Fiabilisation d'un outil de prévision de la demande en électricité au Québec utilisé par Hydro-Québec

Documentation de l'implémentation d'ENLSIP en Julia

20 mars 2022

Pierre Borie

Supervisé par

Fabian Bastin et Stéphane Dellacherie

Résumé

Afin de prévoir efficacement la demande en électricité à court terme, Hydro-Québec, principal producteur d'électricité du Québec, calibre ses modèles de prévision grâce à l'utilisation d'ENLSIP, algorithme d'optimisation écrit en Fortran77. Néanmoins, sa maintenance est désormais complexe à assurer. Afin d'améliorer sa fiabilité, nous en proposons une nouvelle implémentation réalisée dans le langage Julia, langage informatique moderne conçu pour le calcul scientifique haute performance et les sciences des données. Cette dernière s'accompagne d'une modélisation de la méthode d'optimisation utilisée ainsi que de tests effectués sur des problèmes de moindres carrés pour en évaluer les performances. Sont ensuite explorées certaines pistes d'amélioration.

Mots-clés : Optimisation non linéaire sous contraintes, moindres carrés, calibration de modèle, demande en énergie, langage de programmation Julia.

Abstract

In order to predict efficiently short term demand, the main electricity producer in Quebec, Hydro-Quebec, calibrates its own previsions models by using ENLSIP, an algorithm written in Fortran77 harder to maintain correctly nowadays. This report exposes a new implementation made in the Julia programming language. A modelisation of the optimisation method used is detailled. The performances are also evaluated by running the algorithm on least squares test problems. Finally, some ideas of improvement are discussed.

Keywords : Constrained nonlinear programming, least squares, model calibrating, energy demand, Julia programming language.

Table des matières

Table des figures

Introduction

Cette partie a pour but d'introduire le contexte dans lequel mon stage s'est déroulé et présente la problématique sur laquelle j'ai travaillé.

Le chapitre 1 traite de l'ensemble des besoins et livrables attendus du projet puis de la méthode de travail envisagée. Les chapitres 2, 3 et 4 portent sur le travail effectué, à savoir l'étude bibliographique, la finalité de mon travail de recherche et les résultats obtenus. Enfin, une conclusion dresse un bilan du travail réalisé et explore les différentes perspectives d'avenir du projet.

Contexte du stage

Mon stage de projet de fin d'études (PFE) est l'occasion d'une étroite collaboration entre l'Université de Montréal (UdeM) et l'unité qui s'occupe de la prévision de la demande au sein de la division TransÉnergie d'Hydro-Québec, société d'État responsable de la production, du transport et de la distribution de l'électricité dans la province de Québec au Canada. La division TransÉnergie est plus spécifiquement responsable du réseau de transport d'électricité.

À cause de la pandémie due au Covid-19, je n'ai pas pu me rendre sur place. Mon stage s'est donc déroulé entièrement en télé-travail depuis mon domicile à Paris.

Comme dit plus haut, Hydro-Québec est le principal fournisseur d'électricité au Québec. Cette électricité est produite à partir de barrages hydroélectriques dont la bonne gestion des flux d'électricité produits est primordiale pour Hydro-Québec. En effet, l'électricité doit être produite en quantité suffisante pour satisfaire la demande du Québec, sans pour autant en produire plus que nécessaire afin de ne pas générer de pertes. C'est pourquoi Hydro-Québec dispose de plusieurs outils de prévision de la demande en électricité au Québec à plus ou moins long terme.

Mon sujet porte plus particulièrement sur la prévision de la demande sur un horizon de 24 à 48 heures, soit à court terme. Cette prévision est réalisée à l'aide d'un modèle mathématique à plusieurs milliers de paramètres. La quantité d'électricité consommée par la population pouvant fortement varier en fonction des saisons, des jours de la semaine voire de l'heure de la journée, ces paramètres doivent être ajustés avant la réalisation de chaque nouvelle prévision. Grâce à un ensemble de capteurs de puissance répartis le long des lignes de transport haute tension, Hydro-Québec possède une quantité très importante de données de consommation électrique au jour le jour qui sont utilisées pour calibrer au mieux les paramètres de ce modèle. Ce procédé de calibration est réalisé par un algorithme d'optimisation nommé ENLSIP dont la méthode et l'implémentation en Fortran77 ont été réalisées par Lindström et Wedin [12]. Cet algorithme a pour but de résoudre un problème d'optimisation de moindres carrés non linéaires sous contraintes non linéaires. Nous verrons dans la section 2.1 la modélisation de ce type de problèmes et en quoi ce dernier est bien adapté aux besoins d'Hydro-Québec.

Problématique de stage

Différentes complications se posent aujourd'hui avec l'utilisation d'ENLSIP. Tout d'abord, étant codé en Fortran77 et n'ayant pas été mis à jour depuis plus de 30 ans, la maintenance de cet algorithme s'avère de plus en plus difficile à effectuer. De plus, les besoins d'Hydro-Québec en termes de performance de leurs outils de prévision et de taille des problèmes à résoudre ont évolué avec le temps et ne sont plus les mêmes que dans les années 80, le nombre de données dont ils disposent ayant considérablement augmenté et étant amené à continuer à croître exponentiellement dans le futur avec les compteurs intelligents. Cet aspect est accentué par le fait qu'ENLSIP a été conçu à une époque où il y avait de fortes contraintes techniques sur le matériel informatique et la taille des problèmes que l'on pouvait résoudre en un temps raisonnable (soit en quelques minutes). Ensuite, la méthode de résolution implémentée dans ENLSIP ne bénéficie pas des progrès réalisés en programmation non linéaire, qui sont pourtant conséquents depuis les années 80. Hydro-Québec souhaite donc moderniser leurs outils de prévision de la demande par la modification et l'amélioration d'ENLSIP et le passage au langage Julia [2]. Il s'agit en effet d'un langage dédié au calcul scientifique haute performance et les sciences des données, dont la première version a été publiée en 2009 et qui est en plein essor. Ce nouveau langage a la particularité de rendre compatible la simplicité d'un langage de haut niveau tel que Python et la performance d'un langage compilé tel que Fortran.

C'est dans ce cadre que s'inscrit mon sujet de PFE. Ce projet a également fait l'objet d'un financement de la part de Mathematics of Information Technology and Complex Systems (MITACS), organisme national finançant des programmes de recherche et de formation dans des domaines liés à l'innovation industrielle et sociale au Canada.

Quant à mon travail, ce dernier s'articule autour de deux problématiques principales :

- Implémenter en Julia la méthode ENLSIP de résolution d'un problème de moindres carrés sous contraintes développée dans afin de fiabiliser et favoriser la maintenance de cet algorithme.
- Améliorer l'outil de prévision de la demande via l'implémentation, toujours en Julia, d'une méthode de résolution mieux adaptée aux nouveaux besoins d'Hydro-Québec et basée sur des aspects théoriques plus modernes.

Chapitre 1

Besoins et organisation du projet

1.1 Besoins

Avant d'expliciter le travail effectué lors de ce projet, il convient d'aborder ce qui est attendu de la part de l'UdeM et d'Hydro-Québec.

Livrables et travail à effectuer

Dans un premier temps, certains livrables ont été définis avec mes collaborateurs, dont mon maître de stage. Les principales attentes concernent l'implémentation de l'algorithme ENLSIP en Julia. Le code source de ce dernier doit en effet être fourni à travers différents fichiers Julia (format .jl) nécessaires pour l'exécution de l'algorithme. Ce travail ayant vocation à être disponible en open source sous la forme d'une librairie Julia, une documentation en anglais indiquant comment utiliser le solveur et consultable via un fichier HTML est également attendue.

Afin de s'assurer de l'efficacité de l'implémentation réalisée, différents tests sur des problèmes de moindres carrés seront réalisés tout au long de son développement. Ces derniers sont effectués et consultables par des fichiers notebooks Jupyter (format .ipynb), type de fichier compatible avec l'exécution du code Julia. La modélisation des problèmes expérimentés ainsi que les résultats attendus doivent figurer sur ces fichiers de tests.

Les problèmes testés seront de deux catégories :

- problèmes de programmation non linéaire documentés s'inscrivant dans un cadre purement mathématique ;
- problèmes formés à partir des modèles et données réelles utilisés par Hydro-Québec en contexte opérationnel.

Le premier type de problèmes a pour vocation de vérifier le bon déroulement de l'algorithme et sa concordance avec la version codée en Fortran77.

Le dessein derrière le second type de problèmes est de tester l'algorithme dans des cas d'utilisation correspondant au contexte industriel auquel l'unité de prévision de la demande d'Hydro-Québec fait face quotidiennement. Ces problèmes-ci seront d'avantage mis à contribution lorsque l'implémentation d'ENLSIP en Julia sera terminée, afin d'amorcer le remplacement de l'algorithme en Fortran77 par la nouvelle version en Julia. Les différents résultats obtenus à travers des tests effectués sont décrits dans la section 4.5.

Le dernier livrable attendu, requis par Fabian Bastin mon maître de stage, concerne plus parti-

culièrement la méthode d'optimisation employée dans ENLSIP. En effet, la description mathématique de celle-ci n'est pas intégralement documentée par ses auteurs Lindström et Wedin [12]. Or, en vue de moderniser la méthode implémentée dans cet algorithme, il est important d'en comprendre tous les aspects théoriques afin de mieux cibler les points qui peuvent être améliorés. Une documentation avec la formulation mathématique des différents éléments de la méthode d'optimisation d'ENLSIP est donc attendue. Le fruit de ce travail est présentée au chapitre 4.

Résultats et performances attendus

Ensuite, la finalité de ce projet étant de produire un algorithme devant être utilisé en contexte industriel, des résultats et performances sont attendus de la part d'Hydro-Québec.

L'objectif premier derrière le passage à Julia est, pour Hydro-Québec, de fiabiliser plusieurs de ses outils de prévision de la demande en électricité. Cela passe par la retranscription exacte de la méthode usitée dans ENLSIP codée originellement Fortran77 tout en tirant parti des avantages de programmation conférés par le nouveau langage Julia, plus moderne et moins complexe à maintenir que le Fortran77. L'amélioration de l'algorithme et de la méthode utilisée à proprement parler ne vient que dans un second temps. Il est donc primordial de d'abord produire un outil renvoyant des résultats identiques, à la précision machine près, à ceux obtenus avec la version Fortran77.

Enfin, les résultats en termes de temps de calcul entre les deux versions doivent être similaires ou réduits afin que la transition au Julia ne se fasse pas au prix d'une dégradation des performances. Puisqu'il s'agit d'un optimiseur en appui à des outils opérationnels de prévision de la demande court terme, les temps d'exécution sont soumis à des contraintes de temps de calcul, en particulier pour les cas rencontrés en contexte opérationnel. Celles-ci sont de cet ordre :

- de 10 à 30 secondes sur des problèmes de petite taille (i.e. jusqu'à cinq paramètres) ;
- de 10 à 15 minutes pour des problèmes de grande taille, soit issus du contexte opérationnel (i.e. de l'ordre de plusieurs centaines de paramètres).

1.2 Méthode de travail

Ce projet ayant été réalisé en télé-travail permanent depuis mon domicile à Paris, l'éloignement géographique avec mes collaborateurs, tous résidents au Québec, a donc dû être pris en compte dans son organisation. Étant donné que j'étais seul à travailler sur la retranscription en Julia de l'algorithme en Fortran77, cela n'a pas été un réel obstacle pour avancer dans mes recherches.

Développement itératif

Vu la complexité la complexité que présente l'algorithme ENLSIP, il a fallu instaurer une stratégie de développement avec mon maître de stage.

L'idée a été d'ajouter chacune des fonctionnalités de la méthode, décrites au chapitre 4, de façon itérative, ceci afin de complexifier au fur et à mesure l'implémentation en Julia. Des entrevues avec mon maître de stage se sont tenues de façon hebdomadaire. Ces dernières étaient dédiées d'abord à la présentation de mon travail de la semaine passée, puis aux points sur lesquels travailler pour la semaine à venir. Cela a permis de s'adapter aux difficultés rencontrées au cours du développement de l'algorithme. Afin de garder une traçabilité de toutes les versions intermédiaires de mon travail, un répertoire github a également été mis en place.

À chaque nouvel ajout d'une fonctionnalité, mon implémentation a été testée sur les problèmes de tests exposés à la section 4.5. Cela a régulièrement permis de repérer quels éléments devaient être améliorés en priorité pour la prochaine version.

Collaboration avec Hydro-Québec

Ensuite, comme mon stage s'inscrit dans un projet plus global de modernisation des outils de prévision de la demande d'Hydro-Québec, certains membres de l'unité Prévisions de la demande ont également été des parties prenantes de mon PFE. Ces dernières m'ont aidé à accéder à des environnements de travail me permettant de tester mon implémentation d'ENLSIP sur des machines virtuelles et infrastructures proches de celles d'Hydro-Québec en contexte opérationnel. Une présentation de mon travail a eu lieu de façon régulière avec les membres d'Hydro-Québecimpliqués dans le projet également sur une base hebdomadaire.

Cette collaboration avait également pour objectif de mettre en place différents tests de mon algorithme sur des jeux de données réelles et avec des modèles de prévision utilisés par Hydro-Québec. Néanmoins, cela s'est jusqu'à présent avéré difficile à mettre en place. En effet, les modèles étant relativement complexes et codés en Fortran, une passerelle entre les langages Fortran et Julia a dû être mise en place afin de pouvoir appeler ces fonctions en Fortran depuis un script en Julia. Ce travail est néanmoins fortement avancé et sera mis à contribution lors de travaux futurs sur le projet.

Chapitre 2

État de l'art

Les problèmes de moindres carrés occupent une place importante en programmation linéaire et non linéaire [4, 7, 14, 16] et plusieurs méthodes de résolution telles que des méthodes de type Newton, Gauss–Newton [11], Levenberg–Marquardt [13] ou encore de régions de confiance [3] ont déjà beaucoup été étudiées. On trouve de nombreuses applications de ces méthodes, ou de variations autour de ces dernières dans les problèmes de calibration de modèles [10] ou bien dans la régression linéaire en apprentissage statistique [1, 8].

Ce chapitre propose d'abord une modélisation générale d'un problème de moindres carrés sous contraintes puis s'ouvre sur une revue de la littérature de différentes méthodes de résolution utilisées face à ce type de problèmes.

2.1 Présentation du problème de moindres carrés

Comme dit en introduction, l'algorithme utilisé par Hydro-Québec résout un problème de moindres carrés, dont une modélisation dans le cas général est donnée ci–après. Cette catégorie de problèmes d'optimisation occupe une place importante en programmation linéaire et non linéaire.

Les notations utilisées dans cette section s'appuient sur celles utilisées par Lindström et Wedin [12], cet article expliquant la méthode implémentée dans ENLSIP.

