Titre: Title:	Algorithmes de moindres carrés non linéaires en précision mixte avec applications aux problèmes d'ajustement de faisceaux
Auteur: Author:	Antonin Kenens
Date:	2022
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Kenens, A. (2022). Algorithmes de moindres carrés non linéaires en précision mixte avec applications aux problèmes d'ajustement de faisceaux [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/10779/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/10779/
Directeurs de recherche: Advisors:	Dominique Orban, & Youssef Diouane
Programme: Program:	Maîtrise recherche en mathématiques appliquées

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Algorithmes de moindres carrés non linéaires en précision mixte avec applications aux problèmes d'ajustement de faisceaux

ANTONIN KENENS
Département de mathématiques et de génie industriel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Mathématiques appliquées

Décembre 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Algorithmes de moindres carrés non linéaires en précision mixte avec applications aux problèmes d'ajustement de faisceaux

présenté par **Antonin KENENS**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Jonathan JALBERT, président
Dominique ORBAN, membre et directeur de recherche
Youssef DIOUANE, membre et codirecteur de recherche
Fabian BASTIN, membre

DÉDICACE

À ma famille et mes amis,
je vous aime...

REMERCIEMENTS

Je souhaite remercier les nombreuses personnes qui ont permis à ce mémoire de voir le jour, et ainsi clore cette maîtrise qui représente une part importante de ma vie.

Je remercie les membres du jury, constitué de M. Jonathan Albert, M. Fabian Bastin, M. Youssef Diouane et M. Dominique Orban, pour leurs appréciations à l'égard de mon travail.

Je remercie mes directeurs, Dominique Orban et Youssef Diouane, pour la confiance qu'ils m'ont accordé et leurs conseils justes et avisés tout au long de ma maîtrise.

Je remercie tous mes amis et collègues, ceux qui m'ont soutenu depuis la France, ceux qui m'ont accompagné à Montréal et ceux que j'ai eu la chance de rencontrer ici.

Je remercie enfin ma famille, qui m'a toujours encouragé et soutenu où que j'aille et quoi que je fasse.

RÉSUMÉ

Cette maîtrise explore les différentes méthodes de résolution des moindres carrés non-linéaires présentes dans la littérature, les compare et détermine lesquelles sont les plus adaptées pour la résolution de grands problèmes creux. En particulier, nous proposons des alternatives aux problèmes mal conditionnés.

Pour cela, nous avons commencé par étudier en détail les problèmes de moindres carrés non-linéaires et l'algorithme de Levenberg-Marquardt. Nous nous sommes ensuite concentrés sur la méthode de résolution du sous-problème et en particulier les méthodes de résolution inexacte du sous-problème. Nous avons d'abord étudié les méthodes proposées par Ceres, un logiciel de résolution de moindres carrés non linéaires qui représente l'état de l'art en la matière. Nous avons ensuite proposé une reformulation du problème qui permet l'utilisation de différentes méthodes itératives et préconditionneurs. Nous implémentons ces différentes méthodes de résolution en Julia pour les comparer et intégrer différentes technologies comme la précision mixte, la différentiation automatique, et le calcul sur GPUs.

Nos résultats montrent que la reformulation du sous-problème de moindres carrés linéaires sans former les équations normales permet de réduire d'un facteur important le conditionnement des matrices. De plus, l'utilisation de précision mixte, et GPUs, si elle est faite de manière optimale, permet une accélération notable des calculs. Enfin, la différentiation automatique permet de généraliser notre code pour tous les problèmes dont on ne dispose pas d'un détail de la matrice jacobienne.

ABSTRACT

This master's thesis explores and compares the different existing methods in the litterature to solve large sparse nonlinear least squares problems. In particular, we suggest alternative solutions for ill-conditioned problems.

In order to do that we started off by studying the Levenberg-Marquardt algorithm. We then focused on solving the subproblem and in particular inexact resolution of this subproblem. We first investigated the methods offered by Ceres, a state of the art nonlinear least squares solver. We then proceeded to rework the problem based on the KKT conditions of the linear least squares problem without forming normal equations that allows the use of other iterative methods and preconditioners. We implemend all theses solutions in Julia to compare them and integrate different technologies like mixed precision, automatic differentiation, and GPUs computations.

Our results show that the reformulation of the sub-problem without forming the normal equations allows to reduce the conditioning of the matrices. On top of that, if well used, mixed precision and GPUs computations allow a clear improvement of the results. Finally, automatic differentiation use allows to generalize our code for all the problems for which we don't have a detail of the jacobian matrix.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

f	Fonction objectif
F	Fonction $F : \mathbb{R}^n \to \mathbb{R}^m$ telle que $f(x) = \frac{1}{2}\|F(x)\|_2^2$
J	Matrice jacobienne de F
H	$J^T J$
κ	Symbole déterminant le conditionnement d'une matrice
A^s	Matrice A mise à l'échelle
(PC)	Test effectué sur mon ordinateur personnel
(F22)	Test effectué sur le noeud de calculs frontal22

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

1.1 Contexte

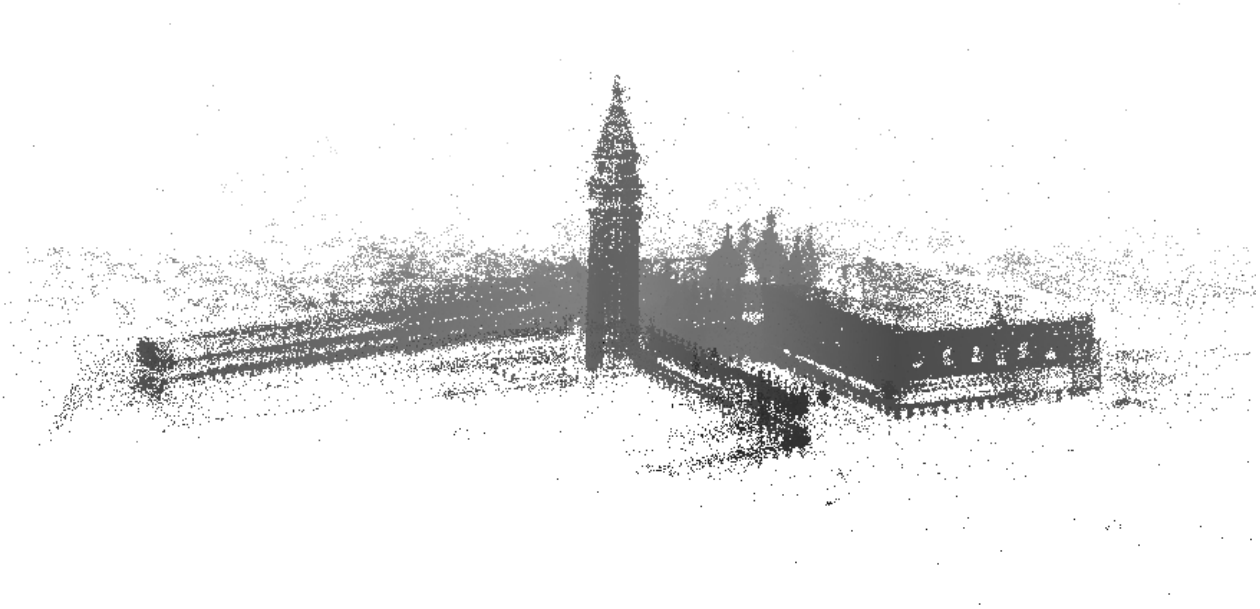

FIGURE 1.1 Modélisation de la place Saint-Marc à Venise pour un problème d'ajustement de faisceaux.

La figure 1.1 est une modélisation en 3 dimensions de la place Saint-Marc à Venise dont les données proviennent de la librairie Bundle Adjustment in the Large (BAL) [1]. Ces problèmes sont tirés d'une expérience visant à reconstruire des bâtiments en 3 dimensions dans un logiciel de modélisation à partir d'une banque de photos publiques. Cela a permis entre autre de reconstruire certaines parties de villes célèbres comme Venise, Rome ou Dubrovnik [2] [3].

