Alors**
  $s_k = s_k^{IN}$
Sinon
  Si ($\left\|\hat{s_k}^{CP}\right\| \geq \delta$) **Alors**
    $s_k = \dfrac{\delta}{\left\|\hat{s_k}^{CP}\right\|} \hat{s_k}^{CP}$
  Sinon
    On cherche $\gamma \in [0,1]$ tel que $s_k = \gamma s_k^{IN} + (1-\gamma)\hat{s_k}^{CP}$ vérifie $\|s_k\| = \delta$
  Fin Si
Fin Si

Algorithme 1: Calcul du pas de dogleg

Cette procédure est construite de sorte que si le pas de Newton inexact est dans la région de confiance, le pas d'essai final sera le pas de Newton inexact, ce qui ramène la méthode INDL à une méthode de Newton inexacte classique. Sinon, lorsque le pas de Cauchy sort de la région de confiance, on modifie ce pas de sorte que sa norme soit égale au rayon de la région de confiance et on choisit ce pas comme pas d'essai final. Si aucune des deux conditions précédentes n'est remplie, on construit le pas d'essai s_k par combinaison linéaire de ${s_k}^{CP}$ et de s_k^{IN}, ce pas d'essai devant être de norme égale au rayon de la région de confiance.

On accompagne la description du calcul du pas de dogleg du schéma présenté figure 1.1, dans le cas de la dimension deux.

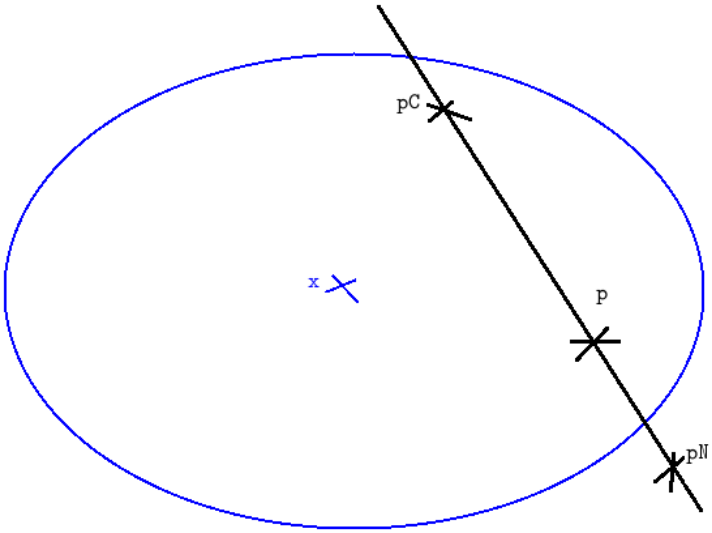

FIGURE 1.1 – Détermination de γ.

1.3.5 Condition d'amélioration

Une fois que le pas de dogleg a été calculé, on introduit une condition qui rejette ou accepte le pas d'essai. Considérons alors, pour l'itération k, la valeur réelle de la réduction de $\|F\|$, soit

$$a_k = \|F(x_k)\| - \|F(x_k + s_k)\|,$$

avec la valeur prédite par la norme du modèle linéaire de F en x_k, soit

$$p_k = \|F(x_k)\| - \left\|F(x_k) + F'(x_k)s_k\right\|.$$

Le pas s_k est alors accepté lorsque

$$a_k \geq 0 \text{ et } a_k \geq \beta p_k, \tag{1.7}$$

pour $\beta \in [0, 1]$ et β indépendant de k. Lorsque la condition d'amélioration (1.7) n'est pas satisfaite, on réduit le rayon de la région de confiance et on recommence la procédure du calcul du pas de dogleg. Dans le cas contraire, on met à jour la valeur de x_k, et on recommence la procédure avec une autre valeur du rayon de la région de confiance.

1.3.6 Récapitulatif

On introduit les paramètres : x_0, $\eta_{\max} \in]0, 1]$, $t \in]0, 1]$, $\theta_{\max} \in]0, 1]$, $\delta_{\min} > 0$, et $\delta \geq \delta_{\min}$

Pour k **de** 1 **à** $\cdots$ **faire**
  Choisir $\eta_k \in]0, \eta_{\max}[$ et s_k^{IN} tel que $\left\|F_k + F_k' s_k^{IN}\right\| \leq \eta_k \|F_k\|$
  Calculer $\hat{d_k}$ puis $\hat{s_k}^{CP}$;
  Calculer le point s_k sur la courbe dogleg vérifiant $\min\{\delta_{\min}, \left\|s_k^{IN}\right\|\} \leq \|s_k\| \leq \delta$
  Tester la condition d'amélioration
  Tant que (la condition d'amélioration n'est pas vérifiée) **faire**
    Réduire le rayon de la région de confiance
    Si ($\delta = \delta_{\min}$) **Alors**
      stop
    Sinon
      Choisir $\theta \in [0, \theta_{\max}]$
      $\delta = \max\{\theta\delta, \delta_{\min}\}$
    Fin Si
    Calculer le point s_k sur la courbe dogleg vérifiant $\min\{\delta_{\min}, \left\|s_k^{IN}\right\|\} \leq \|s_k\| \leq \delta$
    Tester la condition d'amélioration
  Fait
  Calculer le nouvel itéré $x_{k+1} = x_k + s_k$
  Augmenter le rayon de la région de confiance *i.e.* $\delta = \theta'\delta$, $\theta' > 1$
Fin Pour

Algorithme 2: Inexact Newton Dogleg Method

Remarques sur l'algorithme : La boucle tant que n'est pas une boucle infinie. En effet, soit le pas de dogleg est accepté, ou alors l'algorithme s'arrête avec $\delta = \delta_{\min}$, au bout d'un nombre fini d'iterations.

1.3.7 Convergence de l'algorithme

On commence par fournir la définition d'un point stationnaire.

Définition 2 (Point stationnaire)**.** *Soit $x \in \mathbb{R}^n$. On dit que x est un point stationnaire de $\|F\|$ si, pour tout $s \in \mathbb{R}^n$, $\|F(x)\| \leq \|F(x) + F'(x)s\|$* [1].

Le résultat de convergence est fourni par le

Théorème 2. *[Convergence de l'algorithme INDL* [2]*, admis] On suppose que F est continument différentiable. On suppose que la suite (x_k) est généré par l'algorithme "Inexact Newton Dogleg", et que pour une direction de descente définie par* (1.5) *et pour un certain $\epsilon > 0$,*

$$\frac{< \hat{d}_k, d(x_k) >}{\left\|\hat{d}_k\right\| \|d(x_k)\|} \geq \epsilon \tag{1.8}$$

pour tout k. Si $x_\star$ est une limite de la suite (x_k) alors $x_\star$ est un point stationnaire de $\|F\|$. Si, de plus, $F'(x_\star)$ est non-singluière, alors $F(x_\star) = 0$ et $x_k \to x_\star$. De plus, pour tout k suffisamment grand, le pas s_k initial est accepté sans modification de la boucle tant que et $s_k = s_k^{IN}$ est un pas admissible.

Cette méthode est développée dans un contexte *non contraint*, ce qui sginifie que l'on n'impose pas de bornes sur les itérés x_k pour tout k. Tout ce qui suit et notamment l'application à la cinétique chimique nous oblige à repenser la démarche et à réécrire une partie de l'algorithme pour qu'il puisse s'adapter à un contexte *contraint*, et en particulier conserver des contraintes de positivité.

1. Dans le contexte de notre étude, x est un point stationnaire si et seulement si $\nabla f(x) = 0$.
2. On trouvera la démonstration de ce théorème dans [3].

Chapitre 2

Stratégie de résolution d'un système non linéaire d'équations sous contrainte

2.1 Objectif et notations

L'objectif de ce chapitre est de proposer une méthode numérique de résolution de

$$F(x) = 0. \tag{2.1}$$

En revanche, F est une application non-linéaire definie sur X à valeurs dans $\mathbb{R}^n$, X étant un ouvert contenant la boîte à n dimensions $\Omega = \{x \in \mathbb{R}^n, l \leq x \leq u\}$ [1]. Il faut donc modifier certaines procédures de l'algorithme proposé dans le chapitre précédent pour prendre en compte les bornes sur les valeurs de x à une itération donnée. On conserve les notations introduites au chapitre précédent. On note cependant $(x_k)_i$ la $i^{\text{ème}}$ composantge de x_k, et lorsqu'il n'y a pas d'ambiguité, les parenthèses sont omises. On note par ailleurs $\mathcal{M}_n(\mathbb{R})$ l'ensemble des matrices carrées de taille n.
Ce chapitre est une reprise du chapitre précédent, complété par les démonstrations de résultats cités dans [1], [2], et [3].

2.2 Algorithme CODO (COnstrained DOgleg)

Pour déterminer les solutions de $F(x) = 0$, on va, comme lors du chapitre 1 chercher les solutions de

$$\min_{x \in \Omega} f(x), \tag{2.2}$$

où

$$f(x) = \frac{1}{2} \|F(x)\|^2 .$$

La différence avec le chapitre précédent réside dans l'apparition des contraintes $x \in \Omega$. On peut démontrer (*cf.* [4]) que les conditions d'optimalité du premier ordre sont données par

Proposition 1 (Conditions d'optimalité du premier ordre)**.** *Soit D_k une matrice diagonale appelée scaling matrix, on a*

$$D_k \nabla f_k = 0,$$

où $\nabla f_k = (F_k')^T F_k$ et $\nabla f_k = \nabla f(x_k)$ et D_k une matrice diagonale [2] *dont les éléments $d_i(x)$ vérifient*
$\rightarrow$ $d_i(x) = 0$,lorsque $x_i = l_i$ (resp. $x_i = u_i$) et $(\nabla f(x))_i > 0$ (resp. $(\nabla f(x))_i < 0$) ;
$\rightarrow$ $d_i(x) \geq 0$ si $x_i \in \{l_i; u_i\}$ et $(\nabla f(x))_i = 0$ $d_i(x) \geq 0$;
$\rightarrow$ $d_i(x) > 0$, dans tous les autres cas.

L'écriture de ces conditions d'optimalité nous donne une direction de descente qui sera utilisée dans l'algorithme, en l'occurrence $\hat{g}_k = -D_k \nabla f_k$.

1. L'inégalité doit être comprise composante par composante *i.e.* $\forall i, 1 \leq i \leq n, l_i \leq x_i \leq u_i$.
2. L'article [1] fournit quatre exemples de telles matrices.

2.2.1 Le problème de la région de confiance

À chaque itération k, on cherche à résoudre de manière approchée le sous-problème dont la définition est donnée ci-dessous.

Définition 3 (Région de confiance)**.** *Soit $\Delta > 0$. On appelle région de confiance l'ensemble des vecteurs $p \in \mathbb{R}^n$ vérifiant $\|Gp\| \leq \Delta$, $G = G(x) \in \mathcal{M}_n(\mathbb{R})$. On dit que Δ est le rayon de la région de confiance.*

Définition 4 (Sous-problème de la région de confiance)**.** *On suppose $x_k \in \Omega$. On appelle sous-problème de la région de confiance pour l'algorithme Constrained Dogleg le sous-problème suivant :*

$$\min_{p \in \mathbb{R}^n} \{m_k(p) \text{ t.q. } ||G_k p|| \leq \Delta_k, x_k + p \in int(\Omega)\},$$

où $G_k = G(x_k) \in \mathcal{M}_n(\mathbb{R})$, m_k est la norme du modèle linéaire de F en x_k cf. (1.3)*, et Δ_k le rayon de la région de confiance à l'itération k.*

On étudiera deux exemples dans ce stage. Soient g_j, d_j, $1 \leq j \leq n$ les éléments diagonaux de G et de D respectivement et p_j les composantes de p.
− Lorsque $G_k = I$, on dit que la région de confiance est *sphérique*. En effet, on obtiendrait dans ce cas

$$\sum_{j=1}^{n} p_j^2 \leq \Delta_k^2;$$

− Lorsque $G_k = D^{-1/2}$, on dit que la région de confiance est *elliptique*. De la même manière, on obtiendrait

$$\sum_{j=1}^{n} p_j^2 g_j^2 = \sum_{j=1}^{n} \frac{p_j^2}{d_j} \leq \Delta_k^2.$$

Afin de rechercher la solution approchée de ce sous-problème, on va appliquer une méthode de type *Inexact Newton Dogleg*, mais que l'on va adapter à la prise en compte des contraintes $x \in \Omega$ (d'où le nom de *Constrained Dogleg Method*). On a donc besoin de définir un pas de Newton et un pas de Cauchy.

2.2.2 Calcul du pas de Newton

Nous avons vu au dans le chapitre 1 que le pas de Newton est un pas inexact. Dans nos applications numériques, les systèmes non linéaires que l'on doit résoudre sont à de taille moyenne (de l'ordre de la centaine au millier d'inconnues au maximum), on peut donc se permettre de calculer de manière exacte le pas de Newton (l'inversion de la matrice jacobienne se faisant via une décomposition LU). Il s'agit alors ici de calculer $p_k^N \in \mathbb{R}^n$ vérifiant

$$F_k' p_k^N = -F_k$$

à condition de pouvoir inverser la matrice jacobienne de F_k. Remarquons ici que rien ne garantit que $x_k + p_k^N \in \text{int}(\Omega)$. On construit alors

$$\bar{p_k}^N = \alpha_k (P(x_k + p_k^N) - x_k), \tag{2.3}$$

où $\alpha_k \in [0, 1[$ un paramètre donné par l'utilisateur et P étant le projecteur défini de la manière suivante :

$$\begin{cases} \text{si } l_i \leq x_i \leq u_i, & \text{alors } P(x)_i = x_i; \\ \text{si } x_i \leq l_i, & \text{alors } P(x)_i = l_i; \\ \text{si } u_i \leq x_i, & \text{alors } P(x)_i = u_i. \end{cases}$$

On peut aussi écrire la composante i du projecteur sous la forme

$$P(x)_i = \max(l_i, \min(x_i, u_i)).$$

Notons ici que c'est une méthode de projection composante par composante. On remarquera que lors de nos applications numériques, la manière dont les pas (pas de Cauchy et pas de Newton) vont être projetés aura son importance.

2.2.3 Calcul du pas de Cauchy

Ce vecteur est une direction de descente, et doit se situer dans la région de confiance donnée par la défintion 3.