On note q, l, m et n des entiers naturels tels que $q \leq l \leq n \leq m$.

On considère m observations réelles, pouvant s'apparenter à des observations temporelles, (t_i, y_i), les t_i représentant les données d'entrée et les y_i celles de sortie. On souhaite ajuster un modèle h de paramètre $x \in \mathbb{R}^n$ qui est censé approcher au mieux nos observations, tout en satisfaisant l contraintes (potentiellement non linéaires) dont q sont des contraintes d'égalité, les $l-q$ restantes étant des contraintes d'inégalité.

Ces l contraintes sont modélisées par la multi-fonction $c = (c_1, \ldots, c_l)^T : \mathbb{R}^n \to \mathbb{R}^l$ dont les q premières composantes correspondent aux contraintes d'égalité et les $l-q$ suivantes correspondent aux contraintes d'inégalité.

Ensuite, pour $i = 1, \ldots, m$, on modélise l'écart entre la i-ème observation et la prédiction associée par la fonction $r_i : x \mapsto y_i - h(t_i, x)$.

La multi-fonction vectorielle $r = (r_1, \ldots, r_m)^T : \mathbb{R}^n \to \mathbb{R}^m$, aussi appelée fonction des résidus, représente, à travers ses composantes, l'écart entre les observations et les prédictions réalisées

par le modèle h. La somme des écarts au carré entre prédictions et observations modélise alors l'erreur générée par le modèle h pour un certain jeu de paramètres x. On cherche donc à trouver le paramètre x^* qui minimise cette somme, soit :

$$x^* = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m} \left(h(t_i, x) - y_i)\right)^2 .$$

On souhaite trouver les paramètres minimisant la norme euclidienne des résidus, soit la fonction $f : x \mapsto \frac{1}{2}\|r(x)\|^2$.

Le problème de moindres carrés non linéaires sous contraintes se modélise alors de la façon suivante :

$$\begin{cases} \min\limits_{x \in \mathbb{R}^n} \frac{1}{2}\|r(x)\|^2, \\ \text{s.c.} \\ c_i(x) = 0, \quad \text{pour } i = 1, \ldots, q, \\ c_j(x) \geq 0, \quad \text{pour } j = q+1, \ldots, l. \end{cases} \tag{2.1}$$

On se place dans le cas où la fonction r est continûment différentiable deux fois afin que le gradient et la matrice hessienne de la fonction objectif f soient bien définis.

La matrice jacobienne de la fonction r, notée J et de taille $(m \times n)$, s'exprime par :

$$J(x) = \begin{bmatrix} \nabla r_1(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix}. \tag{2.2}$$

On peut alors calculer directement les expressions du gradient et de la matrice hessienne de f :

$$\nabla f(x) = \sum_{k=1}^{m} r_k(x) \nabla r_k(x) = J^T(x) r(x), \tag{2.3}$$

$$\nabla^2 f(x) = \sum_{k=1}^{m} \nabla r_k(x) \nabla r_k(x)^T + \sum_{k=1}^{m} r_k(x) \nabla^2 r_k(x) \tag{2.4}$$

$$= J^T(x) J(x) + \sum_{i=1}^{m} r_i(x) \nabla^2 r_i(x), \tag{2.5}$$

où les $\nabla^2 r_k = \left[\dfrac{\partial^2 r_k}{\partial x_i x_j}\right]_{(i,j)}$, $k = 1, \ldots, m$, sont les hessiennes des m composantes des résidus.

Le problème considéré comprenant des contraintes, on rappelle que si x^*, un minimum local de (2.1), satisfait les conditions KKT, alors il existe $\lambda^* \in \mathbb{R}^l$ tel que :

$$\begin{aligned} \nabla f(x^*) - \sum_{i=1}^{l} \lambda_i^* \nabla c_i(x^*) = 0, & \\ c_i(x^*) = 0, \text{ pour } i = 1, \ldots, q, & \\ c_j(x^*) \geq 0, \text{ pour } j = q+1, \ldots, l, & \\ \lambda_j^* \geq 0, \text{ pour } j = q+1, \ldots, l, & \\ \lambda_j^* c_j(x^*) = 0, \text{ pour } j = q+1, \ldots, l. & \end{aligned} \tag{2.6}$$

Le lagrangien du problème (2.1) s'écrit comme :

$$\mathcal{L} : (x, \lambda) \longmapsto f(x) - \sum_{i=1}^{l} \lambda_i c_i(x), \tag{2.7}$$

où λ désigne le vecteur des multiplicateurs de Lagrange.

2.2 Le cas des moindres carrés linéaires

Une première méthode part du cas où le modèle h tel que présenté en (2.1) est une fonction linéaire de x, i.e. $h(x) = Ax$, où A est une matrice $(m \times n)$, et où il n'y a pas de contraintes.

Le problème (2.1) se réécrit simplement :

$$\min f(x) = \frac{1}{2}\|Ax - y\|^2. \tag{2.8}$$

On a également :

$$\nabla f(x) = A^T(Ax - y), \ \nabla^2 f(x) = A^T A.$$

Le cas linéaire assure trivialement la convexité de la fonction objectif et donc que tout point x^* vérifiant $\nabla f(x^*) = 0$ est un minimiseur global de f, ce qui impose la satisfaction des équations linéaires suivantes, aussi appelées équations normales :

$$A^T A x^* = A^T y. \tag{2.9}$$

Nocedal et Wright [14] présentent trois approches de calcul matriciel pour résoudre le système (2.9) : une s'appuyant sur la factorisation de Cholesky de A, une autre utilisant une factorisation QR de A et enfin une faisant intervenir la décomposition en valeurs singulières de A. Nous allons décrire la deuxième, soit celle avec la factorisation QR, car elle occupe une part importante de l'algorithme ENLSIP sur lequel j'ai travaillé [12].

On rappelle que la factorisation QR de A consiste à définir les matrices :

- Q une matrice orthogonale $(m \times m)$;
- R matrice triangulaire supérieure $(n \times n)$;
- P une matrice de permutation $(n \times n)$,

telles que

$$AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R, \tag{2.10}$$

avec $|r_{11}| \geq |r_{22}| \geq \ldots \geq |r_{nn}|$ où les r_{ii} désignent les éléments diagonaux de R et Q_1 les n premières colonnes de Q. On montre que cette factorisation QR existe pour toute matrice A $(m \times n)$.

En permutant ainsi les colonnes de la matrice A, on peut déterminer son rang en calculant celui de R. En effet, étant triangulaire supérieure et à éléments diagonaux décroissants en valeur absolue, on a :

$$\text{rang}(A) = \max_{1 \leq i \leq n} \{i \mid |r_{ii}| > 0\}. \tag{2.11}$$

Cela permet notamment de détecter efficacement si le système linéaire induit par le problème (2.8) est de rang déficient ou non. On suppose néanmoins que la matrice A est de rang plein pour la suite.

En remarquant que $\|Ax - y\| = \|Q^T(Ax - y)\|$ et en injectant la factorisation (2.10), la solution x^* s'obtient directement par :

$$x^* = PR^{-1}Q_1^T y.$$

D'un point de vue numérique, l'autre avantage de cette approche de résolution est qu'elle ne dégrade pas forcément le conditionnement du problème, ce dernier étant fortement déterminé par celui de la matrice A.

La résolution des équations normales est un enjeu majeur des problèmes de régression linéaire tels que ceux utilisés en apprentissage statistique chez Hastie *et al.* [8] et Johnson [10], pour lesquels les modèles de prédiction sont très souvent linéaires.

2.3 Méthode de Gauss–Newton

On s'intéresse désormais au cas où la fonction objectif est non linéaire. Les approches de type Gauss-Newton sont très utilisées pour les méthodes itératives car sont bien adaptées à cette structure de problèmes. Comme présentées chez Dennis Jr et Schnabel [4] et Nocedal et Wright [14], elles s'appuient sur une linéarisation du premier ordre de la fonction des résidus autour d'un point x :

$$r(x + p) \approx J(x)p + r(x). \tag{2.12}$$

Dans le cadre d'un algorithme itératif, on se place à l'itération $k \in \mathbb{N}$ en un point x_k fixé et on cherche une direction de descente p_k permettant d'obtenir un nouvel itéré $x_{k+1} = x_k + p_k$ faisant diminuer la fonction objectif. Cela est répété jusqu'à la convergence vers un minimum.

Chercher cette direction de descente revient, x_k étant fixé, à résoudre :

$$\min_p f(x_k + p) = \min_p \frac{1}{2}\|r(x_k + p)\|^2.$$

En injectant la linéarisation (2.12), la direction de descente est choisie comme étant la solution du problème, cette fois-ci quadratique :

$$\min_p \frac{1}{2}\|J(x_k)p + r(x_k)\|^2. \tag{2.13}$$

Le sous-problème de l'itération k est un problème de moindres carrés linéaires comme celui en (2.8) auquel on peut par exemple appliquer la méthode décrite en section 2.2. Notons d'ailleurs que la solution de ce sous-problème, notée p_{GN}, vérifie les équations normales (2.9), soit :

$$J(x_k)^T J(x_k) p_{GN} = -J(x_k)^T r(x_k). \tag{2.14}$$

Partant d'une méthode de type Newton où l'on résout pour un vecteur p le système :

$$\nabla^2 f(x_k) p = -\nabla f(x_k), \tag{2.15}$$

on remarque qu'en injectant les expressions du gradient et de la hessienne de la fonction objectif f respectivement en (2.3) et (2.5) dans (2.15), en ignorant les termes associés aux dérivées d'ordre 2, on retrouve les équations normales (2.14)

La méthode de Gauss-Newton s'obtient alors en faisant l'approximation :

$$\nabla^2 f(x_k) \approx J(x_k)^T J(x_k)$$

.

L'avantage de cette dernière est que les hessiennes des résidus n'ont pas à être calculées explicitement. En effet, cela revient à supposer que les termes $r_i(x)\nabla^2 r_i(x)$ de l'expression (2.5) sont nuls, ce qui constitue un important gain de temps de calcul et de mémoire sur des problèmes de grande taille. De plus, il s'avère en pratique que c'est une assez bonne approximation pour les problèmes ayant de faibles valeurs de résidus, ce qui est d'autant plus vrai à la solution optimale, comme mentionné chez Nocedal et Wright [14].

Néanmoins, la méthode présentée n'est pas applicable en tant que tel au cas avec contraintes, qui est pourtant celui du problème (2.1). C'est d'ailleurs une des difficultés rencontrées lors de ma recherche d'ouvrages et travaux sur les moindres carrés ; en effet, la plupart des méthodes de résolution concernent plutôt le cas non contraint. Ce dernier s'applique dans la plupart des problèmes de calibration de modèles, comme en apprentissage automatique par exemple Audibert et Catoni [1].

Comme nous le verrons plus en détail au chapitre 3, la méthode itérative développée par Lindström et Wedin [12] se base sur Gauss-Newton mais a la particularité de justement prendre en compte des contraintes d'égalité et d'inégalité.

Notons simplement qu'à chaque itération, on procède d'abord à une prédiction des contraintes actives à la solution à l'aide d'une estimation des multiplicateurs de Lagrange afin de ne travailler qu'avec des contraintes d'égalité, comme dans une approche dite EQP (pour Equality Quadratic Programming) tel que décrit dans [14]. Les contraintes restantes et les résidus sont ensuite linéarisés et différents sous-problèmes sont résolus à l'aide de décompositions matricielles QR. Cela permet de calculer une direction de descente menant vers un point réalisable, i.e. satisfaisant toutes les contraintes, et faisant diminuer la fonction objectif. On choisit ensuite une longueur de pas le long de cette direction de descente par une recherche linéaire dont la méthode est renseignée par Lindström et Wedin [11].

2.4 Méthode de Levenberg–Marquardt

La méthode de Levenberg-Marquardt est une autre méthode itérative très utilisée pour la résolution de moindres carrés. Son principe général et les principaux aspects théoriques sont explicités par Moré [13]. Le point de départ de cette méthode est, comme en section 2.3, une linéarisation de la fonction des résidus amenant à devoir résoudre à chaque itération le sous problème (2.13) afin de calculer une direction de descente. L'ajout par rapport à la méthode de Gauss–Newton vue

en 2.3 est que l'on prend en compte le fait que la linéarisation n'est pas valable pour tout vecteur p dans la modélisation du sous problème à résoudre. Les directions de descente potentielles sont en effet restreintes à un certain ensemble E_k, d'où en (2.16) le sous problème :

$$\min_{p \in E_k} \frac{1}{2} \|J(x_k)p + r(x_k)\|^2, \tag{2.16}$$

avec $E_k = \{p \ , \ \|D_k p\| \leq \Delta_k\}$ et où :

- D_k est une matrice diagonale prenant en compte le potentiel mauvais conditionnement du problème, ses éléments dépendant des dérivées partielles des résidus ;
- Δ_k est un réel positif.

Or, Moré [13] montre que si p_{LM} est solution de (2.16), alors il existe $\mu_k > 0$ tel que :

$$\left[J(x_k)^T J(x_k) + \mu_k D_k^T D_k\right] p_{LM} = -J(x_k)^T r(x_k). \tag{2.17}$$

On retrouve une formulation similaire à celle de l'équation (2.14), à l'exception du terme $\mu_k D_k^T D_k$ faisant ici office de terme correctif et améliorant la convergence pour les problèmes moins bien conditionnés.

Une procédure type pour cette méthode est décrite par Moré [13] :

1. Pour Δ_k donné, trouver μ_k et p_k vérifiant (2.17).
2. Si $f(x_k + p_k) \leq f(x_k)$ alors $x_{k+1} = x_k + p_k$, sinon $x_{k+1} = x_k$.
3. Choix de Δ_{k+1} et D_{k+1}.

Les choix des paramètres ou des méthodes numériques de résolution peuvent évidemment varier d'une implémentation à l'autre.

Dans des approches plus récentes, comme celle présentée par Yuan [19], l'équation de la direction de descente est plutôt donnée en par :

$$\left[J(x_k)^T J(x_k) + \mu_k I\right] p_{LM} = -J(x_k)^T r(x_k), \tag{2.18}$$

où I est la matrice identité. Le paramètre μ_k est quant à lui mis à jour en fonction des performances des précédentes itérations, c'est-à-dire en fonction de $\|r(x_k)\|$ directement comme suit :

$$\mu_k = \|r(x_k)\|^\delta,$$

pour un certain $\delta \in [1, 2]$ défini au préalable. Sous certaines hypothèses, Yamashita et Fukushima [18] montrent que ceci permet d'obtenir une convergence quadratique vers la solution du problème.