La modélisation de la figure 1.1, est générée à partir d'une technique qui s'appelle la structure acquise à partir d'un mouvement ou *structure from motion* qui consiste à reconstruire des structures en 3 dimensions approximativement de la même façon que notre cerveau crée une perspective à partir de nos yeux.

Un exemple schématisé est donné dans la figure 1.2. On souhaite minimiser la distance entre l'observation $y_{1,1}$, $y_{1,2}$, $y_{1,3}$ et la projection du point X_1, sur les caméras C_1, C_2, C_3 afin de pouvoir recréer le cube en 3 dimensions en agglomérant les perspectives des différentes caméras. On fait ensuite la même chose avec tous les points X_2, X_3, ..., X_7.

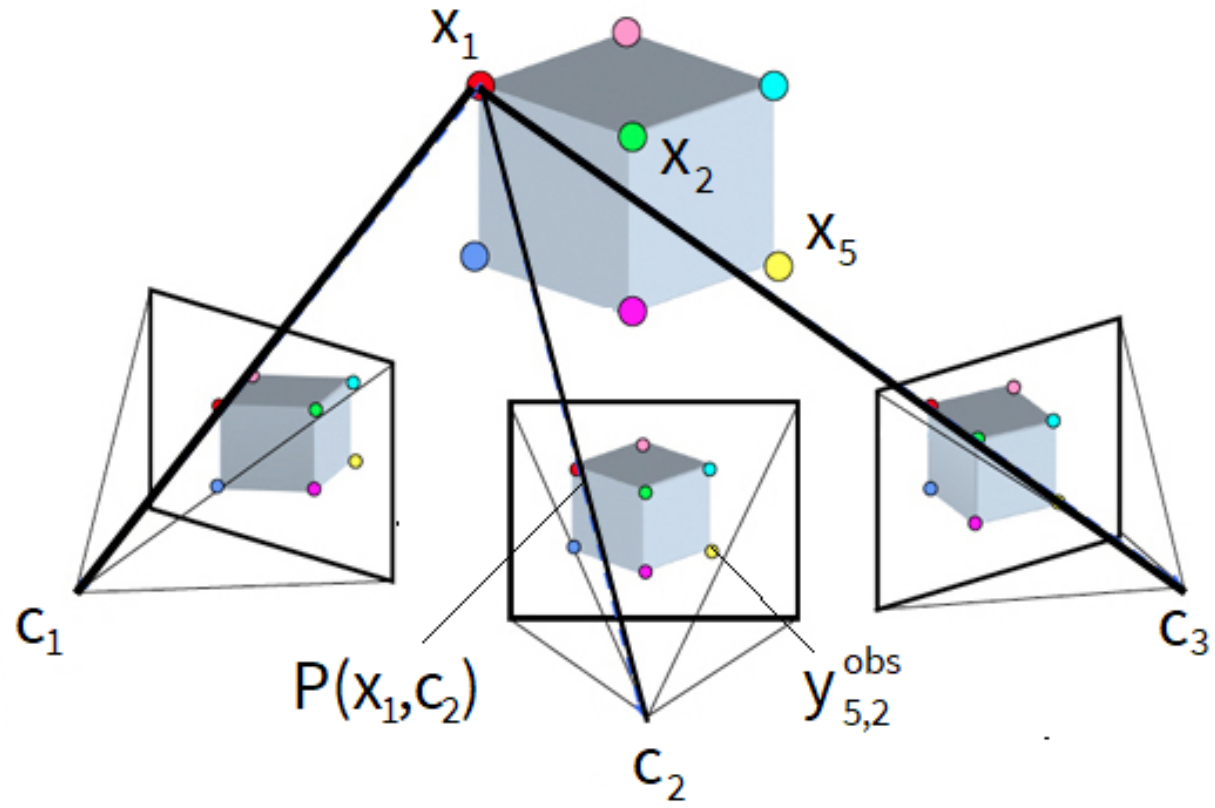

FIGURE 1.2 Exemple de structure acquise à partir d'un mouvement.

On obtient l'équation (1.1)

$$\min_{(X,C)} \sum_{i=1}^{Np} \sum_{j=1}^{Nc} v_{i,j} \| P(X_i, C_j) - y_{i,j}^{obs} \|^2, \tag{1.1}$$

avec

$$\begin{cases} X_i \in \mathbb{R}^3 & \text{correspond aux coordonnées en 3D du ième point,} \\ C_j \in \mathbb{R}^9 & \text{correspond aux 9 paramètres de la jème caméra,} \\ v_{i,j} \in \mathbb{R} & \text{vaut 1 si le point } i \text{ est observé sur la caméra } j \text{ et 0 sinon,} \\ P : \mathbb{R}^{12} \to \mathbb{R}^2 & \text{est la fonction n.l. de projection d'un point sur une caméra,} \\ y_{i,j}^{obs} \in \mathbb{R}^2 & \text{correspond à l'observation en 2D du point } i \text{ sur la caméra } j, \\ Np \in \mathbb{R} & \text{correspond au nombre de points du modèle,} \\ Nc \in \mathbb{R} & \text{correspond au nombre de caméras du modèle.} \end{cases} \tag{1.2}$$

Le problème (1.1), qui consiste à minimiser toutes ces distances pour reconstruire l'objet en 3 dimensions, s'appelle un problème d'ajustement de faisceaux. De manière générale, les problèmes d'ajustement de faisceaux consistent à raffiner une reconstruction visuelle pour produire une structure 3D optimale avec une bonne estimation des paramètres de vision comme la position de la caméra et/ou le calibrage. Le terme "optimal" signifie que les estimations de paramètres sont obtenues en minimisant une fonction de coût (1.1) qui quantifie l'erreur de *fitting* de la fonction [4]. Ces problèmes apparaissent dans plusieurs contextes différents comme la reconstruction 3D d'images satellites [5, 6] ou la modélisation des fonds marins [7].

Dans la figure 1.2, nous avons donc 3 caméras différentes, 7 points différents et 21 observations. En effet, sur chacune des 3 caméras on observe 7 points. Ce qui n'est pas nécessairement toujours le cas, avec un autre angle on pourrait voir l'arrière du cube par exemple. Dans notre cas nous obtenons donc

$$\min_{(X,C)} \sum_{i=1}^{7} \sum_{j=1}^{3} v_{i,j} \| P(X_i, C_j) - y_{i,j}^{obs} \|^2. \tag{1.3}$$

Le problème (1.3) peut être réécrit $\frac{1}{2} \sum_{j=1}^{m} F_j^2(x) = \frac{1}{2} \|F(x)\|_2^2$, c'est à dire qu'il peut être écrit comme un problème de moindres carrés non linéaires. Nous allons donc nous intéresser à la façon dont sont résolus les problèmes de moindres carrés linéaires en prenant comme exemple les problèmes d'ajustement de faisceaux. En effet, il s'avère que les problèmes d'ajustement de faisceaux, et en particulier ceux de la librairie BAL, sont très mal conditionnés. Cela signifie que la solution du problème est très sensible aux perturbations du modèle. Ce conditionnement dépend entre autres de la précision dans laquelle le calcul est effectué.

Nous explorons donc de nouvelles méthodes de résolution qui consistent à améliorer le condi-

tionnement du problème en utilisant par exemple un changement dynamique de précision. Nous explorons aussi l'utilisation de différentes méthodes de résolution ainsi que différentes technologies comme les calculs sur GPU.

Un stage de fin d'études à l'été 2020 consistait en une exploration préliminaire du sujet. La stagiaire a implémenté l'algorithme de Levenberg-Marquardt sur base d'une factorisation symétrique et quasi-définie qui peut être effectuée en n'importe quelle précision.

1.2 Question de recherche

Dans quelle mesure peut-on améliorer la résolution de problèmes d'ajustement de faisceaux en améliorant le conditionnement du système, en utilisant la précision mixte et les calculs sur GPU ?