Proposition 2. *Soit $\hat{g}_k \in \mathbb{R}^n$ défini par $\hat{g}_k = -D_k \nabla f_k$. Dans ce cas, le point de Cauchy noté $p_c(\Delta_k)$ est de la forme $p_c(\Delta_k) = \tau_k \hat{g}_k$ où $\tau_k = min\left(-\frac{{}^tF_k F'_k \hat{g}_k}{||F'_k \hat{g}_k||^2}, \frac{\Delta_k}{||G_k \hat{g}_k||}\right)$ si $x_k + \tau_k \hat{g}_k \in int(\Omega)$.*

On démontre ce résultat en prouvant la proposition suivante :

Proposition 3. *Soient a,$b \in \mathbb{R}^n$, $b \neq 0$. La fonction ψ de la variable réelle x définie par $\psi(x) = \|a + bx\|$ atteint son minimum en*

$$\hat{x} = -\frac{(a,b)}{(b,b)}.$$

Démonstration. *Pour tout réel $x \in \mathbb{R}$, on a :*

$$\psi(x) = \sqrt{(a + bx, a + bx)},$$

ce qui donne par dérivation par rapport à x,

$$\psi'(x) = \frac{2x(b,b) + 2(a,b)}{2\psi(x)}.$$

L'équation $\psi'(x) = 0$ admet donc une solution donnée par

$$\hat{x} = -\frac{(a,b)}{(b,b)}.$$

D'autre part

$$\psi''(x) = \frac{(b,b)\psi(x) - (x(b,b) + (a,b))\psi'(x)}{\psi(x)^2},$$

donc

$$\psi''(\hat{x}) = \frac{(b,b)\psi(\hat{x}) - (x(b,b) + (a,b))\psi'(\hat{x})}{\psi(\hat{x})^2},$$

soit encore

$$\psi''(\hat{x}) = \frac{(b,b)}{\psi(\hat{x})} > 0,$$

ce qui achève la preuve. □

On revient à la démonstration de la proposition 2.
Démonstration. *On cherche la valeur minimale le long de $\hat{g}_k$ et tel que le vecteur obtenu soit dans la région de confiance. Autrement dit, on cherche $\min\{m_k(\tau_k \hat{g}_k) = \left\|F_k + \tau_k F'_k \hat{g}_k\right\| : \|G_k \tau_k \hat{g}_k\| \leq \Delta_k, x_k + \tau_k \hat{g}_k \in int(\Omega)\}$. Il suffit alors de poser $a = F_k$ et $b = F'_k \hat{g}_k$, et d'appliquer le résultat de la proposition 3 ce qui donne la valeur $\tau_k = -\frac{(a,b)}{(b,b)}$, soit encore*

$$\tau_k = -\frac{F_k^T F'_k \hat{g}_k}{\left\|F'_k \hat{g}_k\right\|^2}.$$

D'autre part, la contrainte $\|G_k \tau_k \hat{g}_k\| \leq \Delta_k$ fournit $|\tau_k| \leq \frac{\Delta_k}{\|G_k \hat{g}_k\|}$. D'où le résultat. □

On utilise ici une méthode de point intérieur *i.e.* $x_k \in \text{int}(\Omega)$ pour tout k. Traitons à présent le cas où $x_k + \tau_k \hat{g}_k \notin \text{int}(\Omega)$, avec τ_k calculé par (2.2.3). On souhaite savoir la taille maximale du pas autorisée qui permet de "rester" dans le domaine Ω. On cherche dans un premier temps, pour tout i, $1 \leq i \leq n$, la valeur maximale de λ_i telle que

$$l_i < (x_k)_i + \lambda_i (\hat{g}_k)_i < u_i$$

– Lorsque $(\hat{g}_k)_i = 0$, on peut imposer à λ_i la valeur que l'on veut, par exemple $\lambda_i = +\infty$;
– Lorsque $(\hat{g}_k)_i \neq 0$, on a, si $(\hat{g}_k)_i > 0$

$$\frac{l_i - (x_k)_i}{(\hat{g}_k)_i} \leq \lambda_i \leq \frac{u_i - (x_k)_i}{(\hat{g}_k)_i},$$

et si $(\hat{g}_k)_i < 0$ on a

$$\frac{u_i - (x_k)_i}{(\hat{g}_k)_i} \leq \lambda_i \leq \frac{l_i - (x_k)_i}{(\hat{g}_k)_i},$$

ce qui revient à prendre

$$\lambda_i = \max\left(\frac{u_i - (x_k)_i}{(\hat{g}_k)_i}, \frac{l_i - (x_k)_i}{(\hat{g}_k)_i}\right). \tag{2.4}$$

Ceci étant fait pour chacune des directions, on prend

$$\lambda_k = \min_{1 \leq i \leq n} \lambda_i \tag{2.5}$$

et on impose finalement $\tau_k = \theta\lambda_k$, où $\theta \in [0, 1[$ ($\theta < 1$ pour être sûr que l'on reste à l'intérieur de Ω) est aussi un paramètre fourni par l'utilisateur. Remarquons ici que cette technique de projection est globale dans le sens où le même coefficient est appliqué à toutes les composantes de $\hat{g}_k$.

2.2.4 Calcul du pas de dogleg

Le pas de Cauchy et le pas de Newton ayant été calculés, l'étape suivante consiste à considérer le chemin

$$p(\gamma) = (1 - \gamma)p_c(\Delta_k) + \gamma \bar{p_k}^N, \tag{2.6}$$

où $\gamma \in \mathbb{R}$ et à chercher la valeur de γ minimisant $m_k(p(\gamma))$, sous les contraintes $\|G_k p(\gamma)\| \leq \Delta_k$,et $x_k + p(\gamma) \in \text{int}(\Omega)$. On commence par résoudre le problème sans contrainte avec la

Proposition 4. *La fonction ϕ de la variable γ définie par $\phi(\gamma) = \|F_k + F'_k p(\gamma)\|$ atteint une valeur minimale en $\gamma = \hat{\gamma}$, où*

$$\hat{\gamma} = -\frac{(F_k + F'_k p_c(\Delta_k))^T F'_k (p_c(\Delta_k) - \bar{p_k}^N)}{\left\|F'_k (p_c(\Delta_k) - \bar{p_k}^N)\right\|^2}. \tag{2.7}$$

Démonstration. *On reporte la valeur de $p(\gamma)$ dans $m_k(p(\gamma))$, ce qui donne, en rassemblant les termes linéaires en γ,*

$$m_k(p(\gamma)) = \left\|F_k + F'_k p_c(\Delta_k) + \gamma(F'_k(p_c(\Delta_k) - \bar{p_k}^N))\right\|.$$

On applique de nouveau le résultat de la proposition 3, avec $a = F_k + F'_k p_c(\Delta_k)$ et $b = F'_k(p_c(\Delta_k) - \bar{p_k}^N)$. □

Il nous faut à présent déterminer une condition sur γ pour lesquelles $p(\gamma)$ atteint les frontières de la région de confiance afin de savoir si la valeur $\hat{\gamma}$ calculée par (2.7) est à l'intérieur de cette région. Elles sont données par la

Proposition 5. *$p(\gamma)$ a deux intersections avec les frontières de la région de confiance, données pour $\gamma = \gamma_\pm$, où*

$$\gamma_\pm = \frac{-B \pm \sqrt{B^2 - AC}}{A},$$

où

$$\begin{aligned} A &= \left\|G_k(\bar{p_k}^N - p_c(\Delta_k))\right\|^2, \\ B &= p_c(\Delta_k)^T G_k^2 \,(\bar{p_k}^N - p_c(\Delta_k)), \\ C &= \|G_k p_c(\Delta_k)\|^2 - \Delta_k^2. \end{aligned}$$

Démonstration. *On doit résoudre l'équation* $\|G_k p(\gamma)\|^2 = \Delta_k^2$*. Or,*

$$\begin{aligned}\|G_k p(\gamma)\|^2 &= \left\|G_k((1-\gamma)p_c(\Delta_k) + \gamma \bar{p_k}^N)\right\|^2 \\ &= \left\|G_k p_c(\Delta_k) + \gamma G_k(\bar{p_k}^N - p_c(\Delta_k))\right\|^2 \\ &= \|G_k p_c(\Delta_k)\|^2 + \gamma^2 \left\|G_k(\bar{p_k}^N - p_c(\Delta_k))\right\|^2 \\ &+ 2\gamma(G_k p_c(\Delta_k), G_k(\bar{p_k}^N - p_c(\Delta_k)).\end{aligned}$$

l'équation devient

$$\|G_k p_c(\Delta_k)\|^2 + \gamma^2 \left\|G_k(\bar{p_k}^N - p_c(\Delta_k))\right\|^2 + 2\gamma(G_k p_c(\Delta_k), G_k(\bar{p_k}^N - p_c(\Delta_k)) - \Delta_k^2 = 0,$$

soit encore, puisque G_k *est symétrique*

$$\left\|G_k(\bar{p_k}^N - p_c(\Delta_k))\right\|^2 \gamma^2 + 2p_c(\Delta_k)^T G_k^2(\bar{p_k}^N - p_c(\Delta_k))\gamma + \|G_k p_c(\Delta_k)\|^2 - \Delta_k^2 = 0 \tag{2.8}$$

ou encore

$$A\gamma^2 + 2B\gamma + C = 0,$$

où A, B*, et* C *sont les coefficients de l'énoncé de la proposition. On a* $\bar{p_k}^N - p_c(\Delta_k) \neq 0$ *et le discriminant (réduit)* δ' *est*

$$\delta' = B^2 - AC.$$

Par définition $\|G_k p_c(\Delta_k)\| \leq \Delta_k$ *donc* $C < 0$*. Puisque* $A > 0$*,* $\delta' > 0$*. L'équation* (2.8) *admet donc deux solutions données par*

$$\gamma_\pm = \frac{-B \pm \sqrt{B^2 - AC}}{A},$$

ce qui prouve le résultat. □

Puisque $A > 0$ et $\delta > 0$, on sait que $\|G_k p(\gamma)\|^2 - \Delta_k^2 < 0$ lorsque $\gamma \in [\min(\gamma_+, \gamma_-); \max(\gamma_+, \gamma_-)]$. Or on sait que $C > 0$, donc γ_+ et γ_- n'ont pas le même signe et donc la plus grande des deux racines est positive. Puisque $\gamma_+ - \gamma_- = 2\dfrac{\sqrt{B^2 - AC}}{A} > 0$, on a $\gamma_+ - \gamma_- > 0$. D'où $\|G_k p(\gamma)\|^2 - \Delta_k^2 < 0$ lorque $\gamma \in [\min(\gamma_+, \gamma_-); \max(\gamma_+, \gamma_-)] = [\gamma_-; \gamma_+]$.

Si $\hat{\gamma} \in [0, 1]$, le pas $p(\gamma)$ sera dans la région de confiance et de fait $x_k + p(\gamma)$ sera dans int(Ω), par convexité. Il faut alors envisager les cas $\hat{\gamma} > 1$ et $\hat{\gamma} < 0$. Pour cela on rappelle que

$$p(\gamma) = p_c(\Delta_k) + \gamma(\bar{p_k}^N - p_c(\Delta_k)).$$

$\rightarrow$ Dans le cas $\hat{\gamma} > 1$, on va chercher pour tout i, $1 < i < n$, la valeur maximale de λ_i telle que

$$l_i \leq (x_k)_i + (p_c(\Delta_k))_i + \lambda_i(\bar{p_k}^N - p_c(\Delta_k))_i \leq u_i.$$

Là encore, en raisonnant sur les valeurs de $\bar{p_k}^N - p_c(\Delta_k)$, on aboutit à

$$\lambda_i = \begin{cases} \max\left(\dfrac{l_i - (x_k + p_c(\Delta_k))_i}{(\bar{p_k}^N - p_c(\Delta_k))_i}, \dfrac{u_i - (x_k + p_c(\Delta_k))_i}{(\bar{p_k}^N - p_c(\Delta_k))_i}\right) \text{ si } (\bar{p_k}^N - p_c(\Delta_k))_i \neq 0 \\ +\infty \text{ si } (\bar{p_k}^N - p_c(\Delta_k))_i = 0 \end{cases}.$$

On pose alors $\bar{\gamma}_+ = \min\limits_{1 \leq i \leq n} \lambda_i$. Pour assurer de plus que $p(\gamma)$ est dans la régon de confiance, on pose

$$\gamma = \min(\hat{\gamma}, \gamma_+, \theta\bar{\gamma}_+), \tag{2.9}$$

où θ est un paramètre dans $\in [0, 1[$.
$\rightarrow$ Dans le cas $\hat{\gamma} < 0$, on se ramène au cas $\hat{\gamma} > 1$ en changeant les signes. On obtient

$$\lambda_i = \begin{cases} \max\left(\dfrac{l_i - (x_k + p_c(\Delta_k))_i}{-(\bar{p_k}^N - p_c(\Delta_k))_i}, \dfrac{u_i - (x_k + p_c(\Delta_k))_i}{-(\bar{p_k}^N - p_c(\Delta_k))_i}\right) \text{ si } -(\bar{p_k}^N - p_c(\Delta_k))_i \neq 0 \\ +\infty \text{ si } -(\bar{p_k}^N - p_c(\Delta_k))_i = 0 \end{cases}.$$

On pose alors $\bar{\gamma}_- = -\min_{1\leq i\leq n} \lambda_i$. Pour assurer de plus que $p(\gamma)$ est dans la régon de confiance, on pose

$$\gamma = \max(\hat{\gamma}, \gamma_-, \theta\bar{\gamma}_-), \tag{2.10}$$

où θ est un paramètre dans $\in [0, 1[$.
Une fois que γ a été calculé, on reporte cette valeur dans la relation (2.6).

On peut résumer les étapes de calcul de γ avec le schéma présenté figure 2.1, dans le cas où $n = 2$. Le cadre bleu représente les frontières de la "boîte" Ω, tandis que l'ellipse rouge représente la frontière de la région de confiance.

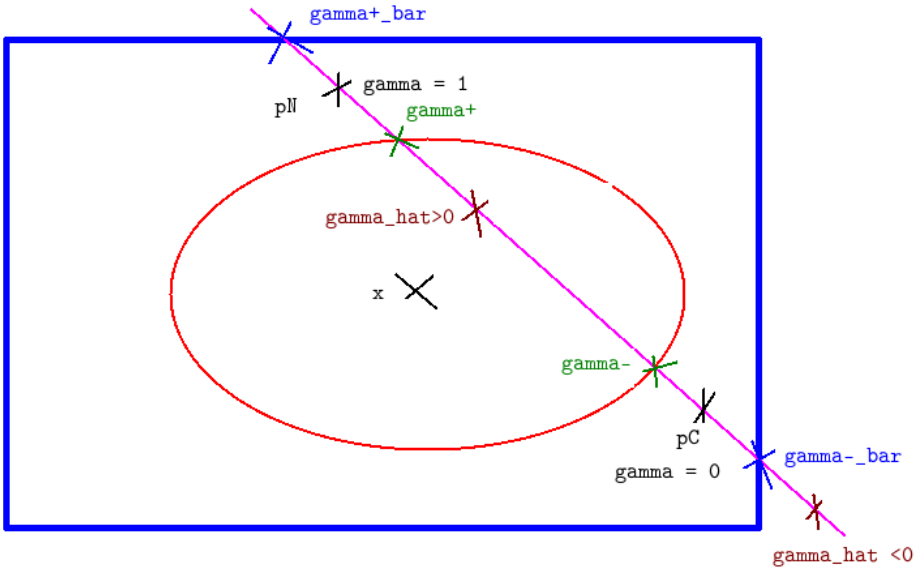

FIGURE 2.1 – Détermination de γ.

2.2.5 Condition d'amélioration

Le pas calculé est celui qui est obtenu pour la valeur de la taille de la région de confiance à l'itération k. Le pas $p(\gamma)$ dépend donc de Δ_k. Par la suite, on se réfère au §1.3.5 pour la vérfication de la condition. Si la condition (1.7) est vérifiée, on pourra poser $x_{k+1} = x_k + p(\Delta_k)$, et on revient à la première étape avec la possibilité d'augmenter la taille de la région de confiance. Dans le cas contraire, on recommence le calcul de $p(\Delta_k)$ avec une autre valeur de Δ_k. Plus exactement, on remplace Δ_k par $\delta\Delta_k$, où $\delta \in [0, 1]$.

2.2.6 Récapitulatif

• **Calcul de γ.** Supposons que l'on ait calculé x_k, $\hat{g}_k$, $p_c(\Delta_k)$ et $\bar{p_k}^N$, et que Δ_k et θ soient donnés. Pour calculer le pas, on applique la procédure suivante :

On détermine $\hat{\gamma}$ (*cf.* (2.7)) ;
Si ($\hat{\gamma} > 0$) **Alors**
  on calcule $\gamma_+, \theta\bar{\gamma}_+$ puis on calcule la valeur de γ par (2.9)
Sinon
  on calcule $\gamma_-, \theta\bar{\gamma}_-$ puis on calcule la valeur de γ par (2.10)
Fin Si

Algorithme 3: Calcul de γ.