2.5 Méthode de régions de confiance

Les régions de confiance [3] sont un autre exemple de méthode pouvant bien s'adapter à la structure des moindres carrés. On reste encore une fois dans le cas sans contraintes.

S'agissant là de méthodes itératives, l'idée est toujours de construire une suite d'itérés x_k convergeant vers la solution du problème. À chaque itération, partant d'un point x_k, on définit un modèle $m_k(x)$ qui approche la fonction objectif f dans un voisinage de x_k. Ce voisinage, noté $\mathcal{B}_k$, est aussi appelé région de confiance et est défini comme suit :

$$\mathcal{B}_k = \{x \in \mathbb{R}^n,\ \|x - x_k\|_k \leq \Delta_k\}\,, \tag{2.19}$$

où Δ_k est le rayon de la région de confiance et $\|\cdot\|_k$ est la norme choisie à l'itération k.

Une fois $\mathcal{B}_k$ défini, on cherche à calculer s_k tel que le point $x_k + s_k$ réduit au mieux la fonction objectif m_k. Si c'est bel et bien le cas, le rayon Δ_k est maintenu voire étendu, sinon on le réduit.

La figure 2.1 décrit les grandes étapes d'un exemple d'algorithme présenté par Conn *et al.* [3].

L'intérêt de l'utilisation des régions de confiance pour les moindres carrés réside dans le fait qu'au lieu de construire directement un modèle de la fonction objectif f, on se base sur un modèle m_k^r de la fonction des résidus à chaque itération k :

$$m_k^f(x) \underset{\text{def}}{=} \frac{1}{2}\|m_k^r(x)\|^2.$$

Conn *et al.* [3] montrent qu'avec un bon choix de modèle pour r, on peut alors obtenir un modèle du second ordre pour f et se ramener à des calculs de moindres carrés linéaires. Sous certaines hypothèses de différentiabilité sur la fonction r, on assure la convergence de l'algorithme présenté en figure 2.1.

Par exemple, Yuan [19] définit une méthode de régions de confiance où le sous-problème à chaque itération est un problème quadratique de la forme :

$$\begin{cases} \min\limits_p \|r(x_k) + J(x_k)p\|^2 + \dfrac{1}{2}p^T B_k p, \\ \text{s.c. } \|p\| \leq \Delta_k. \end{cases} \tag{2.22}$$

où B_k est une certaine matrice définie à chaque itération en fonction des données et paramètres du problème.

2.6 Bilan sur l'état de l'art

L'état de l'art présenté ici dresse un aperçu des grandes familles de méthodes principalement utilisées dans la résolution des problèmes de moindres carrés. Étant donné que l'algorithme sur lequel je travaille date des années 80, je me suis surtout concentré sur des ouvrages datant de la même époque, ou à peu près, afin de mieux comprendre les liens et différences entre ces différentes méthodes.

De plus, comme je n'étais pas familier avec les problèmes de moindres carrés, lire des articles présentant les principes généraux de ces méthodes m'a également fait monter en compétences rapidement sur ce type de problèmes. Cela me permettra notamment de m'intéresser à des articles présentant des méthodes plus complexes comprenant des variations par rapport à celles présentées dans cet état de l'art, comme c'est par exemple le cas pour les méthodes proposées par Orban et Siqueira [15] et Audibert et Catoni [1].

Un autre axe de travail important sera, lorsque je commencerai à travailler à l'amélioration proprement dite de l'algorithme ENLSIP utilisé par Hydro-Québec, d'orienter mes recherches sur des approches de type primal-dual, comme par exemple celle exposée par Wächter [17] pour une méthode de points intérieurs.

Cette dernière est par ailleurs implémentée dans le solveur IPOPT, qui a été utilisé pour comparer les performances de mon implémentation en Julia (voir section 4.5). Ce type de méthodes a

Étape 0 : Initialisation. Le point initial x_0 et un rayon initial de la région de confiance Δ_0 sont donnés. Les constants η_1, η_2, γ_1 et γ_2 sont aussi données et satisfont :

$$0 < \eta_1 \leq \eta_2 < 1 \text{ et } 0 < \gamma_1 \leq \gamma_2 < 1.$$

On calcule $f(x_0)$ et on initialise k à 0.

Étape 1 : Définition du modèle. On choisit une norme $\|.\|_k$ et on définit un modèle m_k sur $\mathcal{B}_k$.

Étape 2 : Calcul du pas. On calcule un pas s_k qui diminue "suffisamment" le modèle m_k et tel que $x_k + s_k \in \mathcal{B}_k$.

Étape 3 : Acceptation du nouveau point. On calcule $f(x_k + s_k)$ et on définit :

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}. \tag{2.20}$$

Si $\rho_k \geq \eta_1$, alors on définit $x_{k+1} = x_k + s_k$; sinon, $x_{k+1} = x_k$.

Étape 4 : Mise à jour du rayon de la région de confiance.

$$\Delta_{k+1} = \begin{cases} [\Delta_k, +\infty[& \text{si } \rho_k \geq \eta_2, \\ [\gamma_2 \Delta_k, \Delta_k] & \text{si } \rho_k \in [\eta_1, \eta_2], \\ [\gamma_1 \Delta_k, \gamma_2 \Delta_k] & \text{si } \rho_k < \eta_1. \end{cases} \tag{2.21}$$

On incrémente k de 1 et on retourne à l'étape 1.

FIGURE 2.1 – Exemple d'algorithme de régions de confiance présenté par Conn *et al.* [3]

l'avantage de mieux tirer parti des contraintes et des multiplicateurs de Lagrange, comparativement à ce qui est fait dans ENLSIP.

Chapitre 3

Description de la méthode d'ENLSIP

L'analyse approfondie des articles de Lindström et Wedin [11, 12] sur la méthode d'ENLSIP (Easy Nonlinear Least Squares Inequality Programming), ainsi que du code source de l'algorithme écrit en Fortran77 par ces derniers, m'a permis de comprendre le fonctionnement de cette méthode d'optimisation. Ce chapitre, réalisé sur recommandation de mon maître de stage, relate la modélisation des principaux aspects théoriques de l'algorithme.

3.1 Principe général

On se place dans le cadre où le problème d'optimisation à résoudre est celui présenté en (2.1), soit de moindres carrés non linéaires sous contraintes . Tout d'abord, il s'agit d'une méthode itérative avec longueur de pas. On rappelle que cela consiste, en partant d'un point de départ x_0, à construire une suite d'itérés $(x_k)_{k\in\mathbb{N}}$ convergeant vers la solution du problème et chaque nouvel itéré x_{k+1} est construit à partir du précédent par la relation :

$$x_{k+1} = x_k + \alpha_k p_k,$$

avec :

- $p_k \in \mathbb{R}^n$ la direction de descente ;
- $\alpha_k \in \mathbb{R}$ la longueur de pas.

Ces deux grandeurs sont calculées à chaque nouvelle itération.

Un autre aspect important de la méthode concerne la gestion des contraintes. Ce point particulier est d'ailleurs une des spécificités d'ENLSIP. En effet, comme vu au chapitre 2, la prise en compte des contraintes de tout type sans hypothèses particulières, hormis la différentiabilité de ces dernières, est relativement rare dans la littérature sur les problèmes de moindres carrés. Cela était d'autant plus le cas dans les années 1980, lors de la période de conception de cet algorithme.

Dans ENLSIP, les auteurs, Lindström et Wedin [12] ont mis en place une approche dite EQP, pour Equality Quadratic Programming, inspirée des travaux de Gill *et al.* [5] sur des problèmes d'optimisation quadratique sous contraintes d'inégalité linéaires. Ce procédé consiste à ne travailler à chaque itération qu'avec des contraintes d'égalité. Nous verrons en 3.2 comment on peut alors se ramener à la résolution de systèmes d'équations linéaires.

À chaque itération, on considère un ensemble réduit de contraintes que l'on estime être actives à la solution optimale x^*, c'est-à-dire égales à 0. Ce dernier est constitué de toutes les contraintes d'égalité ainsi que de certaines inégalités. Cette prédiction est mise à jour au début de chaque itération par le retrait ou l'ajout potentiel d'une contrainte d'inégalité à cet ensemble. L'objectif de cette manœuvre est qu'à mesure que l'on se rapproche de la solution du problème, la prédiction converge vers les contraintes effectivement actives.

Cet ensemble est noté $\mathcal{W}$, pour working-set, et comprend les indices des contraintes considérées comme actives à la solution. Nous verrons en 3.3 comment s'effectue l'ajout ou le retrait d'une contrainte et les implications sur la méthode. L'algorithme 1 présente les grandes étapes de la méthode implémentée dans ENLSIP.

Algorithm 1 Modèle de l'algorithme ENLSIP

Require: x_0 := point initial
  $k \leftarrow 0$
  Initialisation de l'ensemble actif
  repeat
    Mise à jour de l'ensemble actif $\mathcal{W}$
    Calcul de la direction de descente p_k
    Calcul de la longueur de pas α_k
    $x_{k+1} \leftarrow x_k + \alpha_k p_k$
    $k \leftarrow k + 1$
  until critère d'arrêt
  return x_k

Les notations utilisées dans la suite de ce chapitre pour les différentes modélisations mathématiques sont celles du chapitre 2.

3.2 Calcul de la direction de descente

À une itération k donnée, partant du point x_k qui est fixé, la direction de descente p_k est calculée par la résolution d'un problème de type :

$$\begin{cases} \min\limits_{p \in \mathbb{R}^n} \dfrac{1}{2}\|r(x_k + p)\|^2, \\ \text{s.c.} \\ c(x_k + p) = 0. \end{cases} \tag{3.1}$$

La multi-fonction c modélise ici $t \in \mathbb{N}^*$ contraintes d'égalité. On note A sa matrice jacobienne au point x_k. Bien que le problème initial (2.1) puisse présenter des contraintes d'inégalité, nous verrons dans la section3.3 comment ces dernières sont manipulées afin de se ramener au cas où toutes les contraintes sont des égalités.

3.2.1 Linéarisation du problème

La méthode mise en place par Lindström et Wedin [12] est une approche de type Gauss-Newton, telle que présentée à la section 2.3 mais appliquée ici au cas sous contraintes. On s'appuie toujours sur la linéarisation des résidus comme en (2.12) mais également sur celle des contraintes, donnée

par :

$$c(x_k + p) \approx Ap + c(x_k).$$

Ces deux linéarisations amènent à reformuler le problème (3.1) comme un problème sous contraintes linéaires :

$$\begin{cases} \min\limits_{p \in \mathbb{R}^n} \dfrac{1}{2} \|Jp + r(x_k)\|^2, \\ \text{s.c.} \\ Ap + c(x_k) = 0. \end{cases} \tag{3.2}$$

3.2.2 Décomposition en sous-systèmes

Le cœur du calcul de la direction descente réside dans l'utilisation de la décomposition matricielle QR. On souhaite d'abord factoriser la matrice A de taille $t \times n$, $(t \leq n)$ en passant par la factorisation QR de sa transposée :

$$A^T P_a = Q_a \begin{pmatrix} R_a \\ 0 \end{pmatrix}, \tag{3.3}$$

avec :

- Q_a matrice orthogonale $(n \times n)$;
- R_a matrice triangulaire supérieure $(t \times t)$ à éléments diagonaux décroissants en valeur absolue ;
- P_a matrice de permutation $(n \times n)$.

Comme P_a est une matrice de permutation, $P_a^{-1} = P_a^T$ et (3.3) se récrit :

$$A = P_a \begin{pmatrix} L_a \ , \ 0 \end{pmatrix} Q_a^T, \tag{3.4}$$

où $L_a = R_a^T$ est une matrice triangulaire inférieure $(t \times t)$ à éléments diagonaux décroissants en valeur absolue.

Injectant cette factorisation dans (3.2), nous obtenons :

$$Ap = -c(x_k) \iff P_a \begin{pmatrix} L_a \ , \ 0 \end{pmatrix} Q_a^T p = -c(x_k).$$

Posant $Q_a^T p = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$ avec $p_1 \in \mathbb{R}^t$ et $p_2 \in \mathbb{R}^{n-t}$, on a :

$$P_a L_a p_1 = -c(x_k) \iff L_a p_1 = -P_a^T c(x_k) = b.$$

On remarque que p_1, soit les t premiers éléments de p, est totalement déterminé par les contraintes tandis que p_2, soit les $n - t$ derniers éléments de p, peut être choisi librement. Cela se comprend en introduisant l'espace nul de A. Notons Y le bloc des t premières colonnes de Q_a et Z celui des $n - t$ dernières colonnes de Q_a. On remarque que $AZ = 0$ d'où :

$$Ap = AQ_a \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = AY p_1.$$

La stratégie va donc désormais consister à calculer p_1, entièrement déterminé par les contraintes puis, une fois p_1 fixé, de calculer p_2 de sorte à minimiser la fonction objectif du problème linéarisé (3.2).

En injectant p_1 et p_2 dans cette dernière, on peut alors décomposer notre problème en deux nouveaux sous-problèmes :

$$\begin{cases} L_a p_1 = b, \\ \min\limits_{p_2} \dfrac{1}{2} \|J_2 p_2 + J_1 p_1 + r(x_k)\|^2, \end{cases}$$

en introduisant $JQ_a = \begin{pmatrix} J_1 & J_2 \end{pmatrix}$ avec J_1 et J_2 respectivement de taille $(m \times t)$ et $(m \times (n-t))$.

3.2.2.1 Résolution avec stabilisation

Si la matrice A est de rang déficient, il faut réaliser ce qui est appelé dans [12] une stabilisation. Cela passe par la factorisation QR de L_a :

$$L_a P_\ell = Q_\ell R_\ell, \tag{3.5}$$

avec :

- Q_ℓ matrice orthogonale $(t \times t)$;
- R_ℓ matrice triangulaire supérieure $(t \times t)$ à éléments diagonaux décroissants en valeur absolue ;
- P_ℓ matrice de permutation $(t \times t)$.

Partant de $L_a p_1 = b$, le système d'inconnue p_1 devient alors :

$$\begin{aligned} & Q_\ell R_\ell P_\ell^T p_1 = b \\ \Longleftrightarrow & R_\ell P_\ell^T p_1 = Q_\ell^T b \\ \Longleftrightarrow & R_\ell P_\ell^T p_1 = b_1 \text{ avec } b_1 = Q_\ell^T b. \end{aligned}$$

Pour $\omega_1 \leq t$, on définit $R_\ell^{(\omega_1)}$ comme la sous-matrice de R_ℓ constituée des éléments r_{ij}, avec $i \leq \omega_1, j \leq \omega_1$.

Le vecteur $b_1^{(\omega_1)}$ est constitué des ω_1 premiers éléments de b_1.