CHAPITRE 2 REVUE DE LITTÉRATURE

La revue de littérature suivante résume les connaissances nécessaires pour la compréhension du chapitre 3 du mémoire. Les parties 2.1 et 2.2 nous donnent le contexte général dans lequel s'inscrit la recherche. Les parties 2.3 à 2.5 décrivent les différents algorithmes qu'on va utiliser. La partie 2.6 décrit le fonctionnement de Ceres qui est une référence pour la résolution de problèmes aux moindre carrés non linéaires. La partie 2.7 ajoute quelques précisions sur les problèmes d'ajustement de faisceaux. Enfin les parties 2.8 à 2.12 servent à définir les différentes technologies qu'on va utiliser tout au long du mémoire.

2.1 Optimisation numérique

Un problème d'optimisation sans contrainte peut être écrit sous la forme suivante :

$$\min_{x \in \mathbb{R}^n} f(x) \text{ où } f : \mathbb{R}^n \to \mathbb{R}. \tag{2.1}$$

Une variante avec contraintes de bornes s'écrit :

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{sous contraintes } \alpha_i \leq x_i \leq \beta_i,\ i \in \mathcal{I}, \tag{2.2}$$

où $\mathcal{I}$ est l'ensemble d'indices pour les contraintes de bornes et α_i et β_i dans $\bar{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$. Dans notre recherche nous allons nous concentrer principalement sur les problèmes d'optimisation sans contrainte pour lesquels notre fonction objectif f est $\mathcal{C}^2$. C'est-à-dire qu'elle est doublement dérivable et que ses dérivées premières et secondes sont continues. Ceci nous permet de définir le gradient qui est le vecteur des dérivées partielles par rapport à chacune de ses variables :

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}. \tag{2.3}$$

De la même manière, nous pouvons définir la hessienne de f, comme étant la matrice des dérivées secondes de f par rapport à chacune de ses variables :

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}. \tag{2.4}$$

On considère dans la plupart des cas qu'atteindre notre objectif signifie atteindre un minimum local.

Définition 2.1.1. ***Minimum local*** *Un point $x^* \in \mathbb{R}^n$ est un minimum local de* (2.1) *si il y a un voisinage $\mathcal{V}$ de x^* tel que $\forall x \in \mathcal{V},\ f(x^*) \leq f(x)$.*

Afin de vérifier si l'on a atteint notre objectif, nous pouvons nous reposer sur plusieurs critères d'optimalité.

Proposition 2.1.1. ***Condition nécessaire d'optimalité d'ordre 1***
Si $x^\star \in \mathbb{R}^n$ est un minimum local de $f \in \mathcal{C}^1$ alors $\nabla f(x^\star) = 0$.

Proposition 2.1.2. ***Conditions nécessaires d'optimalité d'ordre 2***
Si $x^\star \in \mathbb{R}^n$ est un minimum local de $f \in \mathcal{C}^2$ alors $\nabla f(x^\star) = 0$ et $\nabla^2 f(x^\star)$ est semi-définie positive.

Proposition 2.1.3. ***Conditions suffisantes d'optimalité d'ordre 2***
Soit $x^\star \in \mathbb{R}^n$ tel que $\nabla f(x^\star) = 0$ et $\nabla^2 f(x^\star)$ est définie positive, alors $x^\star$ est un minimum local de $f \in \mathcal{C}^2$.

Dans notre cas, nous cherchons donc à résoudre le problème (2.1) en déterminant un minimum local à partir des conditions nécessaires d'optimalité du premier ordre et ce peu importe le point de départ utilisé. Cette condition supplémentaire sur le point de départ est donnée dans la définition suivante.

Définition 2.1.2. ***Convergence globale.***
Soit une suite x_k de $\mathbb{R}^n$ et une fonction $f : \mathbb{R}^n \to \mathbb{R}$. Si $\lim_{k\to\infty} \|\nabla f(x_k)\| = 0$ peu importe le point de départ $x_0 \in \mathbb{R}^n$, alors la convergence est dite globale.

De plus, sous certaines conditions, le problème converge plus rapidement vers un minimum local. À partir d'un certain seuil tel que défini ci-dessous on parle de convergence quadratique.

Définition 2.1.3. ***Convergence quadratique.***
Soit une suite x_k de $\mathbb{R}^n$ et une fonction $f : \mathbb{R}^n \to \mathbb{R}$. Si $\forall k \in \mathbb{N}$,

$$\frac{\|\nabla f(x_{k+1})\|}{\|\nabla f(x_k)\|^2} \leq c, \tag{2.5}$$

et si $\|f(x_0)\| < \frac{1}{c}$, alors la convergence est dite quadratique.

Ces notions peuvent s'appliquer sur tous les problèmes comme (2.1), mais nous allons les appliquer sur une sous-catégorie qui sont les problèmes aux moindres carrés-non linéaires.

2.2 Problèmes aux moindres carrés non linéaires

Dans le cas des problèmes aux moindres carrés non linéaires, la fonction objectif de notre problème (2.1) est de la forme suivante :

$$f(x) = \frac{1}{2}\sum_{j=1}^{m} F_j^2(x) = \frac{1}{2}\|F(x)\|_2^2, \tag{2.6}$$

où $F : \mathbb{R}^n \to \mathbb{R}^m$ et $F_j : \mathbb{R}^n \to \mathbb{R}$. Ainsi, notre problème d'optimisation générique (2.1) peut être réécrit sous la forme suivante :

$$\min_{x\in\mathbb{R}^n} \frac{1}{2}\|F(x)\|_2^2, \tag{2.7}$$

ce qui nous permet de définir le jacobien $J(x)$ de F, qui est un élément clé dans la résolution de problèmes aux moindres carrés non linéaires. Il s'agit de la matrice $m \times n$ qui est la transposée de la matrice gradient de $F(x)$

$$\forall j \in [1;m],\ F_j \in \mathcal{C}^1,\ J(x) = \left[\frac{\partial F_j}{\partial x_i}\right]_{j=[1;m]\ i=[1;n]} = \begin{bmatrix} \nabla F_1(x)^T \\ \nabla F_2(x)^T \\ \vdots \\ \nabla F_m(x)^T \end{bmatrix}, \tag{2.8}$$

où chaque $\nabla F_j(x),\ j = 1, 2, ..., m$ est le gradient de F_j. Le gradient et la hessienne de f peuvent ensuite être représentés comme suit :

$$\nabla f(x) = \sum_{j=1}^{m} F_j(x)\nabla F_j(x) = J(x)^T F(x), \tag{2.9}$$

et

$$\begin{aligned}\nabla^2 f(x) &= \sum_{j=1}^{m} \nabla F_j(x) \nabla F_j(x)^T + \sum_{j=1}^{m} F_j(x) \nabla^2 F_j(x) \quad \text{si } F_j \in \mathcal{C}^2 \\ &= J(x)^T J(x) + \sum_{j=1}^{m} F_j(x) \nabla^2 F_j(x). \end{aligned} \tag{2.10}$$

Afin de résoudre le problème d'optimisation, on peut utiliser la méthode de Gauss-Newton, qui peut être considéré comme une modification d'une méthode de Newton. La méthode de Newton est une méthode itérative qui consiste à trouver un minimum local d'une fonction en choisissant à chaque itération k une direction de descente d^N basée sur le développement de Taylor d'ordre 2 appliqué appliqué de la fonction objectif au point x_k. Le système qui en résulte pour obtenir la direction de descente est le suivant :

$$\nabla^2 f(x_k) d^N = -\nabla f(x_k). \tag{2.11}$$

Dans le cas de la méthode Gauss-Newton, au lieu de résoudre le système (2.11), on souhaite résoudre le système suivant :

$$J(x_k)^T J(x_k) d^{GN} = -J(x_k)^T F(x_k). \tag{2.12}$$

Ce système 2.12 nommé les équations normales est équivalent à résoudre le problème :

$$\min_{d \in \mathbb{R}^n} \tfrac{1}{2} \| J(x_k) d + F(x_k) \|_2^2, \tag{2.13}$$

à chaque itération. Le système (2.12) découle des équations (2.9) et (2.10), où l'on considère que l'on peut procéder à l'approximation suivante :

$$\nabla^2 f(x_k) \approx J(x_k)^T J(x_k). \tag{2.14}$$

Un défaut posé par la méthode de Gauss-Newton est que la convergence globale telle que donnée dans la définition 2.1.2 nécessite la condition suivante sur la matrice jacobienne $J(x_k)$:

Proposition 2.2.1. *L'algorithme de Gauss-Newton a une convergence globale telle que*

définie dans 2.1.2 si il existe $\gamma > 0$ tel que

$$\|J(x)z\| \geq \gamma\|z\|, \tag{2.15}$$

$\forall x$ dans un voisinage $\mathcal{V}$ de $\{x \mid f(x) \leq f(x_0)\}$ où x_0 est le point de départ de l'algorithme.