• **Calcul du pas de dogleg**

Si ($\|\bar{p_k}^N\| \leq \Delta_k$) **Alors**
  $p_k = \bar{p_k}^N$
Sinon
  Si ($\|p_c(\Delta_k)\| \geq \Delta_k$) **Alors**
    $p_c(\Delta_k) = \dfrac{\Delta_k}{\|p_c(\Delta_k)\|} p_c(\Delta_k)$
  Sinon
    on calcule la valeur de γ par (2.9) ou (2.10)
    $p_k = \gamma\bar{p_k}^N + (1-\gamma)p_c(\Delta_k)$
  Fin Si
Fin Si

Algorithme 4: Calcul du pas de dogleg.

• **Méthode constrained dogleg.** Δ_k et θ sont ici donnés.

Pour k **de** 1 **à** $\cdots$ **faire**
Calcul de $\bar{p_k}^N$ (*cf.* (2.3))
Calcul de $p_c(\Delta_k)$ (*cf.* §2.2.3)
Calcul de γ puis de $p(\gamma)$ (*cf.* §2.2.4)
Test de la condition d'amélioration
Tant que (la condition d'amélioration n'est pas vérifiée) **faire**
$\Delta_k = \theta\Delta_k$, $\theta \in [0, 1[$
Calcul de $\bar{p_k}^N$ (*cf.* (2.3))
Calcul de $p_c(\Delta_k)$ (*cf.* §2.2.3)
Calcul de γ puis de $p(\gamma)$ (*cf.* §2.2.4)
Test de la condition d'amélioration
Fait
Calculer le nouvel itéré $x_{k+1} = x_k + p(\gamma)$
Augmenter le rayon de la région de confiance *i.e.* $\Delta_{k+1} = \theta'\Delta_k$, $\theta' > 1$
Fin Pour

Algorithme 5: Constrained Dogleg Method

Remarque sur l'algorithme : On note que l'on doit, pour faire fonctionner cet algorithme, piloter de nombreux paramètres. L'algorithme se trouve être par ailleurs, après implémentations et tests, très sensible à ces paramètres (variation du nombre d'itérations au cours d'un pas de temps par exemple).

2.2.7 Convergence de l'algorithme Constrained Dogleg

On va supposer les hypothèses suivantes sur la scaling matrice D_k :
• H1 : Les propriétés sur les éléments diagonaux de D_k sont satisfaites ;
• H2 : Soit $B_\rho(x)$ la boule centrée en x de rayon $\rho > 0$, on suppose D_k bornée dans $\Omega \cap B_\rho(x)$ pour tout $x \in \Omega$ et pout tout ρ ;
• H3 : Il existe $\bar{\lambda} > 0$ tel que la distance à la frontière λ_k à partir de x_k le long de $\hat{g}_k = -D_k\nabla f_k$ donnée par (2.5) soit telle que $\lambda_k > \bar{\lambda}$ chaque fois que $\|\nabla f_k\|$ est uniformément borné ;
• H4 : Pour tout $\bar{x} \in \text{int}(\Omega)$, il existe deux constantes positives $\bar{\rho}$ et $\chi_{\bar{x}}$ telles que $B_\rho(\bar{x}) \subset \text{int}(\Omega)$ et que $\left\|D_k(x)^{-1}\right\| \leq \chi_{\bar{x}}$ pour tout $x \in B_{\rho/2}(\bar{x})$.
L'algorithme Constrained Dogleg est globalement convgergente, et localement à convergence rapide. On a même le théorème suivant

Théorème 3 (Convergence de la méthode Constrained Dogleg, admis)**.** *On suppose que les hypothèses H1 à H3 sont satisfaites. Soit (x_k) la suite des points générés par l'aglorithme CoDo. Si cette suite est bornée, alors :*
• *Tous les points limites de la suite (x_k) sont des points stationnaires de f ;*
• *S'il existe une limite $x^\star \in int(\Omega)$ de la suite (x_k) tel que $F'(x^\star)$ ne soit pas singulière alors $\|F_k\| \to 0$ et tous les points d'accumulation de x_k sont soultion de* (2.2) *;*
• *S'il existe un point limite $x_\star \in \Omega$ tel que $F(x_\star) = 0$ et $F'(x_\star)$ inversible alors $x_k \to x_\star$.*

Chapitre 3

Application à un problème de chimie

3.1 L'extraction liquide-liquide

On pourra trouver plus de détails dans [5].
Les phases sont décrites sur la figure ci-dessous 3.1.

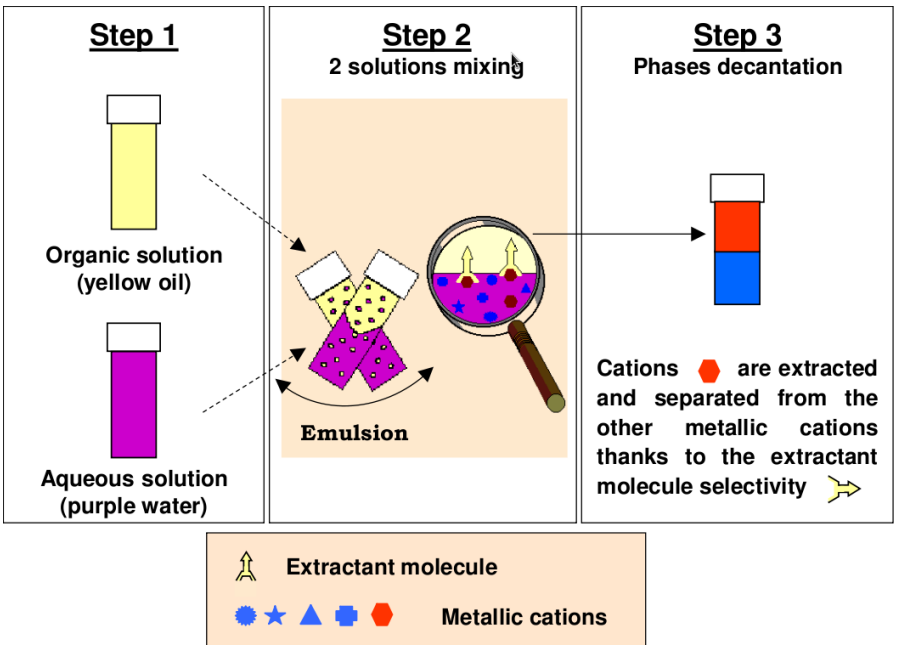

FIGURE 3.1 – Les étapes de l'extraction liquide-liquide.

Phase1. On considère une phase aqueuse composée de plusieurs espèces dont on veut en extraire certains. On considère aussi une phase organique, celle-ci étant constituée de molécules extractantes spécifique à l'espèce que l'on souhaite extraire de la phase aqueuse.
Phase2. On mélange la phase aqueuse et une phase organique de sorte d'obtenir une émulsion. Au cours de cette émulsion, les molécules à extraire se retrouvent à l'interface entre les deux phases et se retrouvent piégées par les molécules extractantes.
Phase3. On laisse décanterle mélange et en conséquence toutes les molécules à extraire sont dans la phase organique, et la phase aqueuse se retrouve débarassée de l'élément à extraire.

3.2 Les équations à résoudre

3.2.1 Les équations issues de la chimie

Description de la solution chimique.

Considérons des espèces E_i, $1 \leq i \leq N_s$, réagissant entre elles suivant
- N_r réactions que l'on écrit sous la forme

$$\sum_{i=1}^{N_s} \delta_i^j E_i \rightarrow \sum_{i=1}^{N_s} \mu_i^j E_i,\ j \in [1, N_r]$$

- N_e équilibres que l'on écrit sous la forme

$$\sum_{i=1}^{N_s} \gamma_i^j E_i \rightleftarrows \sum_{i=1}^{N_s} \theta_i^j E_i,\ j \in [1, N_e]$$

On pose en outre $\beta_i^j = \gamma_i^j - \theta_i^j$ et $\alpha_i^j = \delta_i^j - \mu_i^j$.

Bilan matière.

On suppose que la température est constante. On écrit un bilan matière pour chaque espèce i, associer une contrainte à chaque équilibre puis fermer le système à l'aide de relations simples impliquant la masse et la quantité de matière de cette espèce. Le bilan matière de l'espèce i s'écrit

$$\frac{\partial n_i}{\partial t} + \sum_{j=1}^{N_r} \alpha_i^j \nu_j V + \sum_{j=1}^{N_e} \beta_i^j \lambda_j = 0,\ \forall i \in [1, N_s]. \tag{3.1}$$

Dans l'équation (3.1) :
• n_i est la quantité de matière de l'espèce i ;
• V est le volume de la solution ;
Connaissant la quantité de matière et le volume on peut déterminer la concentration (molaire) c_i par

$$n_i - c_i V = 0; \tag{3.2}$$

• ν_j est la vitesse de réaction de la réaction j avec $\nu_j = \nu_j((c_i)_{i\in[1,N_s]})$;
• le terme en $\sum_j \alpha_i^j \nu_j V$ représente la vitesse de variation de quantité de matière due aux réactions ;
• β_i^j est le coefficent stoechiométrique de l'espèce chimique i dans l'équilibre j ;
• λ_j est l'avancement de l'équilibre j (c'est une inconnue du système) ;
• le terme en $\sum_j \beta_i^j \lambda_j$ représente la vitesse de variation de quantité de matière due aux équilibres.

On écrit en outre :
• la relation entre la quantité de matière n_i, le volume V et la concentration c_i de l'espèce i soit

$$n_i - c_i V = 0; \tag{3.3}$$

• la relation entre la quantité de matière n_i, la masse de solvant m_S, et la molalité [1] b_i soit

$$n_i - b_i m_S = 0; \tag{3.4}$$

• la relation entre la quantité de matière n_i, la masse m_i, et la masse molaire M_i soit

$$m_i - n_i M_i = 0; \tag{3.5}$$

1. La molalilté est donc la quantité de matière de soluté par kilogramme de solvant.

• la contrainte sur les concentrations (c_i) et molalités (b_i) associée à l'équilibre j soit

$$g_j((c_i),(b_i)) = 0. \tag{3.6}$$

On considère le volume V comme une inconnue de notre système. On se donne donc la masse volumique ρ (supposée constante ici) et on donne la relation qui définit la massse volumique *i.e.*

$$\rho V - \sum_{i=1}^{N_s} m_i = 0 \tag{3.7}$$

Le système s'écrit donc

$$\begin{cases} \dfrac{\partial n_i}{\partial t} + \displaystyle\sum_{j=1}^{N_r} \alpha_i^j \nu_j V + \sum_{j=1}^{N_e} \beta_i^j \lambda_j = 0 & i \in [1, N_e] \\ g_j(c_i, b_i) = 0 & j \in [1, N_s] \\ n_i - c_i V = 0 & i \in [1, N_s] \\ n_i - b_i m_S = 0 & i \in [1, N_s] \\ m_i - n_i M_i = 0 & i \in [1, N_s] \\ \rho = \rho^\star((c_i),(b_i)) \\ \rho V - \sum_{i=1}^{N_s} m_i = 0 \end{cases} \tag{3.8}$$

3.2.2 Schéma numérique

On ramène la résolution de (3.8) à la résolution de

$$\begin{cases} \dfrac{dZ}{dt} = H(Z,Y) \\ G(Z,Y) = 0 \end{cases} \tag{3.9}$$

où F est une appplication non linéaire définie sur un ouvert $\mathcal{O}$ de $\mathbb{R}^n$ contenant Ω à valeurs dans $\mathbb{R}^n$, et $X \in \Omega \subset \mathbb{R}^n$. Supposons que la discrétisation du système (3.9) soit réalisée à l'aide d'un schéma d'Euler implicite. On discrétise la dérivée en temps par une différence finie et on suppose le pas de temps constant. le schéma numérique s'écrit par conséquent, pour tout $n \geq 0$,

$$\begin{cases} \dfrac{Z^{n+1} - Z^n}{\Delta t} = H(Z^{n+1}, Y^{n+1}) \\ G(Z^{n+1}, Y^{n+1}) = 0 \end{cases} \tag{3.10}$$

Posons alors $X = \begin{pmatrix} Z \\ Y \end{pmatrix}$. Le schéma (3.10) se réécrit alors

$$F(X^{n+1}) = 0 \tag{3.11}$$

Le schéma (3.11), implique, à chaque pas de temps, la résolution d'un système non linéaire d'équations dont X^{n+1} est la solution. Cette solution sert de condition initiale au pas de temps suivant. Ce système non linéaire d'équations est résolu à l'aide de la méthode vue dans le chapitre 2, puisque l'on impose des contraintes à certaines des composantes de X^{n+1}, pour tout n. Plus précsiément, la stratégie de résolution numérique du système (3.9) consiste à résoudre à chaque pas de temps

$$\min_{SX \geq 0} \frac{1}{2} \left\| F(X^{n+1}) \right\|^2, \tag{3.12}$$

où $S \in \mathcal{M}_n(\mathbb{R})$ vérifie
• $S_{ii} = 1$ s'il y a une contrainte sur x_i ;
• $S_{ii} = 0$ sinon ;
• $S_{ij} = 0$ si $i \neq j$.

On a trois critères d'arrêt qui stoppent la résolution du système non linéaire à un pas de temps donné. Soit $\epsilon > 0$ un seuil.
• Le premier concerne la norme du pas. L'algorithme de résolution du système non linéaire s'arrête lorsque, à une itération k, nous avons

$$\frac{\|p_k - p_{k-1}\|}{\|p_{k-1}\|} < \epsilon. \tag{3.13}$$

Dans le code, pour éviter de diviser par des quantités pouvant être nulles, on remplace le critère (3.13) par

$$\|p_k - p_{k-1}\| - \epsilon \|p_{k-1}\| < 0. \tag{3.14}$$

• Le deuxième concerne $\|F_k\|$. L'algorithme de résolution du système non linéaire s'arrête lorsque, à une itération k, nous avons

$$\frac{\|F_k - F_{k-1}\|}{\|F_{k-1}\|} < \epsilon. \tag{3.15}$$

Dans le code, pour éviter de diviser par des quantités pouvant être nulles, on remplace également le critère (3.15) par

$$\|F_k - F_{k-1}\| - \epsilon \|F_{k-1}\| < 0. \tag{3.16}$$

• Enfin, on limite le nombre d'itérations au cours d'un pas de temps.
Dans toute la suite, la valeur de ϵ est fixée à 10^{-12}. Par ailleurs, tous les résultats présentés dans ce chapitre ont été obtenus avec la matrice de Coleman et Li définie au §B.1 page 56.

3.3 L'environnement informatique

3.3.1 L'environnement Trio_U

Le code Trio_U est un code de CFD (Computational Fluid Dynamics), orienté objet et massivement parallèle pour des applications du domaine nucléaire. Ici, on ne développe pas directement dans le code, mais on utilise quelques classes pour implémenter le schéma (3.10).

3.3.2 L'approche utilisée pour ce problème

On utilise donc l'apporche orientée objet pour ce problème et leurs applications.
• On construit des classes pour définir le type de problème à résoudre (*i.e.* on construit des classes de sorte de pouvoir résoudre $f(t, y, y') = 0$.), le schéma en temps (on n'utilise ici qu'un schéma d'Euler implicite implémenté avec la classe SIDES_Timestepper_Backward_Euler),
• On implémente également des classes pour la résolution de (2.1) à chaque pas de temps Pour cela on met en place une structure d'héritage comme suit :
- On construit une classe "de base" SIDES_Solver_base [2]
- On en fait hériter deux classes : l'une est associée à l'implémentation de la méthode de Newton (il s'agit de la classe SIDES_Solver_Newton) et l'autre à l'implémentation d'une méthode de région de confiance (il s'agit de la classe SIDES_Trust_Region_Solver_base).
- On fait hériter de la classe SIDES_Trust_Region_Solver_base une classe pour la résolution de systèmes non linéaires sans contrainte (il s'agit de la classe Solver_Dogleg), ou avec contraintes (la classe Constrained_Dogleg_Solver).