Par suite, on pose $\delta p_1^{(\omega_1)}$ la solution du système triangulaire d'inconnue y :

$$R_\ell^{(\omega_1)} y = b_1^{(\omega_1)}.$$

On obtient alors :

$$p_1 = P_\ell \begin{pmatrix} \delta p_1^{(\omega_1)} \\ 0 \end{pmatrix}. \tag{3.6}$$

Le vecteur p_1 étant désormais calculé, il reste à résoudre pour p_2 le sous-problème :

$$\min_{p_2} \frac{1}{2} \|J_2 p_2 + (J_1 p_1 + r(x_k))\|^2.$$

Intervient alors la factorisation QR de J_2 :

$$J_2 = Q_2 \begin{pmatrix} R_2 \\ 0 \end{pmatrix} P_2, \tag{3.7}$$

avec :

- Q_2 matrice orthogonale $(m \times m)$;
- R_2 matrice triangulaire supérieure $((n-t) \times (n-t))$ à éléments diagonaux décroissants en valeur absolue ;
- P_2 matrice de permutation $((n-t) \times (n-t))$.

On pose $d_2 = -Q_2^T(r(x_k) + J_1 p_1)$ et pour $\omega_2 \leq n-t$, on définit $R_2^{(\omega_2)}$ comme le bloc triangulaire supérieur composé des ω_2 premières lignes et colonnes de R_2 et $d_2^{(\omega_2)}$ le vecteur constitué des ω_2 premiers éléments de d_2.

Par suite, posant $\delta p_2^{(\omega_2)}$ solution de $R_2^{(\omega_2)} y = d_2^{(\omega_2)}$, on obtient :

$$p_2 = P_2^T \begin{pmatrix} \delta p_2^{(\omega_2)} \\ 0 \end{pmatrix}. \tag{3.8}$$

3.2.2.2 Résolution sans stabilisation

.

Si la matrice A est de rang plein, on peut résoudre directement le système triangulaire inférieur $L_a p_1 = b$ d'inconnue p_1 sans passer par la factorisation QR de L_a. Cela revient à adopter le même raisonnement qu'au paragraphe 3.2.2.1 sur la matrice L_a avec en particulier $\omega_1 = \text{rang}(A)$.

Le vecteur p_1 étant désormais fixé, il reste à résoudre pour p_2 :

$$\min_{p_2} \frac{1}{2} \|J_2 p_2 + (J_1 p_1 + r(x_k))\|^2$$

On procède comme dans le cas avec stabilisation en se servant de la factorisation QR de J_2.

On pose $d = -Q_2^T(r(x_k) + J_1 p_1)$ et pour $\omega_2 \leq n-t$, on définit $R_2^{(\omega_2)}$ comme le bloc triangulaire supérieur composé des ω_2 premières lignes et colonnes de R_2 et $d^{(\omega_2)} = (d_1, d_2, \ldots, d_{\omega_2})^T$.

Par suite, posant $\delta p_2^{(\omega_2)}$ solution de $R_2^{(\omega_2)} y = d^{(\omega_2)}$, on obtient :

$$p_2 = P_2^T \begin{pmatrix} \delta p_2^{(\omega_2)} \\ 0 \end{pmatrix}. \tag{3.9}$$

3.2.2.3 Solution du sous-problème

Finalement, dans les deux cas, la solution du problème sous contraintes linéaires (2.8), déterminée en dimensions (ω_1, ω_2) est donnée par :

$$p^{(\omega_1,\omega_2)} = Q_a \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}. \tag{3.10}$$

Dans le déroulement de l'algorithme, la direction de descente est d'abord calculée en prenant $\omega_1 = \text{rang}(A)$ et $\omega_2 = \text{rang}(J_2)$.

La méthode propose ensuite d'examiner le vecteur alors retourné grâce à la vérification de différents critères, non explicités ici. Leur non satisfaction peut amener à refaire le calcul de la direction de descente en employant une des deux méthodes suivantes :

- une méthode similaire à celle détaillée ci-dessus mais avec des valeurs spécifiques de ω_1 et ω_2 dont le calcul est expliqué à la sous-section 3.2.3 ;
- une méthode de type Newton présentée à la sous-section 3.2.4.

La première permet de restreindre la recherche de direction de descente à un sous-espace de $\mathbb{R}^n$. La seconde permet théoriquement d'approcher plus vite de la solution si le point courant en est déjà proche.

3.2.3 Méthode de Gauss–Newton avec minimisation de sous-espace

Comme nous l'avons vu aux paragraphes 3.2.2.1 et 3.2.2.2, le calcul de la direction de descente nécessite les résolutions successives de systèmes triangulaires et un choix important porte sur le nombre de colonnes de la matrice triangulaire impliqué dans la résolution.

Une première solution consiste à prendre le nombre de colonnes correspondant au rang de la matrice. Néanmoins, dans certains cas de figure, il peut s'avérer intéressant de ne considérer qu'un nombre réduit de colonnes pour obtenir une meilleure solution du problème d'optimisation (3.1). Cela revient, pour chacun des deux systèmes, à restreindre la recherche de solution à des espaces de dimension inférieure au rang des dits systèmes.

On parle alors de minimisation de sous-espace, ou *subspace minimization* chez Lindström et Wedin [12]. Cette méthode ne diffère finalement de la méthode de Gauss–Newton présentée en amont que par les valeurs des entiers ω_1 et ω_2 utilisés au paragraphe 3.2.2.1, contrairement à la méthode de Newton présentée à la sous-section 3.2.4 qui s'appuie sur un sous-problème différent.

Pour la suite de cette partie, n désigne un entier naturel quelconque et on se place dans le cadre général de la résolution du système linéaire :

$$Tx = y. \tag{3.11}$$

Ce système est de dimension n et on a :

- $T = [T_{ij}]$ est une matrice triangulaire supérieure $(n \times n)$ et de rang $\bar{t} > 0$;
- $x = (x_1, \ldots, x_n)^T$ est l'inconnue ;
- $y = (y_1, \ldots y_n)^T$ est le second membre.

Afin faire le lien avec le paragraphe 3.2.2.1, le calcul de dimension sera appliqué au système $R_\ell y = b_1$ afin de calculer ω_1 puis au système $R_2 y = d_2$ pour déterminer ω_2.

Pour la suite, on définit également deux vecteurs $\tau = [\tau_i]_{1 \leq i \leq \bar{t}}$ et $\rho = [\rho_i]_{1 \leq i \leq \bar{t}}$ dont chacune des composantes est respectivement donnée par :

$$\tau_i = \left\| (y_1, \ldots, y_i)^T \right\|, \tag{3.12}$$

$$\rho_i = \left\| \left(\frac{y_1}{T_{11}}, \ldots, \frac{y_i}{T_{ii}} \right)^T \right\|. \tag{3.13}$$

On peut alors calculer une première dimension déterminée par l'indice k, pour k allant de 1 à $\bar{t}$, maximisant la quantité $\|(\tau_1, \ldots, \tau_k)\| * |T_{kk}|$.

Cet entier, noté mindim, est indiqué par les auteurs comme étant la plus petite dimension possible pour la résolution de (3.11).

Le calcul à proprement parler de la dimension du sous-espace se distingue en deux cas, dépendant de la méthode utilisée pour calculer la direction de descente à l'itération précédente.

Gauss–Newton à l'itération précédente

Cela concerne le cas où il n'y a pas eu besoin de refaire le calcul de la direction de descente à la précédente itération.

On prend alors comme dimension le plus grand entier k, pour k allant de mindim à $\bar{t}-1$, tel que :

$$\tau_k < \frac{\tau_{\bar{t}}}{5} \text{ et } \rho_k < \frac{\rho_{\bar{t}}}{2}.$$

Si un tel entier n'existe pas, on prend le maximum entre $\bar{t}-1$ et mindim.

Minimisation de sous-espace à l'itération précédente

Ce cas-ci s'applique lorsque la direction de descente de l'itération précédente a été calculée en utilisant la méthode de Gauss–Newton mais avec des valeurs de dimensions déterminées avec les principes explicités dans cette sous-section 3.2.3. La procédure du calcul de la dimension est décrite dans l'algorithme 2. Celle-ci consiste à analyser certaines composantes des vecteurs τ et rho afin de déterminer la dimension la plus adéquate pour l'itération en cours. La valeur de la dimension utilisée à l'itération précédente est également prise en compte.

Algorithm 2 subspace_previous$(\tau, \rho, \bar{t}, \omega)$

ω est la dimension utilisée pour résoudre le même système à l'itération précédente
dim $\leftarrow \max(1, \omega - 1)$
if $\omega > 1$ **and** $10 * \rho_{\text{dim}} > \rho_{\bar{t}}$ **then**
  return dim
else
  dim $\leftarrow \omega$
  if $\rho_{\text{dim}} > 0.7 * \rho_{\bar{t}}$ **and** $2 * \tau_{\text{dim}} < \tau_{\bar{t}}$ **then**
    return dim
  end if
else if $100 * \tau_{\text{dim}} < \tau_{\text{dim}+1}$ **then**
  return dim
else
  dim $\leftarrow \min_{\omega+1 \leq k \leq \bar{t}} \{k \mid \rho_k > 0.7\rho_{\bar{t}}\}$ {Si un tel indice n'existe pas, on prend la valeur $\bar{t}$}
  return dim
end if

3.2.4 Méthode de type Newton

3.2.4.1 Modèle du sous-problème

La direction de Newton est calculée via la résolution du problème quadratique sous contraintes d'égalité linéaires :

$$\begin{cases} \min\limits_{p \in \mathbb{R}^n} \left[\frac{1}{2} p^T \nabla^2_{xx} \mathcal{L}(x_k, \lambda) p + \nabla f(x_k)^T p \right], \\ \text{s.c.} \\ Ap = -c(x_k). \end{cases} \tag{3.14}$$

où $\nabla^2_{xx}\mathcal{L}(x,\lambda)$ désigne la matrice hessienne du lagrangien introduit en (2.7) par rapport à la variable x, dont voici le détail de l'expression analytique :

$$\begin{aligned}\nabla_x\mathcal{L}(x,\lambda) &= \nabla f(x) - \sum_{i\in\mathcal{W}} \lambda_i \nabla c_i(x) \\ &= J^T(x)r(x) - \sum_{i=1}^{t} \lambda_i \nabla c_i(x), \\ \nabla^2_{xx}\mathcal{L}(x,\lambda) &= J^T(x)J(x) + \sum_{i=1}^{m} r_i(x)\nabla^2 r_i(x) - \sum_{i=1}^{t} \lambda_i \nabla^2 c_i(x),\end{aligned}$$

où $\nabla^2 r_i$ (resp. $\nabla^2 c_i$) désigne la matrice hessienne du i-ème résidu (respectivement de la i-ème contrainte active).

La justification quant à l'utilisation de ce type de problème pour cette partie de l'algorithme n'a pas été trouvée dans le travail des auteurs, hormis la linéarisation des contraintes que l'on retrouve aussi dans la méthode de Gauss-Newton vue à la sous-section 3.2.1.

On posera $\Gamma = -\sum_{i=1}^{t} \lambda_i \nabla^2 c_i(x_k) + \sum_{i=1}^{m} r_i(x_k)\nabla^2 r_i(x_k)$ pour la suite.

La matrice Γ est symétrique comme combinaison linéaire de matrices symétriques.

On peut alors reformuler le problème sous la forme :

$$\begin{cases} \min\limits_{p\in\mathbb{R}^n} \dfrac{1}{2} p^T \left[J^T J + \Gamma\right] p + \left[J^T r(x_k)\right]^T p, \\ \text{s.c.} \\ Ap = -c(x_k), \end{cases} \tag{3.15}$$

3.2.4.2 Calcul de la direction de Newton

On suppose connues les factorisations QR des matrices A, L_a et J_2, respectivement données en (3.3), (3.5) et (3.7). On a également connaissance du rang de A.

Si $\text{rang}(A) = t$, soit $b = -P_a^T c(x_k)$, l'équation des contraintes s'écrit alors :

$$L_a p_1 = b.$$

Étant de rang t, ce système triangulaire inférieur se résout directement.

Si $\text{rang}(A) < t$, on injecte la factorisation QR de L_a dans l'équation des contraintes. Le premier système à résoudre devient alors :

$$R_\ell P_\ell^T p_1 = b_1,$$

avec $b_1 = Q_\ell^T b$.

Soit ω_r le rang de R_ℓ , $R_\ell^{(\omega_r)}$ est la sous-matrice de R_ℓ constituée des éléments r_{ij}, avec $i \leq \omega_r, j \leq \omega_r$.

Le vecteur $b_1^{(\omega_r)}$ est constitué des ω_r premiers éléments de b_1.

Le vecteur δp_1 est la solution du système triangulaire supérieur $R_\ell^{(\omega_r)} \delta p_1 = b_1$, ce qui donne :

$$p_1 = P_\ell \begin{pmatrix} \delta p_1 \\ 0 \end{pmatrix}.$$

Le calcul de p_1 se révèle similaire à celui fait avec la méthode de Gauss-Newton présenté au paragraphe 3.2.2.1.

Puisque $Q_a Q_a^T = I_n$, on a :

$$\begin{aligned}
&\frac{1}{2}p^T \left[J^T J + \Gamma\right] p + \left[J^T r(x_k)\right]^T p \\
&= \frac{1}{2}p^T \left[Q_a Q_a^T J^T J Q_a Q_a^T + Q_a Q_a^T \Gamma Q_a Q_a^T\right] p + \left[J^T r(x_k)\right]^T Q_a Q_a^T p \\
&= \frac{1}{2}(p_1^T \; p_2^T) \left[Q_a^T J^T J Q_a + Q_a^T \Gamma Q_a\right] \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \left[(JQ_a)^T r(x_k)\right]^T Q_a^T p \\
&= \frac{1}{2}(p_1^T \; p_2^T) W \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + h^T \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} \\
&= \varphi(p_2),
\end{aligned}$$

avec $W = Q_a^T J^T J Q_a + Q_a^T \Gamma Q_a$ matrice $n \times n$ symétrique et $h = (J_1 \; J_2)^T r(x_k) \in \mathbb{R}^n$.

À p_1 fixé, on souhaite minimiser φ afin de trouver la solution du problème (3.15).

On pose ensuite $E = Q_a^T J^T J Q_a = \begin{pmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{pmatrix}$ avec :

- E_{11} bloc $(t \times t)$;
- E_{12} bloc $(t \times (n-t))$;
- E_{21} bloc $((n-t) \times t)$;
- E_{22} bloc $((n-t) \times (n-t))$.

On applique le même découpage à W, soit $W = \begin{pmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{pmatrix}$.

Les matrices E et W sont évidemment symétriques et plus particulièrement, on a $W_{12}^T = W_{21}$.