Un algorithme plus robuste qui n'a pas besoin de cette condition sur la jacobienne est l'algorithme de Levenberg-Marquardt.

2.3 Algorithme de Levenberg-Marquardt

Cet algorithme [8,9] a été conçu pour résoudre les problèmes de moindres carrés non linéaires. L'élément central de cet algorithme est de résoudre une version régularisée du système de Gauss-Newton (2.12) :

$$(J(x_k)^T J(x_k) + \lambda^2 I)d_k = -J(x_k)^T F(x_k), \tag{2.16}$$

où on va noter

$$H_\lambda(x_k) = J(x_k)^T J(x_k) + \lambda^2 I. \tag{2.17}$$

Il s'est imposé petit à petit comme une référence pour les problèmes de moindres carrés non-linéaires par rapport aux méthodes de Gauss-Newton ou aux autres variantes grâce à ses meilleures performances sur de nombreux problèmes réels [10]. La façon standard d'aborder ce problème est de résoudre à chaque itération k le problème de moindre carrés linéaires

$$\min_{d\in\mathbb{R}^n} \tfrac{1}{2}\|J(x_k)d + F(x_k)\|_2^2 + \tfrac{1}{2}\lambda_k^2\|d_k\|_2^2, \tag{2.18}$$

qui, si on le résout de manière exacte, est équivalent à trouver la solution du système (2.16). Le second terme λ_k est introduit pour assurer la convergence l'algorithme. En effet, on peut réécrire le problème (2.18) sous la forme suivante :

$$\min_{d\in\mathbb{R}^n} \tfrac{1}{2}\left\| \begin{bmatrix} J(x_k) \\ \lambda_k I \end{bmatrix} d + \begin{bmatrix} F(x_k) \\ 0 \end{bmatrix} \right\|^2, \tag{2.19}$$

où on va noter

$$J_\lambda(x_k) = \begin{bmatrix} J(x_k) \\ \lambda_k I \end{bmatrix}. \tag{2.20}$$

Si $\lambda_k > 0$, alors $J_\lambda(x_k)$ est toujours de rang maximal , ce qui assure ainsi la convergence globale de l'algorithme. La version classique de l'algorithme par régularisation est donnée dans Algorithme 2.3.1.

Algorithme 2.3.1 Version classique de Levenberg-Marquardt

1: On choisit les constantes $0 < \eta_1 \leq \eta_2 < 1$, $0 < \sigma_2 < 1 < \sigma_1$, $\epsilon_a^g > 0$ et $\epsilon_r^g > 0$
2: On choisit $x_0 \in \mathbb{R}^n$ et $\lambda_0 > 0$
3: On évalue $F(x_0)$ et $J(x_0)$
4: On calcule $\|F(x_0)\|$ et $\|J(x_0)^T F(x_0)\|$
5: **pour** k = 0,1,... **faire**
6: **si** $\|J(x_k)^T F(x_k)\| \leq \epsilon_a^g + \epsilon_r^g \|J(x_0)^T F(x_0)\|$ **alors**
7: On sort de l'algorithme
8: **fin si**
9: On calcule le pas tel que

$$d_k \approx \arg\min_{d \in \mathbb{R}^n} \tfrac{1}{2} \left\| \begin{bmatrix} J(x_k) \\ \lambda_k I \end{bmatrix} d + \begin{bmatrix} F(x_k) \\ 0 \end{bmatrix} \right\|^2 \text{ (Approximativement)}$$

10: On calcule le ratio

$$\rho_k = \frac{\frac{1}{2}(\|F(x_k)\|^2 - \|F(x_k + d_k)\|^2)}{\frac{1}{2}(\|F(x_k)\|^2 - \|J(x_k)d_k + F(x_k)\|^2 - \lambda_k^2 \|d_k\|^2)}$$

11: **si** $\rho_k < \eta_1$ **alors**
12: On fixe $x_{k+1} = x_k$
13: $\lambda_k = \max(\lambda_0, \sigma_1 \lambda_k)$
14: **sinon**
15: On fixe $x_{k+1} = x_k + d_k$
16: On évalue $F(x_{k+1})$ et $J(x_{k+1})$
17: **si** $\rho_k \geq \eta_2$ **alors**
18: $\lambda_k = \sigma_2 \lambda_k$
19: **fin si**
20: **fin si**
21: **fin pour**

Les sous-parties 2.3.1 à 2.3.4 décrivent ligne par ligne l'algorithme de Levenberg-Marquardt à l'exception de la résolution du sous problème à la ligne 9 de l'algorithme pour laquelle nous avons ajouté 2 parties distinctes 2.4 et 2.5 étant donné leur importance dans ce mémoire. Les lignes 1 à 4 définissent les constantes et variables nécessaires pour le fonctionnement

de l'algorithme. La ligne 5 est une boucle qui termine lorsque les conditions d'arrêts de l'algorithme à la ligne 6 sont atteintes.

2.3.1 Conditions d'arrêt

Les critères d'arrêt de Levenberg-Marquardt considérés dans notre implémentation pour atteindre sont tirés de [11, 12]. Elles permettent de correspondre aux notions de convergence de l'algorithme de Levenberg-Marquardt décrites dans la sous-partie 2.3.4.

$$f(x_k) \leq \epsilon^f, \tag{2.21}$$

$$\|J(x_k)^T F(x_k)\| \leq \epsilon^g. \tag{2.22}$$

Le premier critère (2.21), vérifie que la valeur de l'objectif a suffisamment diminué et le second (2.22) vérifie que la norme du gradient a suffisamment diminué et donc qu'on atteint les conditions nécessaires d'optimalité du premier ordre telles que définies dans la Proposition 2.1.1. En pratique, on va relâcher ces tolérances strictes en utilisant plutôt les suivantes :

$$f(x_k) \leq \epsilon_a^f + \epsilon_r^f \times f(x_0), \tag{2.23}$$

et

$$\|J(x_k)^T F(x_k)\| \leq \epsilon_a^g + \epsilon_r^g \times \|J(x_0)^T F(x_0)\|. \tag{2.24}$$

Les valeurs ϵ_a^f et ϵ_a^g définissent une tolérance absolu à atteindre et les valeurs ϵ_r^f et ϵ_r^g définissent une tolérance relative aux valeurs initiales du problèmes. Ces critères relâchés sont utiles pour les problèmes dont la valeur initiale de l'objectif ou du gradient est très importante, ce qui est le cas pour les problèmes d'ajustement de faisceaux. Comme on pourra le constater dans la partie 2.7, en pratique il est aussi presque impossible d'atteindre $f(x_k) = 0$ pour les problèmes d'ajustement de faisceaux. C'est donc le critère (2.24) qui représente notre principale condition d'arrêt.

Une fois que l'on a vérifié si l'on atteint où non les conditions d'arrêts, on calcule le pas de notre direction de descente correspondant à la ligne 9.