• On implémente une autre classe pour la gestion des frontières de la boîte Ω dans le cas contraint.

D'autre part pour tester les méthodes de la classe Constrained_Dogleg_Solver :
• on utilise un fichier d'entrée comprenant toutes informations nécessaires sur les constituants du mélange réactif étudié, les réactions entre les constituants, le type de problème que l'on souhaite résoudre, ainsi que les paramètres nécessaires pour faire fonctionner le solveur.
• il y a plusieurs fichiers de sortie une fois la résolution terminée, en particulier le fichier avec l'extension .itr donnant à chaque pas de temps l'évolution de la norme du pas et du résidu en fonction du nombre

2. Le préfixe "SIDES" correspond au nom du projet dans lequel ce stage s'inscrit.

d'itérations.

3.4 Étude de cas-test

3.4.1 Un premier cas-test

Pour ce premier exemple, on considère un système de sorte qu'il y ait 28 inconnues (et donc une matrice jacobienne de taille 28×28). Pour vérifier le bon ou mauvais fonctionnement du code on trace pour chaque étude les courbes d'évolution de la norme de l'incrément et et du résidu (*i.e.* $\|p_k\|$ et $\|F_k\|^2$ en fonction de k, avec une échelle logarithmique pour ces deux quantités). Dans la suite, la norme de l'incrément est en fait la norme du pas de dogleg.

Les équations.

Ce cas-test ne considère que les cinétiques. Ainsi, pour une espèce chimique i, le terme $\sum_j \beta_i^j \lambda_j$ est nul.

Ceci entraîne que l'on ne peut pas non plus écrire une relation de type (3.6). Il s'ensuit que (3.8) se réécrit

$$\begin{cases} \dfrac{\partial n_i}{\partial t} + \sum_j \alpha_i^j \nu_j V = 0 \\ n_i - c_i V = 0 \\ n_i - b_i m_S = 0 \\ m_i - n_i M_i = 0 \\ \rho = \rho^\star((c_i),(b_i)) \\ \rho V - \sum_{i=1}^{N_s} m_i = 0 \end{cases} \tag{3.17}$$

Premier test du solveur CoDoSol.

On souhaite effectuer des calculs avec les données fournies par le tableau 3.1.

θ	0.95
β	0.25
α_k	1.0
type de région de confiance	elliptique
facteur d'agrandissement	4.0
facteur de contraction	0.25
Δ_{min}	10^{-6}

TABLE 3.1 – Valeurs utilisées pour la résolution du système non linéaire par le solveur CoDoSol - premier cas test.

Les résultats sont présentés sur la figure 3.2.

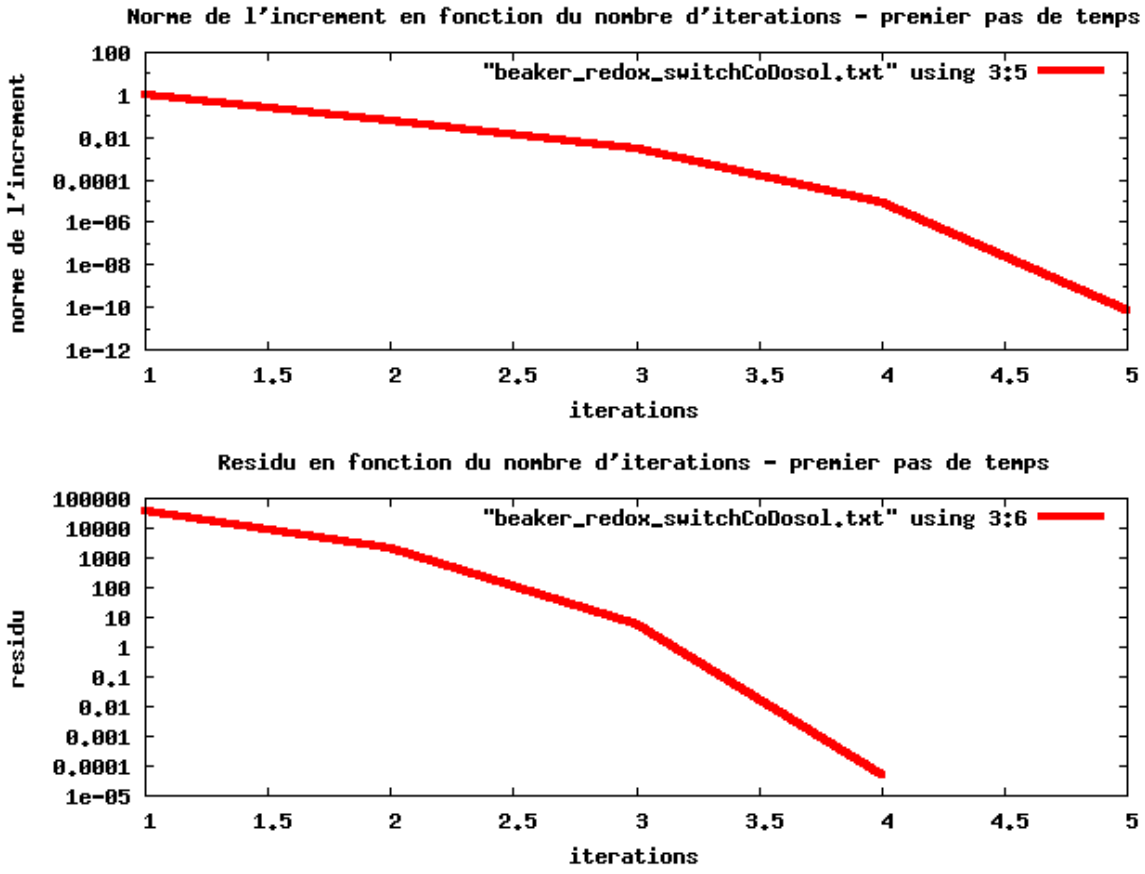

FIGURE 3.2 – Norme de l'incrément et du résidu en fonction du nombre d'itérations - premier cas test.

Sur la figure 3.2, la valeur du résidu pour la dernière itération effectuée n'est pas affichée car celle-ci se révèle en fait négative. Ceci signifie que le critère de convergence de l'algorithme est vérifié. Le critère de convergence de l'algorithme est donc atteint en cinq itérations pour ce cas test-là. Nous allons tester cet algorihme de nouveau, mais en faisant varier la taille minimale du rayon de la région de confiance (voir ci-après).

• **Influence de la taille minimale du rayon de la région de confiance**. Dans l'algorithme qui calcule le pas d'essai, la valeur minimale du rayon de la région de confiance est essentielle. En effet, plus il y a de rejets de pas d'essai, et plus le rayon de la région de confiance est réduit. On peut donc imaginer que changer la valeur minimale du rayon de la région de confiance impacte sur la convergence de la méthode. Les paramètres utilisés pour le solveur Constrained Dogleg sont

θ	0.95
β	0.25
α_k	1.0
type de région de confiance	elliptique
facteur d'agrandissement	4.0
facteur de contraction	0.25
Δ_{min}	10^{-j}, $j \in \{6, 7, 8, 9\}$

TABLE 3.2 – Valeurs utilisées pour la résolution du système non linéaire par le solveur CoDoSol, avec Δ_{min} variable - premier cas test.

Après avoir effectué les mêmes tracés que sur la figure 3.2, on obtient des courbes avec des tendances sensiblement proches pour $\Delta_{\text{min}} \in 10^{-7}, 10^{-8}, 10^{-9}$. Pour ce cas test, il s'avère que la valeur minimale du

rayon de la région de confiance n'a pas ou peu d'influence sur les résultats.
• **Influence du type de la région de confiance**. Les paramètres utilisés pour le solveur Constrained Dogleg sont dans le tableau 3.3.

θ	0.95
t	0.25
α_k	1.0
Δ_{min}	10^{-6}
type de région de confiance	elliptique, sphérique
facteur d'agrandissement	4.0
facteur de contraction	0.25

TABLE 3.3 – Valeurs utilisées pour la résolution du système non linéaire par le solveur CoDoSol, pour deux types de région de confiance - premier cas test.

Pour les régions de confiance du tableau 3.3, on trace sur la figure 3.3 les courbes d'évolution de la norme de l'incrément et et du résidu (Pour une région de confiance ellipitique, on se reportera à la figure 3.2).

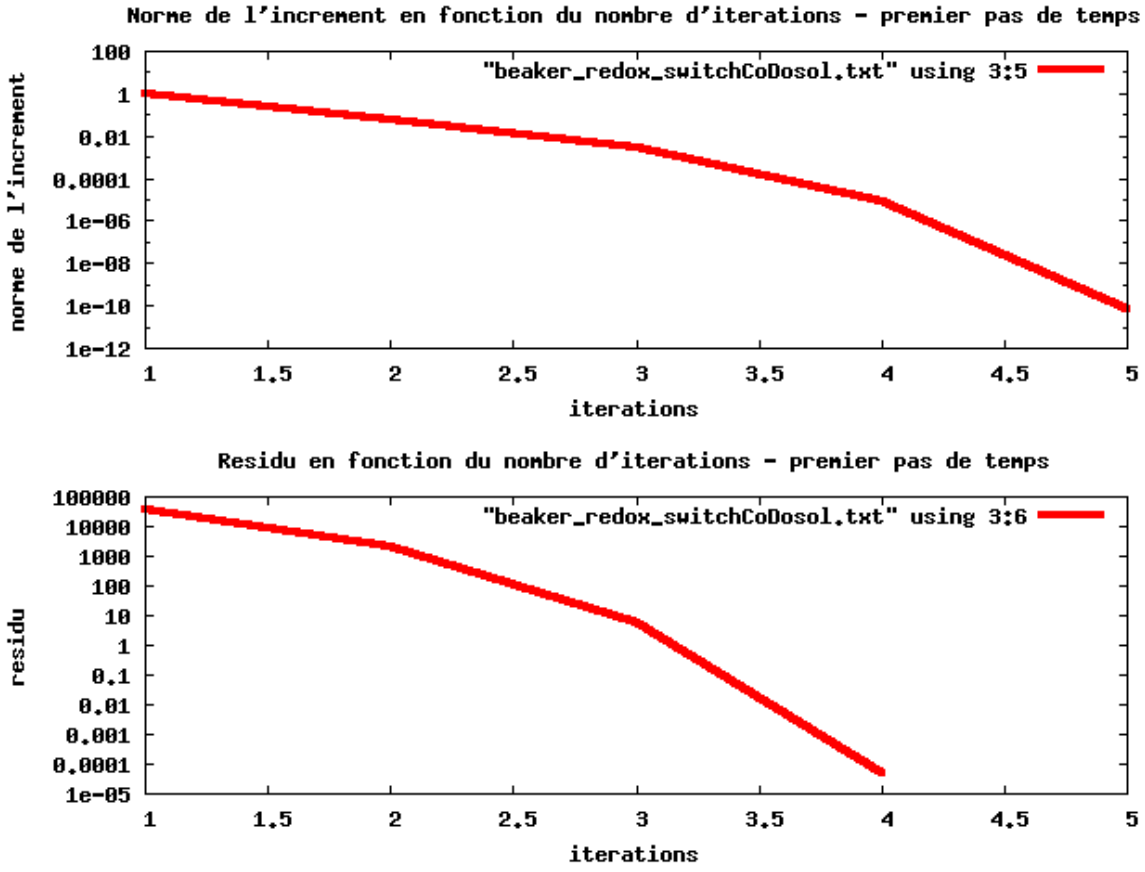

FIGURE 3.3 – Norme de l'incrément et du résidu en fonction du nombre d'itérations pour avec une région de confiance sphérique - premier cas test.

Peu importe le type de la région de confiance considéré dans cette étude, on obtient pour la figure 3.3 des résultats quasi-identiques à ceux obtenus sur la figure 3.2. On peut donc indifféremment choisir, pour les paramètres fournis par le tableau 3.3 une région de confiance sphérique ou ellipitique.

3.4.2 Un deuxième cas-test

Les équations.

Ce cas-test ne considère que des équilibres et la matrice jacobienne est d'ordre 154×154. Ainsi, pour une espèce chimique i, le terme $\sum_j \alpha_i^j \nu_j V$ est nul. Il s'ensuit que (3.8) se réécrit

$$\begin{cases} \dfrac{\partial n_i}{\partial t} + \sum_j \beta_i^j \lambda_j = 0 \\ g_j((c_i),(b_i)) = 0 \\ n_i - c_i V = 0 \\ n_i - b_i m_S = 0 \\ m_i - n_i M_i = 0 \\ \rho = \rho^\star((c_i),(b_i)) \\ \rho V - \sum_{i=1}^{N_s} m_i = 0 \end{cases} \tag{3.18}$$

On va voir dans cette sous-section que le programme qui assure la non-négativité de la solution fonctionne mal dans certains cas (en l'occurrence ce cas test). Les modifications suivantes ont été apportées.

Modification de x_0.

D'après le théorème de convergence de l'algorithme Constrained Dogleg (*cf.* théorème 3), la convergence vers un point de minimum de $\frac{1}{2}\|F(x)\|^2$ ne dépend pas de x_0. On est donc libres de choisir le point x_0 que l'on veut, tant que $x_0 \in \text{int}(\Omega)$ (vu que l'on développe ici une méthode de *point intérieur*). Il se trouve que pour ce cas-test, la condition initiale est telle que certaines composantes de x_0 sont sur la frontière de Ω[3]. L'idée est donc d'"éloigner" les composantes de x_0 qui sont déjà sur la frontière, afin d'éviter de "stagner" sur celles-ci. On propose donc pour ce faire l'algortihme[4]suivant :
Données : $\epsilon > 0$, une valeur d'incrément $d > 0$, le point initial non modifié x_0 et $l \in \mathbb{R}^n$ (*resp.*$u \in \mathbb{R}^n$)le vecteur des bornes inférieures (supérieures) On ajoute que les indices des composantes pour lesquelles on a une borne inférieure et une borne supérieure sont rangés dans deux tableaux.

3. Il n'y a pas de borne supérieure pour ce cas-test.
4. L'algorithme est ici écrit dans le cas général même s'il n'y pas de borne supérieue ici.

Pour i **de** 1 **à** size(l) **faire**
  Calculer $I1$, numéro de la composante de la $i^{\text{ème}}$ borne inférieure. Calculer $e = |(x_0)_{I1} - l_i|$;
  Calculer tol $= \epsilon \max(|(x_0)_{I1}|, |l_i|)$;
  Si (Si $e \leq$ tol) **Alors**
    Pour j **de** 1 **à** size(u) **faire**
      Calculer $I2$, numéro de la composante de la $j^{\text{ème}}$ borne supérieure.
      Si ($I1 = I2$) **Alors**
        $(x_0)_{I1} = (x_0)_{I1} + \frac{1}{2}(l_i + u_j)$
      Sinon
        $(x_0)_{I1} = (x_0)_{I1} + d$
      Fin Si
    Fin Pour
  Fin Si
Fin Pour
Pour i **de** 1 **à** size(u) **faire**
  Calculer $I1$, numéro de la composante de la $i^{\text{ème}}$ borne supérieure. Calculer $e = |(x_0)_{I1} - u_i|$;
  Calculer tol $= \epsilon \max(|(x_0)_{I1}|, |u_i|)$;
  Si (Si $e \leq$ tol) **Alors**
    Pour j **de** 1 **à** size(l) **faire**
      Calculer $I2$, numéro de la composante de la $j^{\text{ème}}$ borne supérieure.
      Si ($I1 = I2$) **Alors**
        $(x_0)_{I1} = (x_0)_{I1} + \frac{1}{2}(l_j + u_i)$
      Sinon
        $(x_0)_{I1} = (x_0)_{I1} + d$
      Fin Si
    Fin Pour
  Fin Si
Fin Pour

Algorithme 6: Modification du vecteur initial

Pour ce cas-test, la deuxième boucle pour n'est jamais exécutée.