On peut alors récrire, pour $v \in \mathbb{R}^{n-t}$:

$$\varphi(v) = \frac{1}{2}\left[2v^T W_{21} p_1 + v^T W_{22} v\right] + v^T J_2 r(x_k) + K$$

avec K indépendant de v.

Par suite :

$$\begin{aligned}
\nabla\varphi(v) &= W_{21} p_1 + W_{22} v + J_2 r(x_k), \\
\nabla^2\varphi(v) &= W_{22}
\end{aligned}.$$

Si W_{22} est définie positive, le minimum de φ s'obtient en annulant simplement son gradient.

Le vecteur p_2 est alors donné par la résolution du système d'inconnue v :

$$W_{22} v = -W_{21} p_1 - J_2 r(x_k)$$

Supposant W_{22} définie positive, on utilise la factorisation de Cholesky [6] de W_{22} pour se ramener à la résolution de deux systèmes triangulaires consécutifs, amenant au calcul explicite de p_2.

La solution du problème (3.15), est finalement donnée par :

$$p = Q_a \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}. \tag{3.16}$$

Si W_{22} n'est pas définie positive, alors la direction de descente n'est pas calculée dans l'itération en cours.

3.3 Mise à jour des contraintes actives

Afin de gérer les contraintes, Lindström et Wedin [12] ont mis en place une stratégie amenant à ne prendre en compte que certaines contraintes et de les modéliser comme des contraintes d'égalité. Cela n'est pertinent que pour les contraintes d'inégalité, les égalités devant par définition être actives à la solution.

3.3.1 Critère de suppression d'une contrainte

Au début de chaque itération de l'algorithme, on calcule une estimation des multiplicateurs de Lagrange associés aux contraintes de l'ensemble de travail $\mathcal{W}$ de l'itération précédente. Suite à l'analyse des valeurs de ces derniers, on décide ou non de retirer au plus une contrainte de $\mathcal{W}$.

Le critère de sélection des contraintes s'inspire de la stratégie d'ensemble actif EQP [5].

Dans cette approche, les multiplicateurs de Lagrange ne sont pas nécessairement réalisables, c'est-à-dire qu'il peut exister une contrainte d'inégalité d'indice s pour laquelle $\lambda_s < 0$.

Si notre estimation des multiplicateurs de Lagrange comporte une ou plusieurs composantes négatives, on décide alors de retirer une des contraintes associées à l'une de celles-ci.

En pratique, on choisit celle dont le multiplicateur est le plus petit, i.e. le plus grand en valeur absolue, parmi ceux qui sont strictement négatifs :

$$|\lambda_{deleted}| = \max_{j \in \mathcal{W}} \{|\lambda_j| \text{ si } \lambda_j < 0 \text{ et } c_j \text{ inégalité}\}.$$

Selon Gill *et al.* [5], ce choix permet en pratique de calculer des directions de descente faisant plus fortement diminuer la fonction objectif.

Cette contrainte que l'on supprime de $\mathcal{W}$ est alors considérée inactive et n'intervient pas dans les calculs relatifs à la direction de descente.

3.3.2 Calcul des multiplicateurs de Lagrange

Supposons que nous sommes en début d'itération k, l'ensemble de travail est inchangé par rapport à l'itération précédente.

La première estimation des multiplicateurs de Lagrange se fait en résolvant pour λ le système (3.17) :

$$A^T \lambda = \nabla f(x_k). \tag{3.17}$$

On résout ce système en utilisant la factorisation QR de A^T, afin de réécrire le système triangulaire supérieur d'inconnue $v = P_a^T \lambda$:

$$R_a v = Q_a^T \nabla f(x).$$

Notre première estimation des multiplicateurs de Lagrange est alors $\lambda = P_a v$. On procède comme décrit en 3.3.1 afin de déterminer quelle contrainte retirer de $\mathcal{W}$. On parle de première estimation car dans le cas où celle-ci n'implique aucune suppression de contrainte, on effectue une nouvelle estimation, supposée meilleure. Le vecteur alors obtenu est également passé au crible de notre critère de sélection.

Avant de réaliser cette seconde estimation, il faut calculer la direction de descente p_{GN} avec la méthode de Gauss-Newton en travaillant avec l'ensemble actif courant, auquel aucune contrainte n'a pour le moment été retirée.

La seconde estimation des multiplicateurs de Lagrange est alors la solution du système :

$$A^T \lambda = J^T \left[J p_{GN} + r(x_k) \right]. \tag{3.18}$$

Notons que le terme de droite peut aussi s'écrire $J^T J p_{GN} + \nabla f(x_k)$.

On résout ce système de façon analogue à celle de la première estimation.

Si aucune contrainte n'est à supprimer, on laisse l'ensemble actif inchangé. Sinon, on retire la contrainte de $\mathcal{W}$ et la ligne correspondant à cette contrainte de A. Cette dernière matrice étant modifiée, toutes les décompositions QR qui en découlent le sont également. Leur mise à jour est donc impérative pour la suite de l'algorithme. Cela se fait par un procédé développé chez Golub et Van Loan [6] faisant appel à des rotations de Givens.

Suite à ces opérations matricielles, on modélise toutes les contraintes comme des égalités et on peut initier le calcul de la direction de descente présenté à la section 3.2.

Si au nouvel itéré obtenu, une contrainte est ou bien violée, i.e. strictement négative dans notre cas, ou bien nulle, celle-ci est ajoutée à l'ensemble actif. Ce critère est évalué à chaque fin d'itération.

3.3.3 Mise à jour de la factorisation QR

La mise à jour de l'ensemble actif à chaque itération peut entraîner la modification de la matrice jacobienne des contraintes actives A. En effet, retirer une contrainte de l'ensemble $\mathcal{W}$ entraîne la suppression des coefficients de la ligne de A associés à cette contrainte. On note $\tilde{A}$ la matrice obtenue en retirant la ligne s de A, s étant donc l'indice de la contrainte supprimée lors de l'itération k. La matrice transposée de $\tilde{A}$ devient alors :

$$\tilde{A}^T = \begin{bmatrix} \nabla c_1(x_k) & \dots & \nabla c_{s-1}(x_k) & \nabla c_{s+1}(x_k) & \dots & \nabla c_t(x_k) \end{bmatrix} \in \mathbb{R}^{n \times (t-1)}. \tag{3.19}$$

La factorisation QR de A^T exposée en (3.3) n'est alors plus valable pour $\tilde{A}^T$, d'où la nécessité de mettre à jour cette dernière. Une première solution pourrait être de calculer la décomposition QR de la nouvelle matrice en appliquant à $\tilde{A}^T$ la même méthode de calcul que celle réalisée sur A^T. On construirait alors entièrement de nouveaux facteurs $\tilde{Q}_a$, $\tilde{R}_a$, et $\tilde{P}_a$ tel que :

$$\tilde{A}^T \tilde{P}_a = \tilde{Q}_a \begin{pmatrix} \tilde{R}_a \\ 0 \end{pmatrix}, \tag{3.20}$$

avec donc :

- $\tilde{Q}_a \in \mathbb{R}^{n \times n}$ matrice orthogonale ;
- $\tilde{R}_a \in \mathbb{R}^{(t-1) \times (t-1)}$ triangulaire supérieure et à éléments diagonaux décroissants en valeur absolue ;
- $\tilde{P}_a \in \mathbb{R}^{(t-1) \times (t-1)}$ matrice de permutation.

Néanmoins, les deux matrices A et $\tilde{A}$ ne différant que d'une colonne, il est possible de travailler directement sur les composantes Q_a, R_a et P_a de (3.3) et de modifier ces dernières pour obtenir les facteurs de la décomposition (3.20). Comme on dispose déjà de la factorisation QR de A^T,

cette stratégie s'avère avantageuse en termes de complexités temporelle et spatiale, l'opération de décomposition étant assez coûteuse en temps de calcul et en allocation d'espace mémoire.

La nouvelle permutation des colonnes se déduit facilement de l'ancienne. En effet, retirer une colonne de A^T ne change pas pour autant l'ordonnancement global des colonnes de $\tilde{A}^T$. Il s'agit donc d'ajuster la matrice P_a afin que l'ordre de permutation des colonnes de $\tilde{A}^T$ préserve celui des colonnes de A^T.

Exemple : Pour $t = 4$, prenons la permutation $(4, 2, 1, 3)$, revenant à avoir la matrice de permutation :

$$P_a = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Si l'on supprime la deuxième colonne de A^T, i.e. $s = 2$, la même permutation mais adaptée devient :

$$\tilde{P}_a = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

Le prochain exemple présenté est directement tiré des travaux de [6].

On se place ici dans le cas où $n = 7$, $t = 6$, $s = 3$.

La matrice de Hessenberg H correspond à la matrice R_a amputée de sa colonne s.

$$Q_a \tilde{A}^T \tilde{P}_a = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = H.$$

Les éléments sous-diagonaux $h_{s+1,s}, \ldots, h_{t,t-1}$ peuvent être annulés par l'application de rotations de Givens successives $G_{t-1}^T \ldots G_s^T H = \tilde{R}_a$. La rotation G_i agit sur les colonnes i et $i+1$. Ainsi, posant $\tilde{Q}_a = Q_a G_s \ldots G_{t-1}$, la factorisation QR de $\tilde{A}^T$ est donnée par :

$$\tilde{A}^T \tilde{P}_a = \tilde{Q}_a \tilde{R}_a. \tag{3.21}$$

3.4 Calcul de la longueur de pas

On considère que la direction de descente p_k a été calculée et on souhaite maintenant déterminer la longueur de pas.

On introduit alors la fonction de mérite :

$$\psi(x, w) = \frac{1}{2}\|r(x)\|^2 + \frac{1}{2}\sum_{i \in \mathcal{W}} w_i c_i(x)^2 + \frac{1}{2}\sum_{j \in \mathcal{I}} w_j \min(0, c_j(x))^2, \tag{3.22}$$

où $w \in \mathbb{R}^l$ désigne le vecteur de pénalités dont chacune des composantes est associée à une contrainte. Leurs valeurs seront définies dans la section suivante. L'ensemble $\mathcal{I}$ comprend les indices des contraintes inactives, c'est-à-dire qui ne sont pas dans l'ensemble de travail $\mathcal{W}$.

À l'itération k, le point courant x_k, la direction de descente p_k et les pénalités $w^{(k)}$ sont fixés. La longueur de pas est alors définie comme :

$$\alpha_k = \underset{\alpha \in [\alpha_{min}, \alpha_{max}]}{\operatorname{argmin}} \ \psi\left(x_k + \alpha p_k, w^{(k)}\right), \tag{3.23}$$

avec $\alpha_{max} = \min_{i \in \mathcal{I}} \left\{ -\dfrac{c_i(x_k)}{\nabla c_i(x_k)^T p_k} \text{ pour } i \text{ tel que } \nabla c_i(x_k)^T p_k < 0 \right\}$.

Si un tel minimum n'existe pas, on prend $\alpha_{max} = 3$.

On définit ensuite $\alpha_{min} = \alpha_{max}/3000$.

On note $\phi : \alpha \mapsto \psi(x_k + \alpha p_k, w^{(k)})$ pour la suite.

Calcul des pénalités

Introduites à la section précédente dans la fonction de mérite, ces pénalités servent à diriger la recherche du pas minimisant au mieux la fonction objectif tout en se maintenant dans un espace où les contraintes restent satisfaites et les linéarisations sont valables. En effet, si on choisit α tel les termes associés aux contraintes sont non nuls, alors la valeur de la fonction de mérite (3.22) va être importante. De plus, certaines contraintes seront violées. Ainsi, en minimisant la fonction ϕ, on favorise les longueurs de pas qui maintiennent la satisfaction des contraintes.

Le calcul des pénalités est réalisé à chaque itération avant de commencer celui du pas et fait l'objet d'un problème d'optimisation. Je ne suis néanmoins pas parvenu à comprendre la justification théorique de l'utilisation de la formulation du problème présenté en (3.24). C'est pourquoi la modélisation présentée ici s'appuie sur les informations que j'ai pu extraire du code source d'ENLSIP en Fortran77.

La description de cette partie de la méthode ENLSIP nécessite l'introduction de certaines grandeurs.

Tout d'abord, le vecteur κ est une collection de 4 différents vecteurs de pénalités. Comme nous le verrons ci-après, le calcul des pénalités fait intervenir celles calculées lors d'itérations précédentes, afin d'obtenir des pénalités de plus en plus grandes à mesure que l'on s'approche de la solution du problème initial 2.1. Seront ainsi fortement défavorisées par ces pénalités les directions de descente s'éloignant de la solution alors que le point courant en est proche.

L'intérêt d'un tel vecteur κ est de garder une trace des pénalités calculées au cours des itérations précédentes, afin de calculer les nouvelles pénalités $w^{(k)}$ sur la base des plus petites pénalités calculées lors de l'une des 4 dernières itérations, au lieu de simplement considérer celles de l'itération précédente. En pratique, cela évite d'obtenir dès les premières itérations des pénalités trop importantes, pouvant détériorer la convergence.

En particulier, $w^{(old)}$ est le dernier élément de κ, i.e. le vecteur contenant les plus petites pénalités calculées au cours des 4 itérations précédentes (le vecteur $\hat{w}^{(old)} \in \mathbb{R}^t$ correspond aux plus petites pénalités associées aux contraintes actives).

On définit également le vecteur

$$z = \left[(\nabla c_i(x_k)^T p_k)^2\right]_{1 \leq i \leq t},$$

ainsi que le réel

$$\mu = \left[\frac{|(Jp_k)^T r(x_k) + \|Jp_k\|^2|}{\delta} - \|Jp_k\|^2 \right],$$

où δ est une constante valant 0.25.

Les vecteurs $w^{(old)}$, z et le réel μ servent à paramétrer le problème d'optimisation à résoudre pour calculer w_k.

Problème d'optimisation à résoudre

Le vecteur w_k est défini comme la solution du problème :

$$\begin{cases} \min\limits_{w \in \mathbb{R}^l} \|w\|, \\ \text{s.c.} \\ y^T \hat{w} \geq \tau \ \ \left(\text{ou } y^T \hat{w} = \tau\right) \\ w_i \geq w_i^{(old)}, \ \text{pour } 1 \leq i \leq l. \end{cases} \tag{3.24}$$

Les composantes de $\hat{w}$ désignent celles de w associés aux indices des contraintes actives, $y \in \mathbb{R}^t$ et $\tau \in \mathbb{R}$ sont définis de différentes façons selon les cas. Le calcul de ces derniers est explicité en annexes sous forme d'algorithmes en pseudo-code.

Dans le cas où la première contrainte est une inégalité, alors les composantes de $w^{(k)}$ sont données par :

$$w_i^{(k)} = \max\left(\frac{\tau}{\|y\|^2} y_i, w_i^{(old)} \right), \ \text{pour } i \in \mathcal{W}.$$

Ceci assure que $y^T \hat{w}^{(k)} \geq \tau$.