2.3.2 Résolution du sous-problème

Comme mentionné plus haut, le pas d_k calculé à chaque itération de l'algorithme de Levenberg-Marquardt est obtenu en résolvant le système (2.16) qui est une version régularisée du système de Gauss-Newton (2.12). Ce sous-problème peut être résolu de plusieurs façons. Soit on l'écrit comme un système linéaire que l'on résout de manière exacte à l'aide de méthodes directes qui consistent à factoriser le membre de gauche du système linéaire. Soit on l'écrit comme un problème de moindres carrés linéaires qu'on peut résoudre de manière inexacte avec des méthodes itératives comme les méthodes de Krylov. Dans la sous-partie 2.3.4, nous observons que l'on peut tout de même obtenir une convergence globale de l'algorithme même en résolvant le sous-problème de manière inexacte. Comme les méthodes directes et itératives constituent une part importante de ce mémoire, elles sont abordées plus en détail dans les parties 2.4 et 2.5. Une fois que l'on a résolu notre sous-problème qui correspond à la ligne 9 de Algorithme 2.3.1 et qu'on a calculé le pas d_k, nous en vérifions la qualité.

2.3.3 Qualification d'une itération

À la ligne 10 de Algorithme 2.3.1, on qualifie la qualité d'une étape. Dans la version inexacte de [11] on choisit

$$\rho_k = \frac{\frac{1}{2}(\|F(x_k)\|^2 - \|F(x_k + d_k)\|^2)}{\frac{1}{2}(\|F(x_k)\|^2 - \|J(x_k)d_k + F(x_k)\|^2 - \lambda_k^2\|d_k\|^2)}. \tag{2.25}$$

Le numérateur est

$$Ared = \tfrac{1}{2}\|F(x_k)\|^2 - \tfrac{1}{2}\|F(x_k + d_k)\|^2, \tag{2.26}$$

et il correspond à $f(x_k) - f(x_k + d_k)$, ce qui correspond à réduction effective (ou Actual REDuction) du modèle avec le pas d_k. Plus cette valeur est importante plus le pas permet une diminution importante de la valeur de l'objectif.

Le dénominateur est

$$Pred = \tfrac{1}{2}\|F(x_k)\|^2 - \tfrac{1}{2}(\|J(x_k)d_k + F(x_k)\|^2 + \lambda^2\|d_k\|^2), \tag{2.27}$$

et il correspond à la réduction prédite (ou Predicted REDuction) de la fonction objectif avec le pas obtenu par la résolution du sous-problème (2.18).

La fraction de ces deux valeurs $Ared$ et $Pred$ permet donc de constater si la réduction du modèle obtenue avec un pas d_k correspond à celle prédite par le sous-problème (2.18). La valeur de ρ_k varie donc entre 0 et 1. Plus ρ_k est proche de 0, plus cela signifie que le pas d_k

obtenu par résolution du sous-problème ne permet pas une réduction de la fonction objectif. Dans l'autre sens, plus ρ_k est proche de 1, plus cela signifie que le pas d_k obtenu permet une bonne réduction de la fonction objectif.

A partir de cette valeur de ρ_k, on détermine comment changer notre modèle du sous-problème pour obtenir un meilleur pas à l'itération suivante. Si ρ_k est proche de 0, on augmente la valeur de λ_k, ce qui signifie augmenter la régularisation du modèle car l'étape est mauvaise et on ne met pas à jour le pas. Cette explication correspond aux lignes 11 à 13 de Algorithme 2.3.1.

Si la valeur de ρ_k est suffisamment supérieure à 0, on met à jour le pas et on évalue $F(x_{k+1})$ et $J(x_{k+1})$ ce qui correspond aux lignes 14 à 16 de Algorithme 2.3.1.

Si la valeur de ρ_k est très proche de 1, on diminue la valeur de λ_k car cela signifie que le pas obtenu est très bon et qu'on peut donc réduire la régularisation. Ceci correspond aux lignes 17 et 18 de Algorithme 2.3.1.

En suivant ce modèle d'algorithme, on peut obtenir plusieurs résultats sur la convergence de l'algorithme.

2.3.4 Convergence de l'algorithme de Levenberg-Marquardt

Le modèle d'algorithme que nous utilisons en pratique est tiré de l'article de Wright et Holt [11]. D'après cet article nous pouvons déterminer la convergence globale et la convergence quadratique locale de l'algorithme de Levenberg-Marquardt.

Hypothèse 2.3.1. *Hypothèses sur les fonctions. $f \in \mathcal{C}^2$ dans l'ensemble*

$$S = \{x \in \mathbb{R}^n | f(x) \leq f(x_0)\}, \tag{2.28}$$

et la jacobienne satisfait l'inégalité

$$\forall x \in S,\ M > 0,\ \|J(x)\| \leq M, \tag{2.29}$$

alors la hessienne est aussi bornée sur S.

Hypothèse 2.3.2. *Résolution inexacte du sous-problème. Si on résout le sous-problème de la ligne 9 de Algorithme 2.3.1 de manière inexacte avec une méthode itérative alors on choisit comme critère d'arrêt*

$$\|(J(x_k)^T J(x_k) + \lambda^2 I)d_k + J(x_k)^T F(x_k)\| \leq \mu_k \|J(x_k)^T F(x_k)\|, \tag{2.30}$$

avec $\mu_k \leq \mu_0 < 1$.

Proposition 2.3.1. *Convergence globale de l'algorithme de Levenberg-Marquardt avec une méthode itérative d'après le théorème 5 de [11]. Si on respecte l'hypothèse 2.3.1, qu'on résout le sous-problème de la ligne 9 de Algorithme 2.3.1 de manière inexacte avec une méthode itérative en respectant l'hypothèse 2.3.2 et si pour toute suite* μ_k*, on peut choisir une suite bornée* λ_k *tel que*

$$\rho_k \geq \eta_1 > 0,$$

où ρ_k *est tel que défini dans* (2.25)*, alors l'algorithme converge globalement.*

Proposition 2.3.2. *Convergence quadratique locale de Levenberg-Marquardt d'après le théorème 11 de [11]. Si* x_k *tend vers* x^* *avec* $F(x^*) = 0$*, que la hessienne de la fonction objectif* f *en* x^* *est définie positive et que la suite* μ_k *satisfait*

$$\mu_k \leq \|J(x_k)^T F(x_k)\|,$$

et ce $\forall k$ *tel que* $k \geq K_1, K_2$ *comme définis dans le théorème 11 de [11], alors la suite* x_k *a une convergence quadratique.*

Des résultats similaires sont obtenus en résolvant le sous-problème de manière exacte dans [13]. Comme nous l'avons vu jusqu'à présent, il existe deux principales façons de résoudre le sous-problème, soit avec les méthodes directes, soit avec des méthodes itératives, nous allons donc détailler comment il peut être résolu avec les méthodes directes.

2.4 Méthodes directes

On a vu dans la partie 2.3.2 que pour résoudre des problèmes aux moindres carrés non linéaires, on résout le système (2.16). Il existe plusieurs méthodes qui consistent à factoriser la matrice de notre système linéaire afin de découper le système en plusieurs parties plus simples à résoudre. La factorisation QR, par exemple consiste ici à factoriser la matrice J en une matrice $\hat{Q}$ dont les colonnes sont orthogonales et $\hat{R}$ une matrice triangulaire supérieure

$$J = \hat{Q}\hat{R}, \tag{2.31}$$

avec $J \in \mathbb{R}^{m\times n}$, $\hat{Q} \in \mathbb{R}^{m\times n}$ et $\hat{R} \in \mathbb{R}^{n\times n}$. Cette factorisation où $\hat{Q} \in \mathbb{R}^{m\times n}$ et $\hat{R} \in \mathbb{R}^{n\times n}$ est appelée factorisation QR réduite avec les colonnes de $\hat{Q}$ 2 à 2 orthogonales. La factorisation

QR complète consiste à ajouter $m-n$ vecteurs orthogonaux à $\hat{Q}$ afin qu'on ait $Q \in \mathbb{R}^{m\times m}$ une matrice orthogonale et $R \in \mathbb{R}^{m\times n}$ une matrice triangulaire supérieure avec les $m-n$ dernières lignes nulles. On calcule ensuite un projecteur orthogonal $P = \hat{Q}\hat{Q}^T$ pour obtenir y qui est la projection orthogonale de b sur $Im(J)$.

$$y = Pb = \hat{Q}\hat{Q}^T b. \tag{2.32}$$

Comme $y \in Im(J)$, le système $Jx = y$ a une solution exacte. En combinant (2.31) et (2.32) on obtient

$$\hat{Q}\hat{R}x = \hat{Q}\hat{Q}^T b. \tag{2.33}$$

En multipliant par $\hat{Q}^T$ à gauche on obtient

$$\hat{R}x = \hat{Q}^T b. \tag{2.34}$$

La matrice R (2.34) est triangulaire supérieure, inversible si J est de rang maximal et peut être résolue par remontée triangulaire.