Modification de la projection du pas de Cauchy.

Nous avons vu dans le §2.2.3 la méthode de projection de $p_c(\Delta_k)$ lorsque $x_k + p_c(\Delta_k) \notin \Omega$. C'est une méthode de projection "globale" car on applique le même coefficient $\theta\lambda_k$, où $\theta \in [0, 1]$ et λ_k est défini par (2.5). Ici l'on va projeter le pas de Cauchy de la même manière que l'on projette le pas de Newton via (2.3). Autrement dit, on définit

$$\bar{p_c}(\Delta_k) = \alpha_k(P(x_k + p_c(\Delta_k)) - x_k). \tag{3.19}$$

Une autre projection du pas de Cauchy.

Plutôt que d'appliquer le même coefficient λ_k à toutes les composantes du pas de Cauchy, on va calculer une valeur λ_i par composante ce qui donne le (petit) algorithme suivant. Soit n le nombre de composantes de $p_c(\Delta_k)$
Pour $i = 1, \cdots, n$:

1. $(p_c(\Delta_k))_i = \theta\lambda_i(p_c(\Delta_k))_i$, avec λ_i défini par (2.4) ;

Étude paramétrique.

On fera varier les paramètres suivants :
- la taille minimale de la région de confiance ;
- le type de région de confiance (elliptique ou sphérique) ;
- le coefficient du test de la condition d'amélioration ;
- le coefficient d'agrandissement du rayon de la région de confiance.

• **Influence de la taille minimale du rayon de la région de confiance**. Les paramètres utilisés pour le solveur Constrained Dogleg sont présentés dans le tableau 3.4.

θ	0.95
β	0.25
α_k	1.0
type de région de confiance	elliptique
facteur d'agrandissement	4.0
facteur de contraction	0.25
Δ_{min}	10^{-j}, $j \in \{6, 7, 8, 9\}$

TABLE 3.4 – Valeurs utilisées pour la résolution du système non linéaire par le solveur CoDoSol, avec Δ_{min} variable - second cas test.

On obtient les résultats sur la figure 3.4.

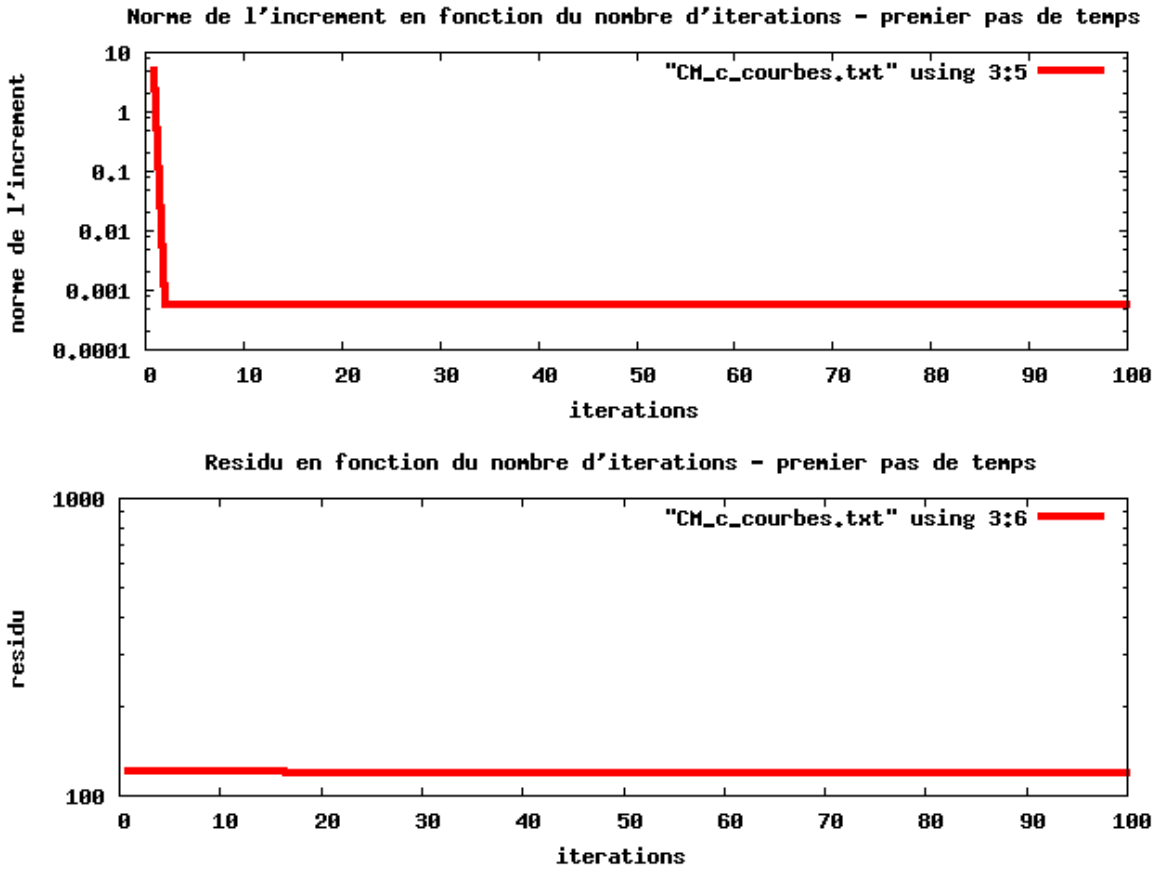

FIGURE 3.4 – Norme de l'incrément et du résidu en fonction du nombre d'itérations pour $\Delta_{\text{min}} = 10^{-6}$ - second cas test.

Il s'avère qu'après calculs et post-traitemenbts, les résultats obtenus pour $\Delta_{\text{min}} \in \{10^{-7}, 10^{-8}, 10^{-9}\}$ sont très très proches de ceux présentés sur la figure 3.4, il n'y aurait donc pas d'influence notable du choix d'une valeur minimale du rayon de la région de confiance. Il m'a donc paru inutile de faire figurer les résultats pour les autres valeurs de β choisies. On ajoute que pour ces calculs, le nombre maximal d'itérations possibles par pas de temps a été atteint et de fait, la norme du résidu n'a pas atteint une valeur suffisament faible pour que l'algorithme s'arrête avant d'atteindre ce seuil.

Notons aussi les fluctuations d'ordre de grandeur pour la norme du pas de dogleg et notons que ces fluctuations se poursuivent jusqu'à la fin de la simulation numérique. Malheureusement, il semble très peu probable qu'avec les données d'entrée du tableau 3.4, et le jeu de données de ce cas-test, on puisse parvenir à faire converger la méthode numérique.

• **Influence du type de la région de confiance**. Les paramètres utilisés pour le solveur Constrained Dogleg sont présentées dans le tableau 3.5.

θ	0.95
β	0.25
α_k	1.0
Δ_{min}	10^{-6}
type de région de confiance	elliptique, sphérique
facteur d'agrandissement	4.0
facteur de contraction	0.25

TABLE 3.5 – Valeurs utilisées pour la résolution du système non linéaire par le solveur CoDoSol, pour deux types de région de confiance - second cas test.

Les courbes obtenues sont tracées sur les figures 3.5 et 3.6.

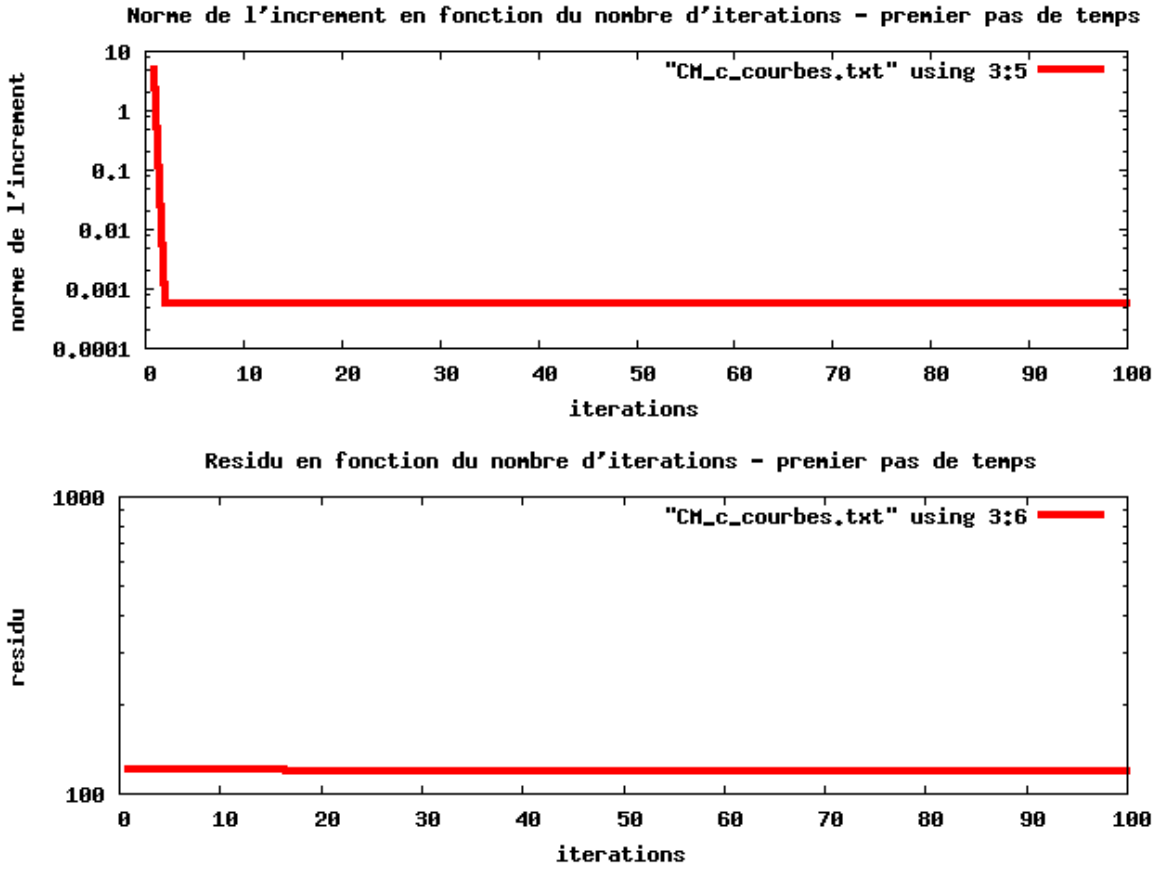

FIGURE 3.5 – Norme de l'incrément et du résidu en fonction du nombre d'itérations avec une région de confiance elliptique - second cas test.

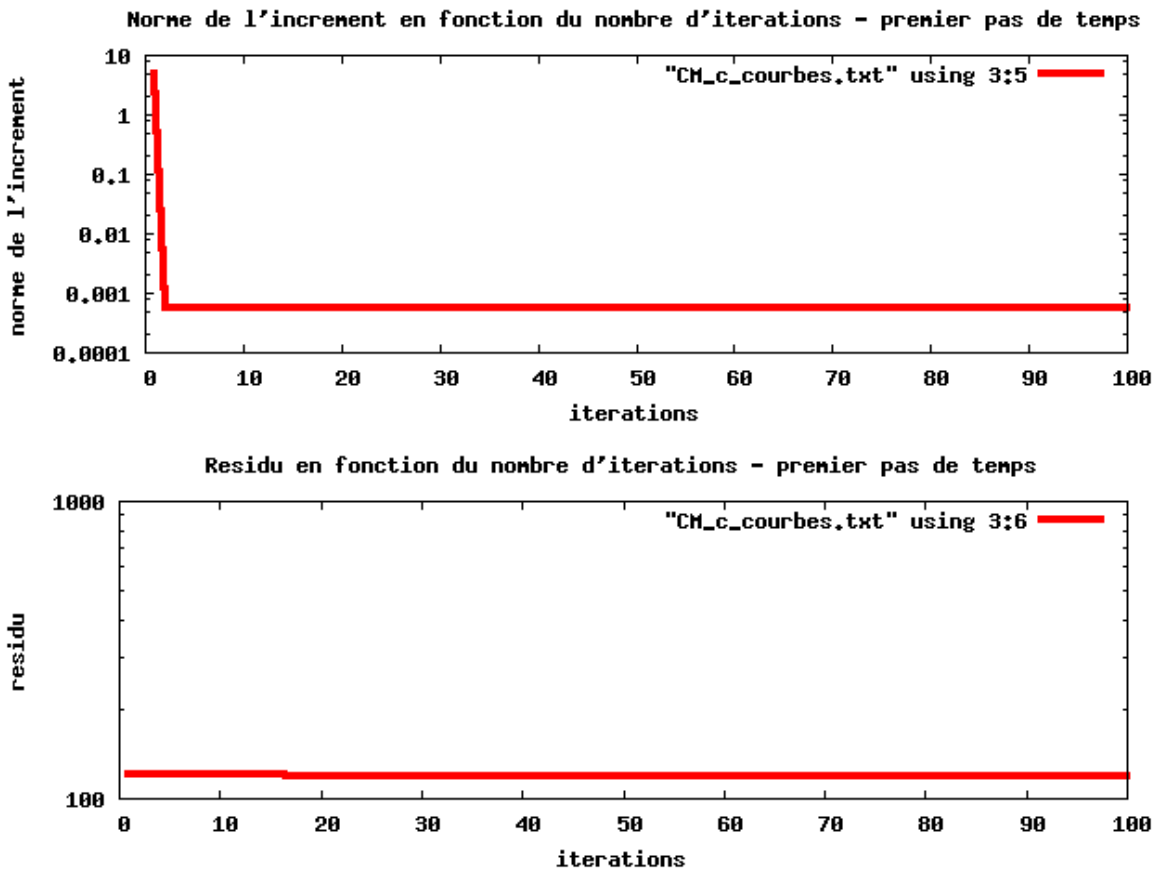

FIGURE 3.6 – Norme de l'incrément et du résidu en fonction du nombre d'itérations pour avec une région de confiance sphérique - second cas test.

Dans [1], il est mentionné que dans la plupart des cas-test, on obtient de meilleurs résultats pour une région de confiance ellpitique. Malgré cela et vu les résultats peu satisfaisants obtenus, on a voulu voir si le cas-test était sensible au type de région de confiance rentré dans les jeu de données. Il semble que pour ce cas-test, ce ne soit pas le cas (*cf.* figures 3.5 et 3.6).
• **Influence du coefficient** β. Les paramètres utilisés pour le solveur Constrained Dogleg sont précisés dans le tableau 3.6.

θ	0.95
β	0.55, 0.65, 0.75, 0.85
α_k	1.0
$\Delta_{\min}$	10^{-6}
type de région de confiance	elliptique
facteur d'agrandissement	4.0
facteur de contraction	0.25

TABLE 3.6 – Valeurs utilisées pour la résolution du système non linéaire par le solveur CoDoSol, pour quatre valeurs du coefficient β - second cas test.

Les courbes obtenues sont tracées sur la figure 3.7.