Pour le cas où la contrainte considérée est une égalité, le principe reste le même sauf que la constante τ est modifiée lorsqu'il y a des indices i dans $\mathcal{W}$ pour lesquels :

$$\frac{\tau}{\|y\|^2} y_i < \hat{w}_i^{(old)}.$$

En ce qui concerne les pénalités des contraintes inactives, on choisit la valeur qui leur avait été donnée à l'itération précédente, soit les composantes associées de $w^{(old)}$.

Calcul de la longueur de pas

Lorsque les pénalités ont été calculées, on cherche à déterminer le point minimisant la fonction $\phi : \alpha \mapsto \psi(x_k + \alpha p_k, w^{(k)})$ sur l'intervalle $[\alpha_{min}, \alpha_{max}]$. On note ce point α^*.

L'idée pour le calcul de α^* est, d'au lieu de calculer directement le minimum de ϕ, de calculer le minimum d'approximations polynomiales successives de ϕ l'interpolant en deux, voire trois points selon les configurations. Ces minimisations sont réalisées sur des fonctions quadratiques, pour lesquelles on sait calculer les extrema sur un intervalle donné.

La méthode de calcul de pas, explicitée dans l'article [11], décrit le cas d'un problème initial sans contraintes, donc sans pénalités, mais le principe de la méthode implémentée dans ENLSIP reste

identique. Les grandes étapes de cette dernière sont listées à la figure 3.1, toutes les minimisations évoquées étant faites sur le même intervalle $[\alpha_{min}, \alpha_{max}]$.

Une méthode de type Armijo-Goldstein est utilisée dans le cas où passé un certain nombre d'approximations polynomiales de ϕ, l'algorithme ne parvient pas à fournir de résultat satisfaisant. La longueur de pas calculée avec cette méthode est cependant moins optimale que celle normalement renvoyée par l'algorithme 3.1. L'implémentation utilisée dans ENLSIP est décrite dans l'algorithme de la figure 3 située en annexes.

Le critère $\alpha\|p_k\| > \varepsilon_{rel}$ **or** $\alpha > \alpha_{min}$ est évalué sur la valeur renvoyée par la procédure de calcul de la longueur de pas. Si ce critère est satisfait, on retourne la valeur 1 comme longueur de pas pour l'itération en cours. Cela assure de ne pas renvoyer une longueur de pas trop petite, qui ne diminuerait donc que légèrement la valeur de la fonction objectif. Ce cas se produit notamment lorsque l'on est très proche d'un optimum local.

Construction des polynômes interpolateurs

Les spécificités de l'implémentation Fortran77 concernent premièrement le cas où l'on approche ϕ par un polynôme l'interpolant aux points 0 et α_i, pour i une étape de l'algorithme de la figure 3.1, fixé. En effet, les coefficients du polynôme interpolateur sont modifiés lorsque l'on considère les contraintes et les pénalités dans la fonction ϕ.

Les polynômes interpolateurs sont définis de la façon suivante.

Soit $P : \alpha \mapsto \frac{1}{2}\|v_2\alpha^2 + v_1\alpha + v_0\|^2$ avec v_0, v_1 et v_2 vecteurs de $\mathbb{R}^{(m+t)}$.

On note :

$$F : \alpha \mapsto \begin{pmatrix} r_1(x_k + \alpha p_k) \\ \vdots \\ r_m(x_k + \alpha p_k) \\ \sqrt{\hat{w}_1}c_1(x_k + \alpha p_k) \\ \vdots \\ \sqrt{\hat{w}_t}c_t(x_k + \alpha p_k) \end{pmatrix}.$$

Les vecteurs v_0, v_1 et v_2 s'expriment alors comme :

1. On initialise α_0 à 0 et α_1 à $\max(1, \alpha_{max})$.
2. On calcule α_2, abscisse du minimum de l'interpolation polynomiale de ϕ en α_1 et α_0.
3. On initialise l'entier i à 1.
4. Si $\phi(\alpha_{i+1}) - \phi(0) \leq \frac{1}{4}\phi'(0)\alpha_{i+1}$ ou $\phi(\alpha_{i+1}) \leq 0.4 * \phi(0)$ aller à 9.
5. On calcule α_{i+2}, abscisse du minimum de la fonction quadratique interpolant ϕ en $\alpha_{i+1},\ \alpha_i$ et α_{i-1}.
6. On remplace les valeurs de $\alpha_{i+1},\ \alpha_i$ et α_{i-1} respectivement par $\alpha_{i+2},\ \alpha_{i+1}$ et α_i.
7. On incrémente i de 1.
8. Retour à l'étape 4.
9. On renvoie la meilleure valeur entre α_i et α_{i+1}, soit celle pour laquelle ϕ est minimale.

FIGURE 3.1 – Algorithme de recherche linéaire pour le calcul du pas

$$
\begin{aligned}
v_0 &= F(0), \\
v_1 &= \left[\nabla r_1(x_k)^T p_k, \ldots, \nabla r_m(x_k)^T p_k, \sqrt{\hat{w}_1}\nabla c_1(x_k)^T p_k, \ldots, \sqrt{\hat{w}_t}\nabla c_t(x_k)^T p_k\right]^T, \\
v_2 &= \left[\frac{1}{\alpha_i}\left(\frac{F_j(\alpha_i) - v_0^{(j)}}{\alpha_i} - v_1^{(j)}\right)\right]_{1\leq j\leq m+t}^T.
\end{aligned}
\tag{3.25}
$$

Le polynôme P ainsi formé avec les coefficients ci-dessus et interpole la fonction ϕ en 0 et α_i. On a également $P'(0) = \phi'(0)$.

Ensuite, il est fait mention dans les calculs exposés dans la figure 3.1 notamment de la dérivée de ϕ en 0. Or, à cause de la fonction min dans l'expression de la fonction de mérite (3.22), ϕ n'est pas dérivable sur $\mathbb{R}$. Afin de palier à cela, on considère qu'au voisinage de 0, soit autour du point x_k :

$$
\phi(\alpha) \approx \frac{1}{2}\|r(x_k + \alpha p_k)\|^2 + \frac{1}{2}\sum_{i\in\mathcal{W}} w_i c_i(x_k + \alpha p_k)^2 = \phi_2(\alpha).
$$

Cette approximation revient à négliger le troisième terme de la fonction ϕ, qui représente les écarts d'irréalisabilité des contraintes inactives.

La fonction ϕ_2 étant quant à elle trivialement dérivable en 0, on peut alors utiliser $\phi_2'(0)$ comme valeur de la dérivée de ϕ en 0.

3.5 Critères d'arrêt

Les critères d'arrêt sont vérifiés en deux étapes à la fin de chaque itération. D'abord, des critères de convergence sont évalués afin de déterminer si le point courant est solution du problème. Si ce n'est pas le cas, des critères algorithmiques sont examinés pour savoir si l'exécution du programme doit être arrêtée ou non (le nombre d'itérations qui dépasse un seuil maximum fixé par l'utilisateur par exemple). Ces derniers n'ayant pas trait à la méthode d'optimisation en soit, seuls les critères de convergence seront détaillés ici.

Conditions nécessaires

Pour la suite, on note :

- x^* le point sur lequel sont analysés les critères de convergence ;
- $x_{previous}$ l'itéré précédent x^* ;
- p^* la dernière direction de descente calculée ;
- λ^* la dernière estimation des multiplicateurs de Lagrange ;
- λ^*_{min} est le plus petit multiplicateur de Lagrange positif parmi ceux associés aux contraintes d'inégalité actives ;
- $|\lambda^*_{max}| = \max_i |\lambda_i|$.

Pour être qualifié de solution potentielle, le point x^* doit vérifier les conditions nécessaires :

$$\sum_{i \in \mathcal{W}} c_i(x^*)^2 < \varepsilon_h, \tag{3.26}$$

$$\forall j \notin \mathcal{W}, c_j(x^*) > 0, \tag{3.27}$$

$$\left\| A^T \lambda^* - \nabla f(x^*) \right\| < \sqrt{\varepsilon_{rel}} \left(1 + \|\nabla f(x^*)\|\right), \tag{3.28}$$

$$\frac{\lambda^*_{min}}{\lambda^*_{max}} \geq \varepsilon_{rel}. \tag{3.29}$$

Les réels ε_{rel}, ε_x, ε_{abs} et ε_h désignent des seuils de tolérance pour les critères associés et sont en pratique égaux à la précision machine.

Ces critères permettent de vérifier si x^* est bien un point réalisable du problème (2.1).

En effet, les conditions (3.26) et (3.27) assurent respectivement que les contraintes considérées comme actives le sont effectivement et que les inégalités restantes sont bien strictement positives. Notons que si une inégalité absente de l'ensemble de travail $\mathcal{W}$ est quand même nulle, le point x^* est considéré comme non réalisable.

Les deux suivantes jugent du respect des conditions KKT (2.6). En particulier, la condition (3.28) permet de vérifier que l'estimation des multiplicateurs de Lagrange est correcte. L'inégalité de la condition (3.29) a trait au conditionnement du système (3.17) et contrôle la validité de la solution λ^* d'un point de vue numérique.

Conditions suffisantes

Si les conditions nécessaires exposées précédemment sont validées, on évalue ensuite les conditions suffisantes :

$$\|d\|^2 \leq \varepsilon_{rel}^2 \|r(x^*)\|^2, \tag{3.30}$$

$$\|r(x^*)\|^2 \leq \varepsilon_{abs}^2, \tag{3.31}$$

$$\|x_{previous} - x^*\| < \varepsilon_x \|x^*\|, \tag{3.32}$$

$$\frac{\|p^*\|}{4} < \sqrt{\varepsilon_{rel}}. \tag{3.33}$$

Ces conditions indiquent si l'on peut stopper l'exécution de l'algorithme et renvoyer x^* comme solution.

On rappelle que d est le second membre du second système résolu pour calculer la direction de descente. Si la condition (3.30) est vérifiée, cela implique que la matrice jacobienne des résidus, autour de x^* est quasiment égale à la matrice nulle et qu'il s'agit donc d'un point stationnaire. La condition (3.31) teste si la valeur de la fonction objectif est arbitrairement petite, 0 étant le plus petit minimum potentiellement atteignable dans des moindres carrés. Ces critères sont en pratique rarement satisfaits, les contraintes pouvant empêcher que la solution soit un point stationnaire ou encore le meilleur paramètre possible.

Les deux autres critères évaluent plutôt si le progrès réalisé entre deux itérations successives est plus ou moins significatif. Cela se caractérise par la proximité arbitraire de deux itérés successifs (condition (3.32)) ou bien par la faible valeur en norme de la prochaine direction de descente (condition (3.33)).

3.6 Commentaires sur la modélisation

Comme dit en préambule de ce chapitre, la description proposée ici s'est appuyée sur l'analyse approfondie des travaux de Lindström et Wedin [11, 12] et du code source d'ENLSIP. Cependant, le contenu de ces articles s'est parfois avéré éloigné de ce qui était fait en pratique dans l'algorithme. Par exemple, le calcul des pénalités décrit dans la section 3.4, effectivement implémenté dans ENLSIP, n'est pas semblable à celui que l'on retrouve dans [12]. Il en va de même pour les critères d'arrêt qui sont bien plus nombreux que ceux de l'article de référence ; idem pour le calcul des dimensions dans la méthode de minimisation de sous-espace vue dans la sous-section 3.2.3.

Chapitre 4

Implémentation d'ENLSIP en Julia

Ce chapitre a pour but de décrire la retranscription en Julia de l'algorithme ENLSIP que j'ai réalisée et dont les aspects théoriques ont été décrits au chapitre 3. Nous discuterons également les choix faits en terme de programmation pour tirer parti au mieux des spécificités de Julia. Enfin, les résultats numériques obtenus sur les différents tests effectués seront présentés.

4.1 Approche globale

L'objectif est de produire un algorithme d'optimisation appliquant la méthode ENLSIP et obtenant les mêmes résultats, à la précision machine près, que la version en Fortran77. C'est pour cette raison que j'ai mis en œuvre le même déroulement d'ensemble que celui présenté en (1). Sur conseil de mon maître de stage, j'ai repris la quasi-totalité des fonctions Fortran appelées dans l'algorithme afin de favoriser la concordance des deux implémentations.

Comme mon travail a pour finalité d'être déployé en contexte industriel par Hydro-Québec, les similarités entre les deux versions s'étendent aussi à la nature des paramètres donnés en entrée pour exécuter le solveur.

Un aspect sous-jacent de ce projet étant d'améliorer la maintenance de l'algorithme, j'ai également cherché à améliorer l'accessibilité de certaines portions du code, jugées plus difficiles à appréhender en Fortran77.

Pour les fonctions où il a été difficile d'extraire les aspects théoriques, je n'ai pas cherché à complètement modifier le code. J'ai plutôt tâché de retranscrire le plus fidèlement possible ce qui était fait en Fortran77 en adaptant seulement la syntaxe au langage Julia.

4.2 Utilisation de librairies Julia

Bien qu'ENLSIP fasse appel à des fonctions issues de certaines libraires, notament LINPACK pour tout ce qui a trait aux calculs d'algèbre linéaire, le langage Julia donne accès à un nombre important de librairies présentant des caractéristiques intéressantes.

Module LinearAlgebra

Comme vu au chapitre 3, le calcul matriciel, la factorisation QR et la résolution de systèmes linéaires occupent une place très importante dans la méthode ENLSIP. On en retrouve notam-

ment dans l'estimation des multiplicateurs de Lagrange ainsi que dans le calcul de la direction de descente.

Pour implémenter ces fonctionnalités, je me suis servi de la librairie Julia `LinearAlgebra`. Cette dernière comprend une grande quantité de fonctions pour le calcul matriciel dont la plupart sont héritées de l'API OpenBLAS. Les aspects théoriques des fonctions disponibles sont majoritairement issus des travaux de Golub et Van Loan [6]. Cet ouvrage présente des méthodes de calcul matriciel avantageuses en termes de complexité temporelle et spatiale.

L'exploitation de ce module a grandement contribué à améliorer la lisibilité du code. Par exemple la factorisation QR d'une matrice s'obtient par un appel à une seule fonction. Il est également possible de faire pivoter les colonnes de la matrice de départ pour obtenir un résultat analogue à celui présenté en (2.10). La version Fortran77 effectue, quant à elle, d'avantages d'étapes de calcul intermédiaires.

De même, la résolution de systèmes linéaires triangulaires a bénéficié du passage à Julia. D'une part, cette opération ne nécessite qu'un seul appel de fonction ; d'autre part, Julia étant un langage fortement typé, la structure des matrices triangulaires est exploitée pour améliorer les performances de la résolution.