Deux résultats sont très importants dans ce contexte :

Théorème 2.4.1. *Toute matrice $J \in \mathbb{R}^{m\times n},\ (m \geq n)$ a une factorisation QR complète et aussi une factorisation réduite.*

Théorème 2.4.2. *Toute matrice $J \in \mathbb{R}^{m\times n},\ (m \geq n)$ de rang plein a une unique factorisation QR réduite $J = QR$ avec $r_{jj} > 0,\ \forall j = 1 : n$.*

La factorisation QR d'une matrice $J \in \mathbb{R}^{m\times n}$ peut être effectuée de différentes façons. Celle qui était utilisée en premier historiquement est l'orthogonalisation de Gram-Schmidt, qui a ensuite été modifiée afin d'en assurer la stabilité numérique. Cette version a une complexité algorithmique de $\sim 2mn^2$. Une autre factorisation qui est plus utilisée est la triangularisation de Householder [14]. D'une part elle a une complexité algorithmique moindre $\sim 2mn^2 - \frac{2}{3}n^3$ et d'autre part elle permet de ne pas former explicitement la matrice Q mais seulement certains vecteurs qui permettent de former les réflexions de Householder. La totalité de la factorisation peut alors être contenue dans la matrice J de départ. La dernière méthode de factorisation utilisée est la factorisation par rotations de Givens [15]. Il s'agit de mettre à zéro les éléments subdiagonaux d'une matrice $J \in \mathbb{R}^{m\times n}$ en lui appliquant des matrices de rotation de la forme suivante.

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}. \tag{2.35}$$

Ce type de factorisation a un intérêt particulier dans le cas de matrices creuses, sinon les autres méthodes mentionnés sont en principe plus performantes. Si le système est petit et que l'on a formé la matrice H_λ comme définie dans l'équation (2.17), alors on peut le résoudre de manière exacte avec des méthodes directes. Une autre façon de résoudre ce problème est d'utiliser la factorisation LDL^T [16]. En suivant un raisonnement relativement similaire à celui utilisé pour la factorisation QR, on peut résoudre le système 2.16. Notons $H_\lambda = J^T J + \lambda I$, on peut factoriser la matrice H_λ.

Théorème 2.4.3. *Si $H_\lambda \in \mathbb{R}^{n\times n}$ est une matrice symétrique indéfinie, alors il existe une matrice triangulaire inférieure L et une matrice diagonale D telle que*

$$H_\lambda = LDL^T. \tag{2.36}$$

Dans notre cas, on a alors

$$H_\lambda d = -J^T F. \tag{2.37}$$

En effectuant la factorisation de H_λ, on obtient

$$LDL^T d = -J^T F. \tag{2.38}$$

Notons $DL^T d = d_1$, on peut résoudre le système 2.39 avec une élimination de Gauss-Jordan

$$Ld_1 = -J^T F, \tag{2.39}$$

puis on peut inverser D et effectuer une élimination de Gauss-Jordan pour obtenir le résultat

$$d = L^{-T} D^{-1} L^{-1} J^T F. \tag{2.40}$$

Cependant, cette méthode de résolution exacte pose 2 problèmes majeurs. D'une part, elle nécessite de former la matrice $H_\lambda = J^T J + \lambda^2 I$ et d'autre part, elle est sous-optimale lorsque la matrice qu'on factorise est creuse car elle effectue de nombreuses opérations inutiles.

2.5 Méthodes itératives

Afin de résoudre des problèmes de moindres carrés linéaires dont la matrice est grande et creuse il est possible d'utiliser des méthodes itératives dont les performances dans ce type de situations sont généralement meilleures que les méthodes directes comme dans la partie précédente 2.4. En particulier, nous nous intéressons au méthodes de Krylov que nous allons détailler ci-dessous.

2.5.1 Construction et fonctionnement des méthodes de Krylov

Soit $A \in \mathbb{R}^{n\times n}$, on écrit son polynôme caractéristique

$$p(X) = \det(XI_n - A) = X^n + p_{n-1}X^{n-1} + \cdots + p_1X + p_0, \tag{2.41}$$

alors d'après le théorème de Cayley-Hamilton, on a

$$p(A) = A^n + p_{n-1}A^{n-1} + \cdots + p_1A + p_0I_n = 0_n. \tag{2.42}$$

Si A est inversible, $p_0 \neq 0$ et

$$A^{-1} = -\frac{1}{p_0}(A^{n-1} + p_{n-1}A^{n-2} + \cdots + p_1I_n). \tag{2.43}$$

On a alors

$$x^\star = A^{-1}b \implies x^\star \in \mathcal{K}_n(A, b), \tag{2.44}$$

où $x^\star$ est la solution de $Ax = b$ et $\mathcal{K}_n(A, b)$ est un espace de Krylov généré par les vecteurs $\{b, Ab, \ldots, A^{n-1}b\}$.

Le but des méthodes de Krylov est de créer itérativement une solution $x_k \in \mathcal{K}_n(A, b)$ du système linéaire $Ax = b$. On utilise différents processus pour générer une base V_k de $\mathcal{K}_k(A, b)$. Ensuite, la projection de A dans le sous-espace de Krylov a une structure particulière, qui permet de générer de manière itérative un vecteur $y_k \in \mathbb{R}^k$ et ensuite $x_k = V_ky_k$.

D'après l'équation (2.43), on pourrait penser que pour résoudre le système $Ax = b$ à l'aide des méthodes de Krylov, on a besoin de n étapes pour obtenir A^{-1} et donc que les méthodes directes restent plus intéressantes mais selon certaines conditions le nombre d'étapes requis est bien moins grand comme on peut le voir ci-dessous.

En effet, on sait qu'un polynôme minimal q de A est un polynôme de degré m tel que $q(A) = 0$, et il divise tous les polynômes tels que $r(A) = 0$. Il divise notamment le polynôme caractéristique de A et il a les mêmes racines :

$$A^{-1} = -\frac{1}{q_0}(A^{m-1} + q_{m-1}A^{m-2} + \cdots + q_1 I_n), \tag{2.45}$$

et donc

$$x^\star = A^{-1}b \implies x^\star \in \mathcal{K}_m(A, b). \tag{2.46}$$

pour qu'on ait $m \ll n$, alors le nombre de racines distinctes de $q(A)$ doit être petit. Cela signifie que A doit avoir un faible nombre de valeurs propres distinctes.

Les différents processus de Krylov sont à la base des méthodes de Krylov. Selon l'espace de Krylov généré, les méthodes de Krylov sont plus ou moins spécialisées dans un ensemble de problèmes linéaires. On détaille en dessous les processus et méthodes qui vont être utilisées pour la résolution de problèmes de moindres carrés non linéaires.

Dans le cadre de cette maîtrise, il y a 3 types de systèmes qui nous intéressent : les systèmes linéaires symétriques, symétriques indéfinis, et les problèmes de moindres carrés linéaires. Les processus associés sont le processus de Lanczos symétrique [17], le processus de Golub-Kahan [18] et Saunders-Simon-Yip [19]. Pour les systèmes linéaires symétriques définis positifs, la méthode la plus couramment utilisée est le gradient conjugué (CG) [20]. Pour les systèmes symétriques il s'agit de MINRES [21], TriCG et TriMR [22]. Pour les problèmes de moindres carrés linéaires il s'agit de LSQR et LSMR [23, 24].