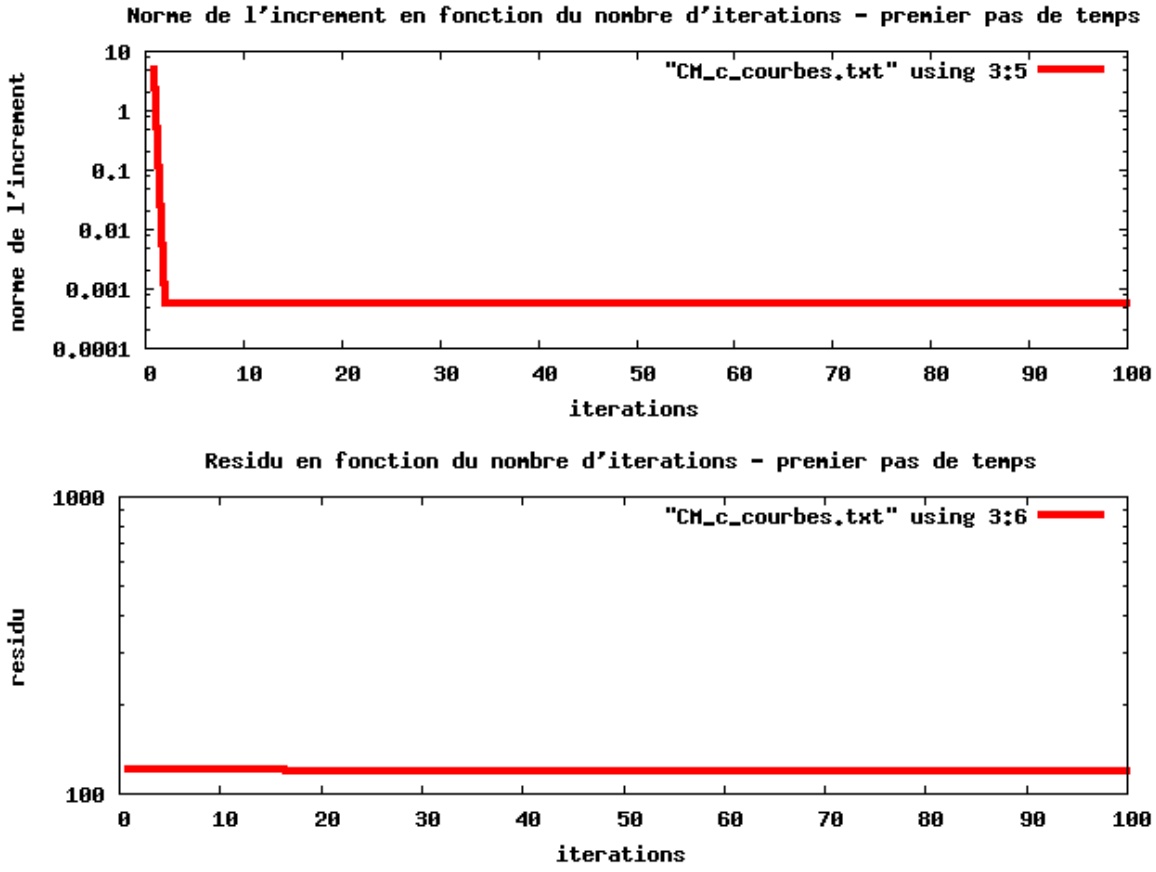

FIGURE 3.7 – Norme de l'incrément et du résidu en fonction du nombre d'itérations pour $\beta = 0.55$ - second cas test.

Lorsque l'on fait varier le coefficient β, on est plus ou moins sélectif sur la manière dont on va choisir le pas. En prenant quatre valeurs de ce coefficient, on se rend compte aussi que celui-ci ne change pas la tendance des courbes. Il m'a donc aussi paru inutile de faire figurer les résultats pour les autres valeurs de β choisies.
• **Influence du facteur d'agrandissement**. Les paramètres utilisés pour le solveur Constrained Dogleg

sont précisés dans le tableau 3.7.

θ	0.95
β	0.25
α_k	1.0
Δ_{min}	10^{-6}
type de région de confiance	elliptique
facteur d'agrandissement	3.0, 4.0, 5.0, 6.0
facteur de contraction	0.25

TABLE 3.7 – Valeurs utilisées pour la résolution du système non linéaire par le solveur CoDoSol, pour plusieurs valeurs du facteur d'agrandissement - second cas test.

Les courbes obtenues sont tracées sur les figures 3.8, 3.9, 3.10 et 3.11.

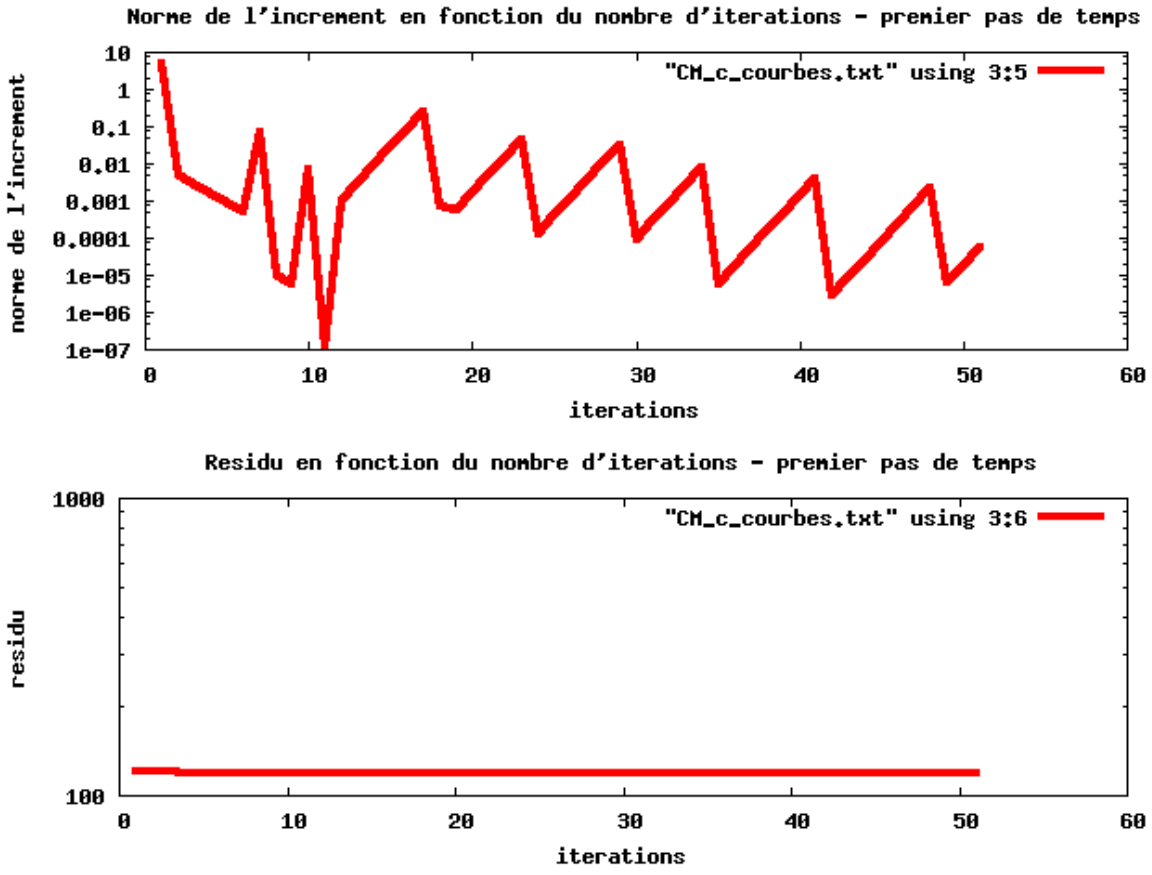

FIGURE 3.8 – Norme de l'incrément et du résidu en fonction du nombre d'itérations pour un facteur d'agrandissement de 3 - second cas test.

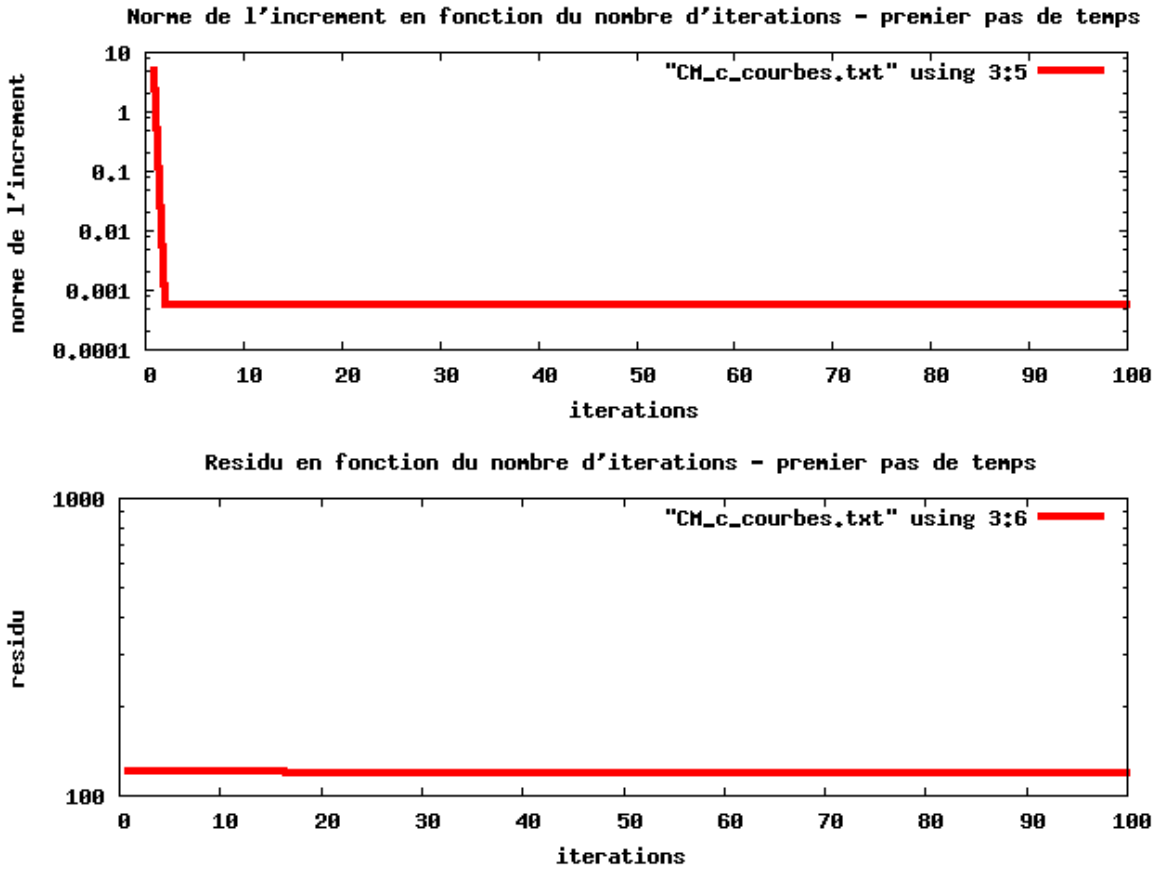

FIGURE 3.9 – Norme de l'incrément et du résidu en fonction du nombre d'itérations pour facteur d'agrandissement de 4 - second cas test.

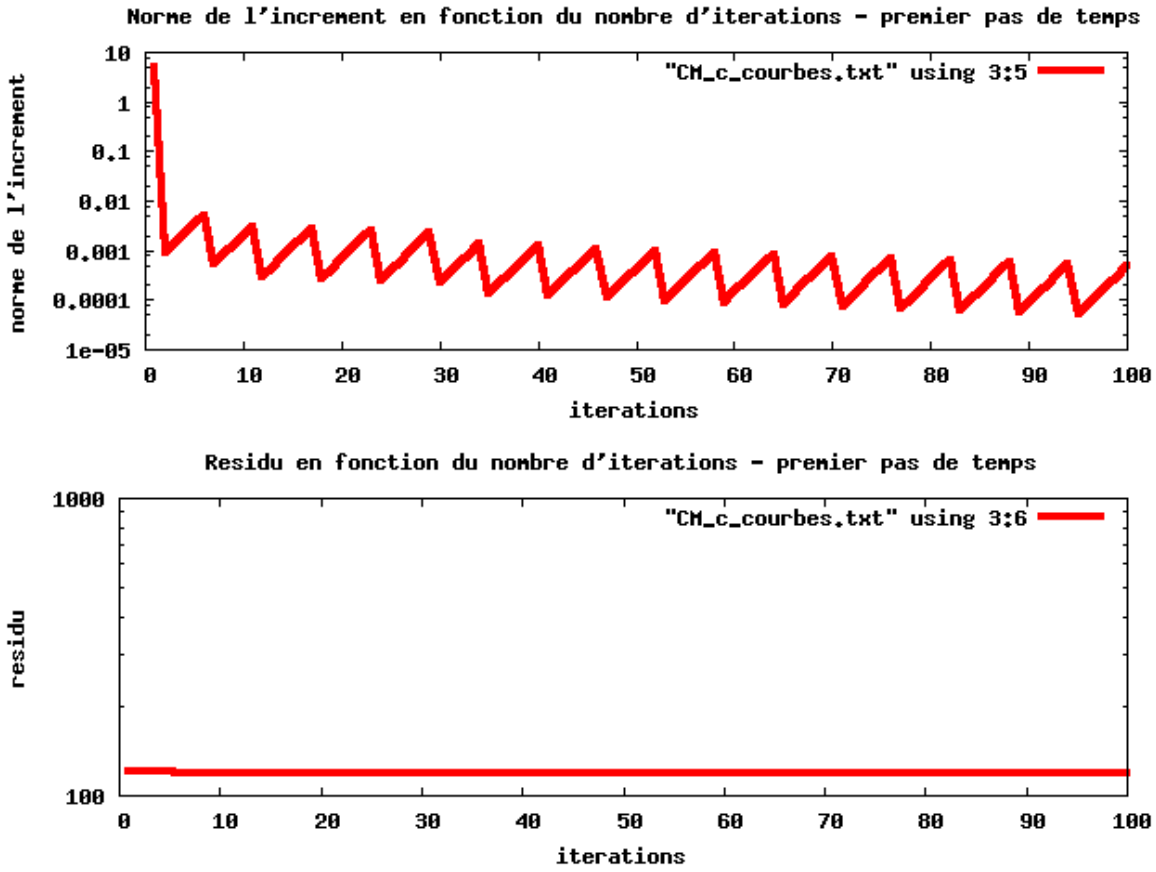

FIGURE 3.10 – Norme de l'incrément et du résidu en fonction du nombre d'itérations pour facteur d'agrandissement de 5 - second cas test.