Module Polynomials

Comme vu en 3.4, le calcul du pas fait intervenir des calculs de polynômes à minimiser sur un intervalle donné. Je me suis donc servi du module `Polynomials` qui permet de modéliser des polynômes à partir de leurs coefficients. La méthode pour la longueur de pas ne faisant intervenir que des polynômes de degré 4, j'ai également eu un accès facilité au calcul direct des racines et extrema.

Néanmoins, en comparant certains résultats intermédiaires d'ENLSIP-Fortran avec ceux de mon implémentation en Julia, j'ai constaté certaines différences qui amenaient à des longueurs de pas différentes. J'ai donc réduit l'utilisation de la librairie `Polynomials` et ai retranscrit certaines fonctions Fortran. Ainsi j'ai réussi à obtenir des résultats identiques.

4.3 Gestion des variables internes

Un autre aspect important concerne la gestion des variables internes dans le déroulement global du solveur d'un point de vue algorithmique.

Ayant été conçu à une période où les contraintes de mémoire des ordinateurs étaient fortes, la lisibilité du code Fortran77 d'ENLSIP s'en retrouve fortement touchée. En effet, la plupart des variables de l'algorithme sont modifiées dans le corps même des différentes fonctions. Cela implique qu'une variable peut prendre différentes valeurs ayant une signification différente selon les fonctions où elle intervient.

Cette logique d'économie de la mémoire n'est plus vraiment d'actualité et ce indépendamment de l'utilisation du langage Julia. En revanche, dans le but de correspondre à l'algorithme en Fortran, j'ai maintenu ce paradigme de modifier les variables dans le corps même des fonctions. Cela a nécessité certaines adaptations car en Julia il n'est par exemple pas possible de sauvegarder les modifications de nombres entiers ou flottants. C'est néanmoins possible sur les objets s'apparentant à des structures de données, comme les `arrays`.

Puisque Julia offre la possibilité de créer ses propres types via l'utilisation de `structures`, j'ai

pu adapter mon implémentation et mettre en place des éléments absentes du codage d'ENLSIP en Fortran77. En effet, j'ai favorisé l'intégration de nouvelles structures de données, comportant différents attributs, pour une gestion plus efficace des variables de l'algorithme.

Les différents types que j'ai créés et implémentés sont listés ci-dessous :

Iteration : contient toutes les informations utiles d'une itération passée (point initial, direction de descente, longueur de pas...) ;

WorkingSet : comprend les indices des contraintes actives ;

ActiveConstraint : contient l'évaluation des contraintes actives ainsi que leur matrice jacobienne.

Les `structures` présentent également l'avantage de sauvegarder les modifications d'attributs réalisées dans le cœur des fonctions.

4.4 Difficultés rencontrées

Le code source fourni avec la librairie ENLSIP étant très documenté, je disposais d'une très bonne base pour transposer la méthode en Julia. L'algorithme ayant été utilisé sans être modifié par Hydro-Québec pendant près de trente ans, sa robustesse n'était plus à prouver et je savais également à quels résultats je devais aboutir. C'est pour cela que j'ai volontairement limité les évolutions dans mon travail d'implémentation, hormis celles décrites ci-dessus.

En revanche, l'analyse du code source en vue de comprendre ce que faisait chacune des fonctions intermédiaires s'est révélée être une tâche plus longue et minutieuse que ce qui était initialement prévu. De plus, ma connaissance du Fortran, surtout dans sa version 77, était très lacunaire au début du projet. J'ai donc dû m'ajuster à une syntaxe différente de ce à quoi j'étais habitué dans un algorithme très complexe et mettant en œuvre une méthode d'optimisation dont je ne connaissais pas tous les aspects théoriques.

Tout ce travail a néanmoins mené à la retranscription complète de l'algorithme en Julia, ce qui constitue déjà une étape importante du projet.

4.5 Résultats numériques

Pour vérifier la bonne mise en œuvre de l'algorithme en Julia, plusieurs tests sur des problèmes de moindres carrés non linéaires sous contraintes ont été réalisés. Au moment de l'écriture de ce rapport, je ne dispose pas encore des ressources nécessaires pour exécuter l'algorithme sur des problèmes se rapportant au contexte industriel d'Hydro-Québec. Les tests se restreignent donc à la résolution de problèmes mathématiques en petite dimension, soit avec au plus cinq paramètres.

Cela permet néanmoins d'examiner les deux points suivants : d'une part comparer la concordance des résultats entre les versions Julia et Fortran77 et d'autre part, comparer les performances de cette méthode d'optimisation avec d'autres solveurs de programmation non linéaire.

La suite de cette section présente les résultats obtenus sur les différents tests qui ont pu être réalisés.

4.5.1 Élaboration et modélisation des problèmes de test

Les trois problèmes m'ayant servi de base de test sont extraits de la collection de problèmes d'optimisation de Hock et Schittkowski [9]. Cet ouvrage comptabilise près d'une centaine de

problèmes de programmation non linéaire sous contraintes conçus pour tester des algorithmes d'optimisation non linéaire.

Chaque problème est documenté avec les informations suivantes :

- l'expression analytique de la fonction objectif et des contraintes ;
- un point initial ;
- la solution attendue avec la valeur de la fonction objectif associée ;
- les éventuelles données ou valeurs de paramètres nécessaires au calcul des contraintes ou de la fonction objectif.

Ces informations ont favorisé la vérification des résultats obtenus. Je n'ai volontairement pas fait de tests en modifiant le point départ. Étant donné que ma priorité était de vérifier que mon implémentation fonctionnait correctement, j'ai préféré me restreindre dans un cadre où les solutions et les points initiaux associés étaient connus.

Mon maître de stage et moi même avons sélectionné exclusivement des problèmes de moindres carrés, étant donné que c'est pour cette catégorie de problèmes que l'algorithme est conçu.

Les tests avec d'autres algorithmes d'optimisation se sont limités à l'utilisation de la version Julia du solveur open source IPOPT [17]. Sa conception repose sur une méthode de points intérieurs avec une approche primal-dual.

Contrairement à ENLSIP, son emploi n'est pas restreint aux moindres carrés mais sa robustesse et sa capacité à résoudre des problèmes présentant des contraintes non linéaires en font un élément de comparaison pertinent.

4.5.2 Concordance avec ENLSIP-Fortran77

Afin de comparer les résultats des deux implémentations d'ENLSIP, je me suis servi d'un problème test dont l'implémentation en Fortran77 avait déjà été proposée par Lindström et Wedin [12]. Il s'agit du problème numéro 65 issu de [9] ; sa modélisation est donnée en (4.1) :

$$\begin{cases} \min (x_1 - x_2)^2 + \dfrac{(x_1 + x_2 - 10)^2}{9} + (x_3 - 5)^2, \\ \text{s.c.} \\ 48 - x_1^2 - x_2^2 - x_3^2 \geq 0, \\ -4.5 \leq x_i \leq 4.5, \ \text{pour } i = 1, 2, \\ -5 \leq x_3 \leq 5. \end{cases} \tag{4.1}$$

Point initial $x_0 = (-5, 5, 0)^T$;
Solution attendue $x^* = (3.650461821, 3.65046168, 4.6204170507)^T$;
Fonction objectif à la solution $f(x^*) = 0.9535288567$.

La fonction objectif s'identifie à la norme au carré de la fonction de résidus :

$$r : x \longmapsto \left(x_1 - x_2, \frac{x_1 - x_2 - 10}{3}, x_3 - 5 \right)^T.$$

Ce problème a également été une constante source d'améliorations de mon algorithme puisque j'ai pu l'utiliser en parallèle du développement de mon implémentation. En effet, disposant des détails

de chaque itération de l'exécution en Fortran77, j'ai pu apporter énormément de corrections à mon code en m'appuyant sur les différents résultats intermédiaires observés en Fortran77 et que j'étais censé obtenir avec le code en Julia.

Les figures 4.1 et 4.2 présentent le déroulement itération par itération des implémentations d'ENLSIP en Fortran et Julia. Les informations affichées correspondent aux grandeurs suivantes :

(iter) numéro de l'itération ;

(fsum et objective) valeur de la fonction objectif ;

(hsum et cx_sum) norme du vecteur des contraintes actives ;

(dxnorm et $\|\mathbf{p}\|$) norme de la direction de descente ;

(koda, kocd et dimA, dimJ2) dimensions utilisées pour le calcul de la direction de descente ;

(alpha et α) longueur de pas ;

(conv. speed) facteur évaluant la vitesse de convergence ;

(max(w) et max weight) la pénalité la plus élevée ;

(reduction) diminution de la valeur de la fonction objectif ;

(working set) indices des contraintes actives ;

```
        COLLECTED INFORMATION FOR ITERATION STEPS

 K   FSUM(K)    HSUM(K)    LAGRES    DXNORM  KODA KODC   ALPHA  CONV.SPEED  MAX(W)   PREDICTED REDUCTION WORKING SET
 0  0.136E+03  0.450E+01 0.500E+01 0.500E+01   2    1 0.481E+00 0.000E+00 0.100E+00 0.295E+02 0.160E+02   2  6  1
 1  0.116E+03  0.466E+02 0.715E+01 0.684E+01   2    1 0.430E+00 0.180E+01 0.100E+00 0.412E+02 0.266E+02   6  1
 2  0.783E+02  0.661E-01 0.933E+01 0.819E+01   2    1 0.758E+00 0.891E+00 0.100E+00 0.782E+02 0.737E+02   6  7
 3  0.467E+01  0.218E+02 0.277E+01 0.144E+01   2    1 0.100E+01 0.582E+00 0.100E+00 0.589E+01 0.546E+01   7  1
 4  0.955E+00  0.429E+01 0.348E+00 0.466E+00   1    2 0.100E+01 0.418E+00 0.100E+00 0.446E+00 0.441E+00   1
 5  0.938E+00  0.470E-01 0.371E-01 0.667E-01   1    2 0.100E+01 0.106E+00 0.707E+00 0.177E-01 0.177E-01   1
 6  0.953E+00  0.198E-04 0.530E-02 0.792E-02   1    2 0.100E+01 0.349E-01 0.348E+02 0.365E-03 0.365E-03   1
 7  0.954E+00  0.394E-08 0.651E-03 0.997E-03   1    2 0.100E+01 0.104E+00 0.245E+04 0.515E-05 0.515E-05   1
 8  0.954E+00  0.989E-12 0.819E-04 0.125E-03   1    2 0.100E+01 0.125E+00 0.155E+06 0.817E-07 0.816E-07   1
 9  0.954E+00  0.245E-15 0.103E-04 0.157E-04   1    2 0.100E+01 0.126E+00 0.983E+07 0.129E-08 0.129E-08   1
10  0.954E+00  0.602E-19 0.129E-05 0.197E-05   1    2 0.100E+01 0.125E+00 0.627E+09 0.202E-10 0.201E-10   1
11  0.954E+00  0.152E-22 0.160E-06 0.245E-06   1    2 0.100E+01 0.124E+00 0.395E+11 0.321E-12 0.324E-12   1
```

FIGURE 4.1 – Itérations ENLSIP–Fortran sur le Pb65

```
****************************************
*                                      *
*          ENLSIP-JULIA-0.3.0          *
*                                      *
****************************************

Starting point : [-5.0, 5.0, 0.0]

Number of equality constraints   : 0
Number of inequality constraints : 7
Constraints internal scaling     : false

iter    objective    cx_sum   reduction     ||p||   dimA  dimJ2     α     conv. speed   max weight   working set
   0  1.158907e+02  4.50e+00   1.60e+01   5.002e+00   2     1   4.81e-01    0.00e+00     1.00e-01    (1,2,6)
   1  7.831910e+01  4.66e+01   2.66e+01   6.835e+00   2     1   4.30e-01    1.80e+00     1.00e-01    (1,6)
   2  4.665009e+00  6.61e-02   7.37e+01   8.193e+00   2     1   7.58e-01    8.91e-01     1.00e-01    (6,7)
   3  9.547961e-01  2.18e+01   5.46e+00   1.439e+00   2     1   1.00e+00    5.82e-01     1.00e-01    (1,7)
   4  9.376705e-01  4.29e+00   4.41e-01   4.655e-01   1     2   1.00e+00    4.18e-01     1.00e-01    (1)
   5  9.532018e-01  4.70e-02   1.77e-02   6.668e-02   1     2   1.00e+00    1.06e-01     7.07e-01    (1)
   6  9.535243e-01  1.98e-05   3.65e-04   7.921e-03   1     2   1.00e+00    3.49e-02     3.48e+01    (1)
   7  9.535288e-01  3.94e-09   5.15e-06   9.971e-04   1     2   1.00e+00    1.04e-01     2.45e+03    (1)
   8  9.535289e-01  9.88e-13   8.16e-08   1.252e-04   1     2   1.00e+00    1.25e-01     1.55e+05    (1)
   9  9.535289e-01  2.46e-16   1.29e-09   1.572e-05   1     2   1.00e+00    1.26e-01     9.83e+06    (1)
  10  9.535289e-01  6.08e-20   2.03e-11   1.974e-06   1     2   1.00e+00    1.26e-01     6.24e+08    (1)
  11  9.535289e-01  1.54e-23   3.22e-13   2.474e-07   1     2   1.00e+00    1.25e-01     3.93e+10    (1)
```

FIGURE 4.2 – Itérations ENLSIP–Julia sur le Pb65

On observe une concordance exacte des résultats intermédiaires obtenus et du nombre d'itérations effectuées. Cela prouve la bonne retranscription de la méthode, tout du moins sur ce problème.

De futurs tests, réalisés à partir de modèles d'Hydro-Québec, permettront probablement de tirer de nouvelles conclusions à ce sujet.

4.5.3 Comparaison des résultats et performances avec le solveur IPOPT

En plus du problème 65 déjà présenté, deux autres problèmes, le 57 et le 42 de la collection de Hock et Schittkowski [9], ont été utilisés afin de comparer les performances d'ENLSIP en Julia avec IPOPT. Leurs modélisations et informations sont données respectivement en (4.2) et en (4.3).

Commençons par la description du problème 57 :

$$\begin{cases} \min f(x) = \sum_{i=1}^{44} f_i(x)^2, \\ \text{s.c.} \\ 0.49x_2 - x_1x_2 - 0.09 \geq 0, \\ x_1 \geq 0.4,\ x_2 \geq -4. \end{cases} \tag{4.2}$$

Point initial $x_0 = (0.42, 5)^T$;

Solution attendue $x^* = (0.419952675, 1.284845629)^T$;

Fonction objectif à la solution $f(x^*) = 0.02845966972$,

avec $f_i(x) = b_i - x_1 - (0.49 - x_1)\exp(-x_2(a_i - 8))$ pour $i = 1, \ldots, 44$.