2.5.2 Choix des méthodes de Krylov

LSQR

D'après la forme du problème à la ligne 9 de Algorithme 2.3.1, la première idée serait d'utiliser une méthode qui résout les problèmes de moindres carrés linéaires. L'algorithme LSQR [23] semble être le plus évident afin de minimiser la norme du résidu. C'est une méthode basée sur le processus de bidiagonalisation de Golub-Kahan [18]. LSQR appliqué sur un système linéaire régularisé comme à la ligne 9 de Algorithme 2.3.1 est équivalent à appliquer la méthode du gradient conjugué aux équations normales $(J^T J + \lambda^2 I)d = -J^T F$. La norme du résidu $\|r_k\|$ décroît de manière monotone et on peut accéder à une estimation des valeurs $\|r_k\|$, $\|J^T r_k\|$, $\|x_k\|$, $\|J\|$ et au conditionnement $\kappa(J)$ à moindre coût.

LSMR

Cependant, dans le sous problème 9 de Algorithme 2.3.1 notre priorité est surtout d'atteindre des conditions d'optimalité du premier ordre, autrement dit de minimiser la norme du gradient. Une autre méthode nommée LSMR semblerait alors plus appropriée. En effet, elle est équivalente à MINRES [21] appliquée aux équations normales de sorte que $\|A^T r_k\|$ décroisse de manière monotone. De plus, on peut aussi prouver que $\|r_k\|$ décroît de manière monotone et que sa valeur est assez proche de celle obtenue par LSQR. Enfin, comme pour LSQR, on peut aussi accéder à une estimation des valeurs $\|r_k\|$, $\|J^T r_k\|$, $\|x_k\|$, $\|J\|$ et au conditionnement $\kappa(J)$ à moindre coût.

MINRES

La méthode MINRES qu'on a nommé dans la partie précédente s'applique sur les problèmes carrés symétriques et se base sur le processus de Lanczos symétrique. $\|r_k\|$ y décroît de manière monotone et cette méthode produit à chaque itération une valeur de $\|A^T r_k\|$ optimale. On peut aussi évaluer $\|r_k\|$ à moindre coût.

Comparaison de méthodes

Ces trois méthodes ont une complexité relative aux nombre d'itérations effectuées. Elles sont regroupées dans le tableau 2.1 tiré de l'article original sur LSMR [24].

Dans ce tableau 2.1 on voit la taille de mémoire nécessaire à chaque itération pour résoudre un système $Ax = b$ avec $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$ et $b \in \mathbb{R}^m$. Cependant cette mémoire peut être écrasée d'une itération à l'autre. Les colonnes *Calcul* correspondent à la taille des produits matrices-vecteur. La notion de taille signifie que si on exécute notre code en précision double pour LSMR par exemple, il faut 2 vecteurs de Float64 de taille m en mémoire et 4 vecteur de taille n. On constate aussi que la complexité de MINRES et ses performances sont semblables à celles de LSMR et LSQR. On peut alors se demander pourquoi ne pas utiliser directement cette méthode plutôt que LSMR. En effet MINRES ne requiert aussi que des opérations

Tableau 2.1 Complexité algorithmique de LSMR, LSQR et MINRES.

	Mémoire		Calcul	
	m	n	m	n
LSMR	2	4	3	6
LSQR	2	3	3	5
MINRES sur $A^T Ax = A^T b$	1	6		8

matrice-vecteur mais ne s'applique que sur des matrices carrés symétriques. Or ce n'est pas réaliste en pratique d'utiliser MINRES sur les équations normales car cela réduirait fortement la stabilité du système. Dans la partie 3 nous abordons une manière de reformuler le problème sous forme de matrice carrée pour y appliquer la méthode MINRES [21]. Cette reformulation du problème produit une matrice carrée $A \in \mathbb{R}^{(m+n)\times(m+n)}$ donc le tableau 2.1 ne s'applique pas de la même façon. Sa complexité dépend aussi du nombre d'itérations effectuées. Par exemple pour une matrice carrée de taille m au bout de k itérations, elle a une complexité en mémoire de $(k+3+1/k)m+nz$ et en calcul de $(k+2)m$ où nz représente les éléments non nuls de la matrice.

TriCG et TriMR

Les méthodes TriCG et TriMR sont rapidement abordées dans la suite du mémoire et elles se basent sur le processus de Saunders-Simon-Yip [19]. Ce sont des méthodes dont les performances sont optimales pour une matrice de la forme suivante :

$$\begin{bmatrix} I & J \\ J^T & -I \end{bmatrix} \tag{2.47}$$

C'est cette forme de matrice (2.47) que nous allons retrouver plus tard dans le mémoire, mais avec un second membre qui n'est pas de la bonne forme pour ces méthodes c'est pourquoi nous apporterons quelques modifications pour les faire fonctionner.

Une fois que nous avons décrit l'aspect théorique de la résolution des moindres carrés non linéaires, nous pouvons désormais nous concentrer sur la résolution pratique de ces problèmes en commençant par une librairie Ceres qui est notre référence sur l'état de l'art des librairies qui résolvent les moindres carrés non linéaires.

2.6 Ceres

Ceres Solver est une librairie open source en C++ servant à modéliser et résoudre de larges et difficiles problèmes d'optimisation. Elle peut être utilisé pour résoudre des problèmes de moindres carrés non linéaires avec ou sans contraintes de bornes [25].

Cette librairie propose principalement d'utiliser l'algorithme de Levenberg-Marquardt pour résoudre des problèmes de moindres carrés non linéaires. Les méthodes disponibles pour la résolution du sous-problème sont la factorisation QR de la matrice J_λ, la factorisation LDL^T de la matrice H_λ, l'algorithme CG sur H_λ, ou le complément de Schur sur H_λ.

Même si les objectifs de Ceres semblent être assez similaires à ceux de notre projet, nous nous démarquons de différentes façons. En particulier grâce au fait de coder la librairie en Julia ce qui permet d'accéder à différentes technologies comme le *multiple dispatch* pour un support du code en précision mixte mais qui permet aussi un support plus simple du code sur GPU.

Un grand intérêt de la librairie Ceres est qu'elle est très largement utilisée en industrie [26] et donc qu'elle nous donne un bon repère sur les directions qui ont été peu ou pas explorées pour la résolution de problèmes aux moindres carrés non linéaires.

Historiquement la librairie MINPACK [27] en FORTRAN est aussi une référence dans le domaine et plus récemment la librairie SBA [28] pour les problèmes d'ajustement de faisceaux dont nous allons approfondir la description dans la partie suivante.

2.7 Détails sur les problèmes d'ajustement de faisceaux

Dans l'introduction, nous avons donné une première description des problèmes d'ajustement de faisceaux. Certains éléments pratiques sont expliqués plus en détail dans la partie ci-dessous.