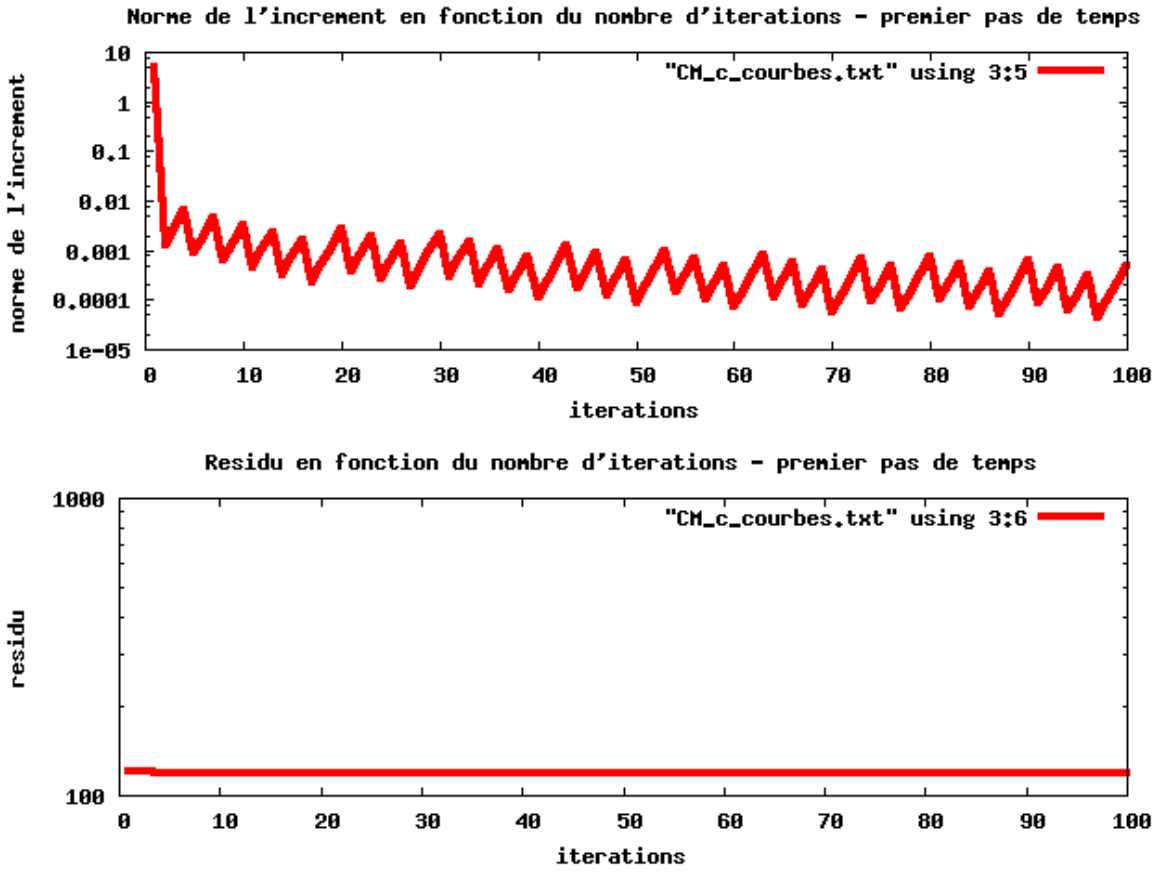

FIGURE 3.11 – Norme de l'incrément et du résidu en fonction du nombre d'itérations pour facteur d'agrandissement de 6 - second cas test.

Le fait de faire varier le facteur d'agrandissement du rayon de la région de confiance, semble pour ce cas-test avoir un effet sur la norme du pas de dogleg comme on le voit sur les figures 3.8, 3.9, 3.10 et 3.11. Il semblerait que le facteur d'agrandissement fasse diminuer la norme du pas de dogleg, sans pour autant diminuer la norme du résidu. On notera aussi que pour les quatre résultats obtenus, le calcul s'est arrêté car le seuil maximal du nombre d'itérations que l'on a autorisé par pas de temps a été atteint (sauf le cas de la figure 3.8). Au niveau du pas de dogleg, les comportements sont également très différents d'une valeur de ce facteur d'agrandissement à l'autre. Il me paraît difficile de saisir la sensibilité des réusltats vis-à-vis de ces paramètres.

Conclusion

Le sujet sur lequel j'ai travaillé pendant plus de cinq mois se révèle être à la fin bien plus complexe et bien plus vaste à la fois. Il ne suffisait pas ici de développer quelques méthodes, de les tester et d'observer les résultats. L'importance des nombreux paramètres qui sont à la fois dans l'article et dans le code et la manière de les piolter constitue à elle seule une difficulté pour ma part assez importante. Une autre difficulté a été de comprendre pourquoi la méthode Constrained Dogleg prévalait devant d'autres méthodes permettant de résoudre ce type de problèmes. Ce stage nécessitait par ailleurs un bon esprit de sythèse vu la masse d'information recueillie sur une période assez courte et le travail de sélection des informations pour pour bien comprendre l'essentiel de la méthode.

Il est bien évident que l'aspect informatique a une part prépondérante dans ce travail. On ajoutera que la programmation orientée objet et surtout l'héritage multiple semnle être la voie la moins difficile pour implémenter la méthode Constrained Dogleg. Comme lors de mes expériences précédentes, j'ai dû rentrer dans un code, comprendre la logique (surtout pour l'héritage des classes) et surtout proposer des méthodes, ce qui était nouveau pour moi, ayant été principalement (et notammment lors de mon stage précédent) du côté utilisateur. Ainsi, j'ai pu grâce à ce stage, prorgesser dans ce domaine.

Ma formation étant principalement axée sur les méthodes numériques et l'analyse numérique des équations aux dérivées partielles "classiques". Je connaissais peu voire pas du tout cette méthode de résolution de systèmes d'équations non-linéaires par minimisation de fonctionnelle. Elle fait appel à des méthodes d'optimisation que je ne connaissais pas du tout jusqu'ici. Ce stage m'a donc permis de les comprendre et bien entendu de les utiliser. Cette stratégie, bien qu'elle semble assez lourde à implémenter et à tester, peut s'avérer payante et permet d'obtenir des résultats satisfaisants (*cf.* le premier cas-test traité dans ce rapport.).

Annexe A

Constrained Dogleg Solver (CoDoSol)