Les fonctions f_i peuvent s'interpréter comme la différence entre l'observation b_i et la prédiction du modèle de paramètres x_1 et x_2 réalisée en a_i. Bien que cela reste un problème théorique, sa forme est bien adaptée au cas d'utilisation chez Hydro-Québecde l'algorithme ENLSIP. On peut s'attendre à une convergence vers la solution en un nombre plus faible d'itérations qu'avec IPOPT.

Les valeurs des données sont fournies dans l'ouvrage [9].

On remarque que le point initial est relativement proche de la solution attendue, en particulier la première composante. Ce problème s'est donc avéré être également un bon moyen de tester la robustesse de mon implémentation.

Passons désormais à la description du problème 42 :

$$\begin{cases} \min\ f(x) = (x_1 - 1)^2 + (x_2 - 2)^2 + (x_3 - 3)^2 + (x_4 - 4)^2, \\ \text{s.c.} \\ x_1 - 2 = 0, \\ x_3^2 + x_4^2 - 2 = 0. \end{cases} \tag{4.3}$$

Point initial $x_0 = (1, 1, 1, 1)^T$;

Solution attendue $x^* = (2, 2, 0.6\sqrt{2}, 0.8\sqrt{2})^T$;

Fonction objectif à la solution $f(x^*) = 28 - 10\sqrt{2} \approx 13.8578643$.

Le problème 42 peut sembler plus simple que les deux autres. Par exemple, la première contrainte impose directement la valeur du paramètre à l'optimum x_1. Il présente néanmoins l'intérêt de

n'avoir que des contraintes d'égalité et de partir d'un point non réalisable, ce qui permet d'éprouver la capacité de l'algorithme à gérer ce genre de configurations.

Les différents tests ayant amené aux résultats présentés dans les tableaux 4.1, 4.2 et 4.3 ont été réalisés sur un ordinateur équipé d'un processeur IntelCore i5 de 10ème génération. La version 1.6.0 de Julia a été utilisée et les temps de calcul ont été mesurés à l'aide de la librairie Julia `BenchmarkTools`.

Le temps d'exécution n'a néanmoins pas pu être évalué sur ENLSIP-Fortran77.

Plusieurs constats peuvent être tirés de ces résultats. Tout d'abord, les minima trouvés par les deux algorithmes sont extrêmement proches, voire quasiment identiques, les différences apparaissant à partir de la huitième ou neuvième décimale selon les problèmes. L'objectif de parvenir à trouver des résultats corrects avec l'implémentation en Julia d'ENLSIP semble donc globalement atteint.

En termes de performances, les temps de calcul d'ENLSIP-Julia remplissent largement les exigences fixées au chapitre 1 et sont du même ordre de grandeur que ceux d'IPOPT. Ce dernier point s'explique sûrement par le fait que les problèmes testés sont de très petite dimension et ne permettent donc pas de tirer des conclusions sur les différences de vitesse d'exécution.

Ces premiers résultats se révèlent donc très encourageants pour la suite du développement d'ENLSIP en Julia. Comme dit précédemment, d'autres tests seront réalisés sur des problèmes en plus grande dimension afin de se rapprocher petit à petit des problèmes effectivement résolus par l'outil de prévision dont Hydro-Québec se sert en production.

Solveur	Itérations	Temps de calcul	Objectif
ENLSIP-Julia	5	1.663 sec	2.845966972e-02
IPOPT	24	2.003 sec	2.845966907e-02

Table 4.1 – Comparaison d'ENLSIP-Julia avec IPOPT sur le problème 57

Solveur	Itérations	Temps de calcul	Objectif
ENLSIP-Julia	15	1.493 sec	1.385786438e+01
IPOPT	10	1.584 sec	1.385786437e+01

Table 4.2 – Comparaison d'ENLSIP-Julia et IPOPT sur le problème 42

Solveur	Itérations	Temps de calcul	Objectif
ENLSIP-Fortran77	11	–	0.953529e+01
ENLSIP-Julia	11	1.585 sec	0.953528856e+01
IPOPT	12	2.372 sec	0.953528856e+01

Table 4.3 – Comparaison d'ENLSIP avec IPOPT sur le problème 65

Conclusion

À la fin de mon temps de stage, je suis parvenu à retranscrire l'algorithme ENLSIP en Julia. Ce travail s'est avéré prendre plus de temps que prévu, tant l'analyse du code source écrit en Fortran77 s'est avérée de plus en plus complexe à mesure que mon travail avançait.

Les premiers tests effectués ont principalement contribué à mettre en avant la bonne retranscription en Julia de l'algorithme initialement codé en Fortran77. Cette implémentation bénéficie déjà des avantages de Julia en terme d'utilisation de librairies et de meilleure lisibilité du code. Néanmoins, les gains potentiels en terme de performances n'ont pas pu être évalués spécifiquement avec les tests réalisés sur des problèmes en relativement petite dimension.

La retranscription de l'algorithme n'étant pas complètement terminée, son accessibilité en tant que librairie Julia n'a pas encore été entamée mais fait toujours partie des objectifs finaux du projet.

Les prochains travaux seront dès lors consacrées à la mise en place de problèmes exploitant des données d'Hydro-Québec et faisant appel à des modèles d'évaluation effectivement utilisés en production. Les tests réalisés apporteront de ce fait de nouveaux éléments de réponse quant à la pertinence du passage au langage Julia pour les outils de prévision de la demande d'Hydro-Québec, et ce indépendamment des gains de lisibilité du Julia par rapport au Fortran77, ce qui est déjà un enjeu essentiel en terme de pérennité et de transmission des connaissances. Cette démarche permettra également de mettre l'implémentation réalisée dans des situations pouvant potentiellement mettre en difficulté la méthode d'optimisation utilisée par ENLSIP, contrairement aux cas relativement simples vus précédemment. Cela pourrait potentiellement révéler de nouveaux bogues non détectés jusqu'à présent ou des portions de code mal retranscrites, notamment tout ce qui a trait à la gestion des erreurs. Leur correction pourrait évidemment perfectionner l'algorithme.

Un autre point à aborder concerne à la réalisation de tests sur un échantillon plus important de solveurs de programmation non linéaire. Ainsi, l'on pourra comparer avec plus de pertinence l'efficacité d'ENLSIP face à des algorithmes plus modernes.

Le projet sur lequel j'ai travaillé a aussi pour finalité de fiabiliser et moderniser des outils de prévision de la demande actuellement en production chez Hydro-Québec, ce qui constitue le second grand objectif du projet après celui consistant à retranscrire en Julia ENLSIP-Fortran77 et à valider le nouveau codage sur des problèmes jouets. Ces deux objectifs n'ont pas pu être traités en parallèle puisque la complétion du premier objectif était la condition nécessaire de réalisation du second. En outre, la retranscription en Julia a tout compte fait occupé la quasi-totalité de mon stage, tant l'analyse de l'algorithme codé en Fortran77 s'est révélée plus complexe et longue que prévu. De plus, perfectionner la méthode requiert déjà de mettre en exergue les points devant être améliorés. Ceci ne peut être fait autrement qu'en confrontant ENLSIP à d'autres algorithmes d'optimisations déployant des méthodes plus actuelles. Certains points d'amélioration ont néanmoins pu être discutés au travers de mes échanges avec mon maître de stage.

Un premier aspect pouvant être investigué concerne la gestion des contraintes. L'approche EQP développée dans ENLSIP amène à restreindre le nombre de contraintes traitées et à ne travailler qu'avec des égalités, ce qui se révèle avantageux dans le cadre des moindres carrés. Néanmoins, les avancées récentes dans les approches de type primal-dual, tel que celle présentée par Wächter [17] et adoptée dans IPOPT, permettent de mieux tirer parti de la totalité des contraintes et des multiplicateurs de Lagrange, améliorant grandement la convergence. Cela est d'autant plus le cas lorsque l'on se situe proche de la solution, configuration où ENLSIP peut être amélioré et bénéficier de ce type de travaux. On remarque par exemple sur la figure 4.2 que les cinq dernières itérations ne modifient la fonction objectif qu'à partir de la huitième décimale, ce qui a en réalité peu d'impact sur la solution obtenue.

Un autre aspect traite du calcul de la longueur de pas. La méthode décrite par Lindström et Wedin [11], développée dans les années 1980, présente des caractéristiques pouvant s'apparenter aux méthodes de régions de confiance [3]. L'idée d'approcher la fonction objectif pénalisée afin de calculer des valeurs de pas de plus en plus affinées est en effet commune aux deux approches. Agrémenter le calcul du pas d'une méthode de ce type pourrait donc également s'avérer pertinent en vue de la modernisation d'ENLSIP.

En somme, la modernisation de la méthode d'optimisation d'ENLSIP proprement dite pourra faire l'objet de futurs projets de recherche pour Hydro-Québec et l'UdeM.

Bibliographie

[1] J.-Y. Audibert et O. Catoni : Robust linear least squares regression. *The Annals of Statistics*, 39:2766–2794, 2011.

[2] Jeff Bezanson, Alan Edelman, Stefan Karpinski et Viral B Shah : Julia : A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. URL https://epubs.siam.org/doi/10.1137/141000671.

[3] A.R. Conn, N.I.M. Gould et Ph.L. Toint : *Trust-Region Methods*. MPS-SIAM series on Optimization. Mathematical Programming Society and SIAM, 2000.

[4] J.E. Dennis Jr et R.B. Schnabel : *Numerical Methods for Unconstrained optimization and Nonlinear Equations*. SIAM, 1996.

[5] Philip E. Gill, Walter Murray, Michael A. Saunders et Margaret H. Wright : Model building and practical aspects of nonlinear programming. *In* Klaus Schittkowski, éditeur : *Computational Mathematical Programming*, pages 209–247, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg. ISBN 978-3-642-82450-0.

[6] G.H. Golub et C.F. Van Loan : *Matrix Computations*. Johns Hopkins University Press, quatrième édition, 2013.

[7] R. J. Hanson et Fred T. Krogh : A quadratic-tensor model algorithm for nonlinear least-squares problems with linear constraints. *ACM Trans. Math. Softw.*, 18(2):115–133, jun 1992. ISSN 0098-3500. URL https://doi.org/10.1145/146847.146857.

[8] T. Hastie, R. Tibshirani et J. Friedman : *The Elements of Statistical Learning : Data Mining, Inference and Prediction*. Springer Series in Statistics. Springer, seconde édition, 2009.

[9] W. Hock et K. Schittkowski : *Test Examples for Nonlinear Programming Codes*, volume 187 de *Lecture Notes in Economics and Mathematical Systems*. Springer, seconde édition, 1980.

[10] M.L. Johnson : Nonlinear least-squares fitting methods. *Methods in cell biology*, 84:781–805, 2008.

[11] P. Lindström et P.Å. Wedin : A new linesearch algorithm for nonlinear least squares problems. *Mathematical Programming*, Vol. 29(3):268–296, 1984.

[12] P. Lindström et P.Å. Wedin : Gauss-Newton based algorithms for constrained nonlinear least squares problems. *Technical Report from the Institute of Information procressing, University of Umeå, Sweden*, 1988.

[13] Jorge J. Moré : The Levenberg-Marquardt algorithm : Implementation and theory. *In* G. A. Watson, éditeur : *Numerical Analysis*, pages 105–116, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg. ISBN 978-3-540-35972-2.

[14] J. Nocedal et S.J. Wright : *Numerical optimization*. Springer series in Operation Reasearch and Financial Engineering. Springer, seconde édition, 1999.

[15] D. Orban et A.S. Siqueira : A regularization method for constrained nonlinear least squares. *Computational Optimization and Application*, 76:961–989, 2020.

[16] K. Schittkowski : Solving constrained nonlinear least squares problems by a general purpose SQP-method. *Trends in Mathematical Optimization*, 84:294–309, 1988.

[17] A. Wächter : *An interior point algorithm for large-scale non linear optimization with applications in Process Engineering*. Thèse de doctorat, Carnegie Mellon university, 2002.

[18] N. Yamashita et M. Fukushima : On the rate of convergence of the Levenberg-Marquardt method. *Computing*, (Supplement) 15:237–249, 2001.

[19] Y. Yuan : Recent advances in numerical methods for nonlinear equations and non linear least squares. *Numerical Algebra, Control and Optimization*, 1:15–34, 2011.

Annexes

Définition des composantes du calcul des pénalités

On rappelle :

- $z = \left[(\nabla \hat{c}_i(x_k)^T p_k)^2\right]_{1 \leq i \leq t}$
- $\mu = \left[\frac{|(J_k p_k)^T r(x_k) + \|J_k p_k\|^2|}{\delta} - \|J_k p_k\|^2\right]$ $(\delta = 0.25)$
- t est le nombre de contraintes actives à l'itération k
- ω_1 est la dimension utilisée dans le calcul du premier membre de p_k la direction de descente(voir 3.2)

1. Si $z^T \hat{w}^{(old)} \geq \mu$

   $\tau = 0$
   for $i = 1 : t$ **do**
   $e = \nabla \hat{c}_i(x_k)^T p_k (\nabla \hat{c}_i(x_k)^T p_k + \hat{c}_i(x_k))$
   if $e > 0$ **then**
   $y_i = e$
   else
   $\tau = \tau - e * \hat{w}_i^{(old)}$, $y_i = 0$
   end if
   end for

2. Si $z^T \hat{w}^{(old)} < \mu$ et $\omega 1 \neq t$

   $\tau = \mu$
   for $i = 1 : t$ **do**
   $e = -\nabla \hat{c}_i(x_k)^T p_k * \hat{c}_i(x_k)$
   if $e > 0$ **then**
   $y_i = e$
   else
   $\tau = \tau - e * \hat{w}_i^{(old)}$, $y_i = 0$
   end if
   end for

   Dans les deux cas ci-dessous, la première contrainte du problème 3.24 est $y^T \hat{w} \geq \tau$.

3. Si $z^T \hat{w}^{(old)} < \mu$ et $\omega_1 = t$

   $y = z$ et $\tau = \mu$

   Et la première contrainte du problème 3.24 est $y^T \hat{w} = \tau$.

Algorithm 3 Calcul de longueur de pas avec une méthode de type Armijo-Goldstein

function goldstein_armijo_step
$\alpha = \alpha_0$
while $\phi(\alpha) \geq \phi(0) + \gamma * \alpha * \phi'(0)$ **and** $(\alpha\|p_k\| > \varepsilon_{rel}$ **or** $\alpha > \alpha_{min})$ **do**
  $\alpha = \alpha/2$
end while
return α {*Les constantes* ε_{rel} *et* γ *désignent respectivement la racine carrée de la précision machine sur les nombres flottants et une constante valant* 0.25 *en pratique*}