Cette annexe présente la version MATLAB du code qui permet de déterminer

$$\text{argmin}_{x\in\Omega}\frac{1}{2}\|F(x)\|^2$$

```
function [sol,ierr,output,history,grad,diagnostic]=CoDoSol(x,F,l,u,tol,parms,Fjac)
%
%   Globally convergent  solver for constrained nonlinear
%   systems of equations
%
%           F(x)=0          l<=x<=u
%
%   where  F: R^n --> R^n.
%   The algorithm combines Newton method and a (either spherical or elliptical)
%   trust-region procedure. The merit function used is f(x)=norm(F(x)).
%   The elliptical trust-region is defined employing the scaling
%   diagonal matrix D used to compute the scaled gradient.
%   The trust-region problem is approximately solved by a "constrained dogleg" method.
%   Strictly feasible iterates are generated.
%   For more details, see
%
%        S. Bellavia, M. Macconi, S. Pieraccini, "Constrained Dogleg methods
%              for nonlinear systems with simple bounds",
%              COAP,  Vol. 53, pp. 771-794, 2012
%
%   Uses  dmatrici.m, diffjac.m
%
%   The problem to be solved has to be provided defining the following function(s):
%
%           - function_name.m
%                function [y] = function_name(x)
%                Input:
%                   x = point in which the function has to be evaluated
%                Output:
%                   y = F(x)
%           - Jac_function_name.m (optional)
%                function [J] = Jac_function_name(x)
%                Input:
%                   x = point in which the Jacobian has to be evaluated
%                Output:
%                   J = F'(x)
%
%
%    Further, the following function may be provided if the user would like to
%    apply his own scaling matrix:
%
%           - usersd.m (optional)
%                function [d] = usersd(x,grad,l,u)
%                Input:
%                   x = point in which the scaling matrix has to be evaluated
```

```
%                grad = gradient of f, preevaluated
%                 l,u = constraints
%                Output:
%                   d = array such that d=diag(D(x)).
%
%
%    REMARK: If the user would like to work with SPARSE MATRICES, and
%            function Jac_function_name.m is not provided, it is only required that function
%            function_name.m provides sparse output.
%            Then, the approximated Jacobian will be automatically computed as a
%            sparse matrix. If function Jac_function_name.m is provided by the user, then also
%            its output has obviously to be a sparse matrix.
%            When the Jacobian is stored as a sparse matrix, the Newton step is
%            computed via the built-in Matlab function LU with the syntax as to use UMFPACK.
%
%
%
%
%    Typical usages:
%
%    function [sol,ierr,output]=CoDoSol(x,F,l,u,tol,parms)
%    ------------------------------------------------------
%
%    The Jacobian matrix of F is approximated using
%    finite-differences.
%
%    Input:
%
%      x     = initial iterate x_0.
%      F     = function that accepts input vector x and returns the
%              vector F(x).
%      l,u   = vectors containing the lower and upper bounds on the
%              variables.
%              set l(i) = -Inf if x(i) is unbounded below;
%              set u(i) = Inf if x(i) is unbounded above.
%      tol   = [atol,rtol] absolute/relative error tolerances used
%              in the stopping criterion:
%                     norm(F(x))<=atol+rtol*norm(F(x_0)).
%            =[] the default value [1.0e-6,0] is used.
%      parms = [maxit,maxnf,tr,delta,scaling,outflag]
%              maxit=  maximum number of nonlinear iterations;
%              maxnf=  maximum number of F-evaluations (F-evaluations
%                      due to the Jacobian approximation are not included);
%                 tr= choice of the trust-region type
%                       0  Spherical trust-region (G=I)
%                       1  Elliptical trust-region (G=D^(-1/2))
%              delta=  choice of the initial trust region radius Delta.
%                      -1  then Delta=1;
%                      -2  then Delta=norm(G(x_0)^(-1)grad(f(x_0)));
%                      >0  then Delta=delta.
%            scaling= choice of the scaling matrix
%                      <0   Kanzow-Klug scaling matrix, with gamma=-scaling
%                       0   Coleman-Li scaling matrix
%                       1   Hager et al. scaling matrix
%                       2   User supplied scaling matrix
%            outflag=  printlevel
%                      0   The final summary output is printed:
%                             - number of performed iterations,
%                             - number of performed F-evaluations (F-evaluations due to
%                               the Jacobian approximation are not included);
%                             - norm(F(sol))
%                      >0  one line of summary output for each iteration is printed:
%                             - iteration number k;
%                             - norm(F(x_k));
%                             - number of trust region radius reductions;
%                             - value of gamma used in the path computation;
%                             - ratio of two successive nonlinear residuals;
```

```
%
%          =[] the default value [300 1000 1 -1 0 2] is used.
%
%   Output:
%     sol    =   final estimate of the solution.
%     ierr   = error indicator:
%               0   upon successful termination;
%               1   the limiting number of iterations has been reached;
%               2   the limiting number of F-evaluations has been reached;
%               3   the trust region radius Delta has become too small
%                   (Delta<sqrt(eps));
%               4   no improvement for the nonlinear reasidual could be obtained:
%                   abs(norm(F(x_k))-norm(F(x_{k-1})))<=100*eps*norm(F(x_k));
%               5   the sequence has approached a minumum of f in the box:
%                   norm(D(x_k)*grad(f(x_k)))<100*eps.
%               6   an overflow would be generated when computing the scaling
%                   matrix D since the sequence is approaching a bound.
%     output =   vector containing:
%                         - number of performed iterations;
%                         - number of performed F-evaluations (F-evaluations due to
%                                       the Jacobian approximation are not included);
%                         - norm(F(sol));
%                         - norm(D(sol)grad(f(sol)));
%                         - total number of reductions of the trust region radius.
%
%
%
%
%  function [sol,ierr,output]=CoDoSol(x,F,l,u,tol,parms,Fjac)
%  ----------------------------------------------------------
%
%  Solves as above with the Jacobian matrix of F evaluated analitically
%  by the user-supplied function Fjac. Function Fjac(X) must
%  returns the Jacobian matrix of the function F evaluated at x.
%
%
%
%
%  function [sol,ierr,output,history]=CoDoSol(x,F,l,u,tol,parms,..)
%  ----------------------------------------------------------------
%
%  Also returns a matrix named history that describes the convergence
%  history of CoDoSol. Each row of history contains:
%                         - iteration number k;
%                         - norm(F(x_k));
%                         - number of  trust region radius reductions;
%                         - value of gamma used in the path computation;
%                         - ratio of two successive nonlinear residuals.
%
%
%
%
%  function [sol,ierr,output,history,grad]=CoDoSol(x,F,l,u,tol,parms,..)
%  ---------------------------------------------------------------------
%
%  Also returns the gradient grad of the merit function f at sol.
%
%
%
%
%  function [sol,ierr,output,history,grad,diagnostic]=CoDoSol(x,F,l,u,tol,parms,..)
%  --------------------------------------------------------------------------------
%
%  If full matrices are used, also returns some diagnostic information:
%                 diagnostic(1)=rank of the Jacobian matrix at sol,
%                               computed by the Matlab function rank;
%                 diagnostic(2:n+1)=singular values of the Jacobian
```

```
%                                   matrix at sol, computed by the Matlab
%                                   function svd.
%
%
%
%  function [sol,ierr,output]=CoDoSol(x,F,l,u,tol)
%  function [sol,ierr,output]=CoDoSol(x,F,l,u)
%  ------------------------------------------------------------
%
%  The following default values are used for tol and/or parms (the same applies in general if
%  tol and/or parms are assigned empty values):
%
%             tol:
%                absolute tolerance = 1.0e-6
%                relative tolerance = 0
%           parms:
%                300 as maximum number of nonlinear iterations
%                1000 as maximum number of function evaluations
%                elliptical trust-region
%                Initial trust-region radius Delta_0=1
%                Coleman-Li scaling matrix
%                one line of summary output for each iteration is printed
%
%  Internal parameters:
%
%   t=0.25                : used for accuracy requirements;
%   theta1=0.99995,sigma=0.99995 : used to ensure strictly feasible iterates;
%   delta1=0.25,delta2=2         : used to update the trust-region size;
%   w=0.75                       : parameter that governs the increase of the
%                                  trust-region size.
%


%
%   INITIALIZATION
%

n=length(x);
ierr=0; nridut=0;
itc=0;
nvf=0;

if any(x<=l|x>=u)
   disp('ATTENTION: the initial guess is not strictly feasible')
   return
end

fx=feval(F,x);
nvf=nvf+1; fnrm=norm(fx);
fnrm2=fnrm^2;
%
%   DEFAULT VALUES FOR TOL AND/OR PARMS
%
if nargin <=4 | isempty(tol)
   tol = [1.0e-6  0];
end
if nargin <=5 | isempty(parms)
   parms = [300 1000 1 -1 0 2];
end

%
%   PARAMETER SETTINGS
%
epsilon=100*eps;
Deltamin=sqrt(eps);
atol=tol(1); rtol=tol(2); stoptol=atol+rtol*fnrm;
maxit=parms(1); maxnf=parms(2); tr=parms(3); Delta=parms(4); scal=parms(5); outflag=parms(6);
```

```
%
%   INTERNAL PARAMETERS
%
r=0.1; t=0.25; w=0.75; delta1=0.25; delta2=2; thetal=0.99995; sigma=0.99995;
%
%   INITIAL OUTPUT
%
if nargout>=4
     history(1,:)=[itc,fnrm,0,0,0];
end
if outflag>0
    disp(sprintf('\n'))
    disp(sprintf('%s it %s ||F||_2 %s rid_step %s gamma %s ratio',...
                  blanks(3),blanks(4),blanks(2),blanks(2),blanks(7)))
end
%
%   ITERATION
%
count_bb=0;
lambda=0;
grad=ones(n,1);
while (fnrm>stoptol & itc<maxit & nvf<maxnf)
     itc=itc+1;
     fnrm0=fnrm;
%
%   JACOBIAN EVALUATION
%
     if nargin==7
        jac=feval(Fjac,x);
     else
        jac=diffjac(x,F,fx,l,u);
     end
     grad_old=grad;
     grad= jac'*fx;
%
%   CALCULATION OF THE SCALING MATRICES D (D), D^1/2 (DSQR), AND INV(D^1/2) (DMSQR)
%
     if scal==1
       if itc==1
        lambda=max(1.d-2,norm(grad));
       else
        lambda=max(1.d-2,(p'*(grad-grad_old))/(p'*p));
       end
     end

     [d,dsqr,dmsqr,ierr]=dmatrici(x,grad,l,u,scal,lambda);
     if ierr==6
        break;
     end

%
%   TRUST-REGION TYPE ASSIGNMENT
%
     if tr==0
       G = ones(n,1);
     else
       G = dmsqr;
     end

     dsqrgrad=dsqr.*grad;
     dgrad=d.*grad;
     Gdgrad=G.*dgrad;
     jdgrad=jac*dgrad;
     Gmgrad=grad./G;

     ndsqrgrad=norm(dsqrgrad);
```

```
      ndgrad=norm(dgrad);
      nGdgrad=norm(Gdgrad);
      njdgrad=norm(jdgrad);
      nGmgrad=norm(Gmgrad);

      if ndgrad<epsilon
         ierr=5;
         break;
      end

%
%    COMPUTATION OF THE MINIMIZER (PC) OF THE QUADRATIC MODEL ALONG D*GRAD
%
      vert=(ndsqrgrad/njdgrad)^2;
      pc=-vert*dgrad;
      pcv=pc;
      npc=norm(pc);
      pcc=pc;

%
%    INITIAL TRUST REGION RADIUS
%
      if itc==1
         if Delta==-1
            Delta=1;
         else
            if Delta==-2
               Delta=nGmgrad;
            end
         end
      end
%
%    NEWTON STEP
%
       if ~issparse(jac) | ~isnew
              [L,U,P] = lu(jac);
       else
              [L,U,P,Q] = lu(jac);
       end

       jacsingular = 0;
       if min(abs(diag(U)))== 0
          jacsingular = 1;
       end

       if ~jacsingular
          if ~issparse(jac) | ~isnew
              sn=-L\(P*fx);
              sn=U\sn;
          else
              sn=-L\(P*fx);
              sn=U\sn;
              sn=Q*sn;
          end

%
%    PROJECTED NEWTON STEP
%
          if (x+sn)>l & (x+sn)<u
             indpr=0;
          else
             indpr=1;
             sn=max(x+sn,l)-x;
             sn=min(x+sn,u)-x;
             sn=max(sigma,1-norm(sn))*sn;
          end
       end
```

```
%
%    TRUST-REGION STRATEGY
%
      rhof=0; nridu=-1;
      while (rhof<t & Delta>Deltamin)
            nridu=nridu+1;
%
%    COMPUTATION OF THE CAUCHY POINT
%
            if vert*nGdgrad>Delta    %norm(dmsqr.*pcc)>Delta
               pcv=-Delta*dgrad/nGdgrad;
            end
%
%    COMPUTATION OF THE TRUNCATED CAUCHY POINT
%
            pciv=pcv;
            npcv=norm(pcv);
            for i=1:n
                if pcv(i)~=0
                   alp(i)=max((l(i)-x(i))/pcv(i),(u(i)-x(i))/pcv(i));
                else
                   alp(i)=Inf;
                end
            end
            alpha1=min(alp);
            if alpha1<=1
               pciv=max(thetal,1-npcv)*alpha1*pcv;
            end
%
%    PATH COMPUTATION
%
            if ~jacsingular
%
%    THE JACOBIAN IS NONSINGULAR
%
               aa=fx+jac*pciv;
               seg=sn-pciv;
               bb=jac*seg;
               if norm(bb)~=0
                  gammamin=-aa'*bb/norm(bb)^2;
                  a=norm(G.*seg)^2;
                  b=(G.*pciv)'*(G.*seg);
                  c=norm(G.*pciv)^2-Delta^2;
                  l1=(-b+sign(-b)*sqrt(b^2-a*c))/a;
                  l2=c/(l1*a);
                  if gammamin>0
                     gammaend=max(l1,l2);
                     gamma=min(gammamin,gammaend);
                     if gamma >1
                        for i=1:n
                            if seg(i)~=0
                               alp(i)=max((l(i)-(x(i)+pciv(i)))/seg(i),...
                                          (u(i)-(x(i)+pciv(i)))/seg(i));
                            else
                               alp(i)=Inf;
                            end
                        end
                        alpha1=min(alp);
                        gamma=min(thetal*alpha1,gamma);
                     end
                  else
                     gammaend=min(l1,l2);
                     gamma=max(gammamin,gammaend);
                     if gamma<0
                        for i=1:n
                            if seg(i)~=0
                               alp(i)=max((l(i)-(x(i)+pciv(i)))/(-seg(i)),...
```

```
                                        (u(i)-(x(i)+pciv(i)))/(-seg(i)));
                            else
                               alp(i)=Inf;
                            end
                        end
                        alpha1=min(alp);
                        gamma=max(-theta1*alpha1,gamma);
                    end
                end
             else
                gamma=0;
             end
             p=(1-gamma)*pciv+gamma*sn;
          else
%    THE JACOBIAN IS SINGULAR
             p=pciv;
             gamma=0;
          end

%
%    ACCURACY REQUIREMENTS
%

          xpp=x+p;
          fxpp=feval(F,xpp);
          nvf=nvf+1;
          fnrmxpp=norm(fxpp);
          rhof=(fnrm-fnrmxpp)/(fnrm-norm(fx+jac*p));
          Deltas=Delta;
          Delta=min(delta1*Delta,0.5*norm(G.*p));
     end

     if (Delta<=Deltamin & rhof<t)
        nridu=nridu+1;
        ierr=3;
        break
     end
     Delta=Deltas;
%
%    UPDATING OF THE ITERATE
%
     x=xpp;
     del=100*eps;
     inddel=0;
     for i=1:n
         if x(i)<l(i)+del
            x(i)=l(i)+del;
            inddel=1;
         end
         if x(i)>u(i)-del
            x(i)=u(i)-del;
            inddel=1;
         end
     end

     if inddel==0
        fx=fxpp;
     else
        fx=feval(F,x);
        nvf=nvf+1;
     end

     fnrm=fnrmxpp;
     fnrm2=fnrm^2;
     rat=fnrm/fnrm0;
     nridut=nridut+nridu;
```

```
%
%   STORING AND PRINTING THE ITERATION'S SUMMARY
%
     if nargout>=4
        history(itc+1,:)=[itc, fnrm, nridu, gamma, rat];
     end
     if outflag>0
        disp(sprintf('   %3d    %10.5e   %3d     %12.5e   %10.5e   ',itc,fnrm,nridu,full(gamma),rat))
     end
     if (abs(fnrm-fnrm0)<=epsilon*fnrm & fnrm>stoptol)
        ierr=4;
        break
        return
     end
%
%   UPDATING OF THE TRUST-REGION SIZE
%
     if rhof>w
        Delta=max(Delta,delta2*norm(G.*p));
     end
end
%
%   FINAL OUTPUT
%
sol=x;
if nargin==7
   jac=feval(Fjac,x);
else
   jac=diffjac(x,F,fx,l,u);
end
grad_old=grad;
grad=jac'*fx;
if scal==1
   if itc==1
      lambda=max(1.d-2,norm(grad));
   else
      lambda=max(1.d-2,(p'*(grad-grad_old))/(p'*p));
   end
end
[d,dsqr,dmsqr,ind]=dmatrici(x,grad,l,u,scal,lambda);
ndgrad=norm(d.*grad);
output=[itc,nvf,fnrm,ndgrad,nridut];

if nargout==6
   if ~issparse(jac)
      diagnostic(1)=rank(jac);
      diagnostic(2:n+1)=svd(jac);
   else
      diagnostic=sprintf('You are working with sparse matrices, \n no diagnostic provided within this
   end
end

if (ierr==0 & fnrm>stoptol)
   if itc==maxit
      ierr=1;
   else
      ierr=2;
   end
end
if ierr>=1
   disp(sprintf('%s %1.0f','FAILURE, ierr= ',ierr))
   return
end

disp(sprintf('\n'))
disp(sprintf('FINAL OUTPUT:'))
disp(sprintf('                 iterations = %d',itc))
```

```
disp(sprintf('F-evaluations (no Jacobian) = %d',nvf))
disp(sprintf('     final value of ||F||_2 = %10.5e',fnrm))
disp(sprintf('\n'))

%   end of AS_ID function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [jac]=diffjac(x,F,F0,l,u);
%
%   Compute an approximation of the Jacobian matrix of F at x by forward finite differences
%
%   [jac] = diffjac(x,F,F0,l,u);
%
%   Input:
%      x,F = point and function
%       F0 = F(x), preevaluated
%      l,u = constraints
%
%   Output:
%      jac = approximated Jacobian matrix
%

n=length(x);
epsnew=sqrt(eps);
for j=1:n
%
%   CHOICE OF THE STEPLENGHT FOR THE FORWARD DIFFERENCES
%
    if x(j)==0
       h=epsnew;
    else
       h=epsnew*sign(x(j))*max(abs(x(j)),norm(x,1)/n);
    end
    xhj=x(j)+h;
%
%
    if (xhj<l(j) |xhj >u(j))
%
%   THE NEW POINT XHJ IS NOT FEASIBLE. IN THIS CASE THE BACKWARD
%   DIFFERENCE IS USED
%
       h=-h;
       xhj=x(j)+h;
       if (xhj<l(j) |xhj >u(j))
            disp('Function diffjac: loosing of feasibility using both backward and forward differences
            stop
       end
    end
    xh=x;  xh(j)=xhj;
    F1=feval(F,xh);
    jac(:,j)=(F1-F0)/h;
end

%   end of DIFFJAC function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [d,dsqr,dmsqr,ind]=dmatrici(x,grad,l,u,scal,lambda)

%
%
%   Compute the scaling matrix D and the matrices inv(D), (D)^(-2)
%
%   [d,dsqr,dmsqr,ind]=dmatrici(x,grad,l,u,scal,lambda);
%
%   Input:
```

```
%    x      = point
%    grad   = gradient of f, preevaluated
%    l,u    = constraints
%    scal   = parameter for choosing the scaling matrix
%    lambda = parameter needed for the choice scal=1;
%

%   Output:
%    d     =  array such that d=diag(D(x)).
%    dsqr  =  array such that dsqr=diag((D(x))^1/2).
%    dmsqr =  array such that dmsqr=diag((D(x))^-1/2).
%    ind   =  0 upon successful termination.
%          =  6 an overflow would be generated when computing
%             the scaling matrix D.
%
ind=0;
n=length(x);
d=ones(n,1); dsqr=ones(n,1);  dmsqr=ones(n,1);
if scal==1
%
%   HAGER ET AL. SCALING MATRIX
%
   d=(1/lambda)*ones(n,1);
   for i=1:n
       if grad(i)<0
          if u(i)~=Inf
             diff=u(i)-x(i);
             d(i)=1/(lambda-grad(i)/diff);
             if d(i)<=1/realmax
                ind=6;
                return
             end
          end
       else
          if l(i)~=-Inf
             diff=x(i)-l(i);
             d(i)=1/(lambda+grad(i)/diff);
             if d(i)<=1/realmax
                ind=6;
                return
             end
          end
       end
   end
   dsqr=sqrt(d);
   dmsqr=1./dsqr;
elseif scal==0
%
%   COLEMAN-LI SCALING MATRIX
%
       for i=1:n
           if grad(i)<0
              if u(i)~=Inf
                 diff=u(i)-x(i);
                 sqdiff=sqrt(diff);
                 if diff>=(1/realmax)
                    dmsqr(i)=1/sqdiff;
                    dsqr(i)=sqdiff;
                    d(i)=diff;
                 else
                    ind=6;
                    return
                 end
              end
           else
              if l(i)~=-Inf
                 diff=x(i)-l(i);
```

```
                 sqdiff=sqrt(diff);
                 if diff>=(1/realmax)
                    dmsqr(i)=1/sqdiff;
                    dsqr(i)=sqdiff;
                    d(i)=diff;
                 else
                    ind=6;
                    return
                 end
              end
           end
       end
elseif scal<0
%
%   KANZOW-KLUG SCALING MATRIX
%
         gamma = -scal;
         chk = (l==-Inf)&(u==Inf);
         a = x-l+gamma*max(0,-grad);
         b = u-x+gamma*max(0,grad);
         c = min(a,b);
         for i=1:n
             if chk(i)==1
                dmsqr(i) = 1;
             else
                if c(i)>1/realmax
                   dmsqr(i) =1/sqrt(c(i));
                else
                   ind=6;
                   return
                end
             end
         end
         dsqr = 1./dmsqr;
         d = dsqr.^2;
elseif scal==2
%
%   USER'S OWN SCALING MATRIX
%
         d = usersd(x,grad,l,u);
         dsqr = sqrt(d);
         dmsqr = 1./dsqr;
end

%   end of DMATRICI function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [a] = isnew
v=version;
cp=0; i1=0; i2=0;
for i=1:length(v)
   if v(i)=='.'
      cp=cp+1;
      if cp==2
         break
      end
   end
   if cp<1
      i1 = i1+1;
      x1(i1)=v(i);
   else
      if v(i)~='.'
         i2 = i2+1;
         x2(i2)=v(i);
      end
   end
```

```
end
c1=str2num(x1);
c2=str2num(x2);
a = (c1>=7|(c1==6 & c2>=5));
```

Annexe B

Exemples de scaling matrices

Dans cette annexe, on définit deux matrices de scaling dont le calcul est implémenté dans le code C++. Leurs définitions sont fournies ci-après.

B.1 La matrice de scaling de Coleman et Li

La matrice est diagonale et ses éléments diagonaux sont donnés par

$$d_i^{CL}(x) = \begin{cases} u_i - x_i & \text{si } (\nabla f(x))_i < 0 \text{ et } u_i < +\infty, \\ x_i - l_i & \text{si } (\nabla f(x))_i > 0 \text{ et } l_i > -\infty, \\ \min(x_i - l_i, u_i - x_i) & \text{si } (\nabla f(x))_i = 0 \text{ et } l_i > -\infty \text{ ou } u_i < +\infty, \\ 1 & \text{sinon.} \end{cases}$$

B.2 La matrice de scaling HMZ

C'est aussi une matrice diagonale dont les éléments sont donnés par

$$d_i^{HMZ}(x) = \frac{X_i(x)}{\alpha(x) X_i(x) + |\nabla f(x)_i|}$$

avec

$$X_i(x) = \begin{cases} u_i - x_i & \text{si } (\nabla f(x))_i < 0 \text{ et } u_i < +\infty, \\ x_i - l_i & \text{si } (\nabla f(x))_i > 0 \text{ et } l_i > -\infty, \\ 1 & \text{si } (\nabla f(x))_i = 0, \end{cases}$$

et $\alpha(x)$ est une fonction continue strictement positive pour tout x et uniformément bornée.
Pour nos applications, on a imposé

$$\alpha(x) = 1 + \frac{1}{1 + x^2}$$

Bibliographie

[1] Macconi M. Pieraccini S. Bellavia S. Constrained dogleg methods for nonlinear systems with simple bounds. *Computational Optimization and Applications*, 2012.

[2] Pieraccini S. Bellavia S., Macconi M. On affine-scaling inexact dogleg methods for bound-contrained nonlinear systems. *Computational Optimization and Applications Tech. Rep. 5/2009*, 2009.

[3] Pawlowski R. P. et al. Inexact newton dogleg methods. *SIAM J. Numer. Anal.,11 (2010), pp. 2465–2471*, 2012.

[4] M. Ulbrich M. Heinkenschloss and S. Ulbrich. Superlinear and quadratic convergence of affine-scaling interior-point newton methods for problems with simple bounds without strict complementarity assumptions. *Math. Program., 86, pp. 615–635*, 1999.

[5] Montarnal P. Métivier L. Strategies for solving index one dae with non-negative constraints : Application to liquid-liquid extraction. *Journal of Computational Physics*, 2012.