Notre vecteur résidu, obtenu à partir de la fonction résidu (1.1) a la forme

$$F(x) = \begin{bmatrix} obs_1 \\ obs_2 \\ \vdots \\ obs_{\frac{m}{2}} \end{bmatrix}, \tag{2.48}$$

avec

$$obs_i = \begin{bmatrix} x_i - x_{obs_i} \\ y_i - y_{obs_i} \end{bmatrix}. \tag{2.49}$$

La taille m du résidu (2.48) correspond donc à $m = 2 \times nobs$ où $nobs$ est le nombre total d'observations. Comme on peut le voir, on souhaite reconstruire l'objet en 3 dimensions alors qu'il n'y a que des coordonnées (x, y) pour chaque observation. Cela est dû au fait que la photo est en 2 dimensions et on doit donc projeter nos coordonnées en 3 dimensions sur un espace en 2 dimensions. On effectue cette projection dans notre fonction résidu. Pour chaque bloc de notre vecteur résidu on a besoin de coordonnées en 3 dimensions d'un point et des paramètres d'une caméra afin de projeter ce point sur l'image en 2 dimensions. Notre vecteur

de variables a donc la forme :

$$\begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_{npts} \\ C_1 \\ C_2 \\ \vdots \\ C_{ncams} \end{bmatrix}. \tag{2.50}$$

Dans cette équation (2.50), X_i correspond aux coordonnées en 3 dimensions du ième point et C_j correspond aux paramètres de la jème caméra. Ainsi, chaque bloc X_i est de la forme :

$$X_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}. \tag{2.51}$$

De plus, chaque caméra possède 9 paramètres, ce qui nous donne un vecteur C_j détaillé comme suit :

$$C_j = \begin{bmatrix} r_j^x \\ r_j^y \\ r_j^z \\ t_j^x \\ t_j^y \\ t_j^z \\ k_j^1 \\ k_j^2 \\ f_j \end{bmatrix}. \tag{2.52}$$

On a donc (2.50) $\in \mathbb{R}^{3\times npts+9\times ncams}$.

La matrice jacobienne, quant à elle, a la forme :

$$J(X,C) = \begin{bmatrix} JX_{1,1} & \dots & JX_{1,npts} & JC_{1,1} & \dots JC_{1,ncams} \\ \vdots & & \vdots & \vdots & \vdots \\ JX_{nobs,1} & \dots & JX_{nobs,npts} & JC_{nobs,1} & \dots JC_{nobs,ncams} \end{bmatrix}, \tag{2.53}$$

où $JX_{i,j} \neq 0$ si et seulement si il y a un point j qui correspond à l'observation i. De même, $JC_{i,j} \neq 0$ si et seulement si il y a une caméra j qui correspond à l'observation i. Or, les points et les caméras sont corrélés mais on peut réarranger à l'avance l'un ou l'autre pour avoir une structure de la jacobienne plus avantageuse. Comme la partie correspondant aux points $JX_{i,j}$ est plus volumineuse dans la jacobienne, on les organise par ordre croissant ce qui nous donne la structure en V quasiment diagonale par blocs caractéristique de ces problèmes. Cela signifie aussi qu'on peut déterminer la structure de la matrice jacobienne avec un algorithme de complexité linéaire.

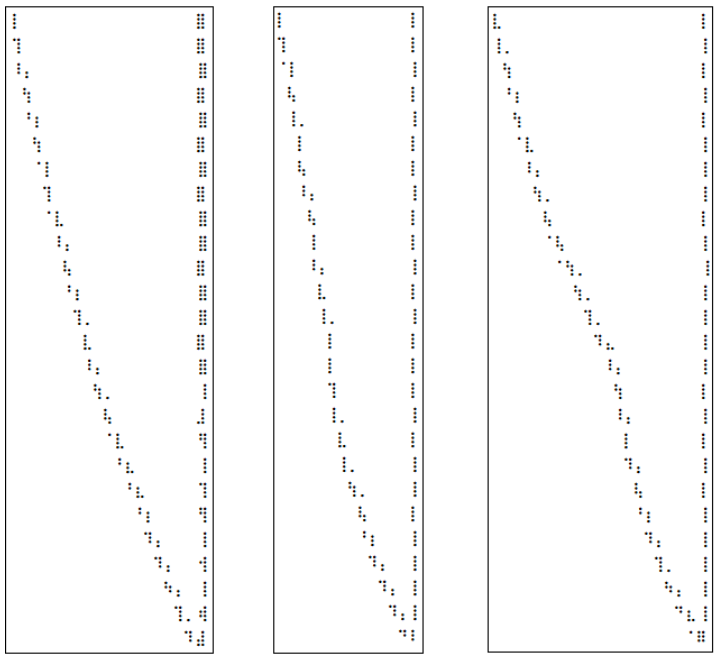

FIGURE 2.1 Exemples de jacobiennes de problèmes d'ajustement de faisceaux

2.8 Le langage Julia

Nous travaillons avec le langage Julia, adéquat grâce au dispatch multiple, un mécanisme informatique permettant d'écrire une fonction une seule fois, et voir celle-ci compilée à la volée pour différentes précisions suivant le type des arguments. Sur les plateformes émergentes qui offrent plusieurs systèmes en virgule flottante, telles que les cartes graphiques Pascal, mes

algorithmes permettent de potentielles économies d'énergie. L'écosystème Julia permet de diffuser mes méthodes efficacement et aux utilisateurs de les appliquer facilement à divers domaines.

Un autre élément crucial de Julia est que c'est un langage mathématique qui possède les avantages d'un langage haut niveau, c'est-à-dire la facilité de programmation l'absence de gestion directe de la mémoire entre autre, mais qui conserve des performances comparables à celles de langages bas niveau comme C,C++ et Fortran.

2.9 Temps d'exécution et consommation énergétique

Jusqu'à présent nous avons mentionné le fait de rendre les algorithmes plus rapides et moins énergivores sans expliquer la distinction entre les deux. Même si il semble assez intuitif qu'un calcul plus rapide sur une même machine consomme moins d'énergie, la réalité est un peu plus compliquée.

En effet, le temps d'exécution et la consommation énergétique dépendent aussi des accès en mémoire par exemple avec un facteur différent pour l'un et l'autre. Cependant, il existe des mesures assez précises pour comparer les deux. En particulier pour les systèmes embarqués où cette information est cruciale [29].

Outre le nombre d'opérations, la précision dans laquelle les opérations sont effectuées ont un impact direct sur le temps d'exécution et la consommation d'énergie [30] tout comme le fait d'exécuter le code sur GPU au lieu de CPU comme le permettent certaines librairies de Julia [31]. L'amélioration d'efficacité du code en temps d'exécution et consommation est un enjeu critique étant donné la consommation énergétique des machines sur lesquelles sont lancées ces algorithmes [32].

Cependant cette amélioration présente des limites. D'une part nous ne couvrons que les améliorations d'un point de vue mathématique et les performances en temps et énergie ne sont pas analysées en profondeur. D'autre part, ces économies n'ont d'intérêt que si elles n'entraînent pas par la suite un plus grand nombre de calculs.

2.10 GPU

Les GPUs actuels ne sont plus seulement des moteurs graphiques mais aussi des processeurs avec une forte capacité de parallélisation et des performances qui peuvent dépasser les CPUs dans certaines situations [33].

La rapide augmentation de leur programmabilité et de leurs performances a nourri un intérêt

grandissant dans la communauté scientifique qui leur a trouvé un grand nombre d'applications.

Il y a 3 principales raisons d'utiliser un GPU pour un problème donné :

1. Le nombre d'opérations à effectuer est important.
2. Il y a une fore possibilité de parallélisation.
3. La capacité de calcul est plus important que la latence.

Les GPUs peuvent en effet effectuer un très grand nombre d'opérations en simultané et ont une plus grande bande passante. Leur défaut étant que la gestion de priorité n'étant pas aussi forte que sur un CPU, il est plus difficile d'assurer un résultat rapide.

Les GPUs sont particulièrement efficaces pour effectuer la même opération sur un très grand nombre d'éléments. En effet, les éléments sont regroupés en blocs et ces blocs sont traités en parallèle.

Un dernier point intéressant à noter est que la structure des GPUs est traditionnellement optimisée pour gérer les opérations en précision simple. En effet, historiquement la précision simple était suffisante pour les opérations graphiques ce qui fait que pour des opérations de calculs en double précision les performances sont réduites.

CUDA est une librairie en C et C++, portée aussi en Julia [34]. Elle permet de plus facilement porter le code sur GPU et sera notre référence lorsque nous parlerons de portage sur GPU dans la partie 3.7.

2.11 Multi-précision

La disponibilité grandissante de basses et très hautes précisions pour l'arithmétique provoque un intérêt pour l'utilisation dans différents domaines dont l'optimisation. Cette augmentation de disponibilité est principalement due à l'utilisation de précision multiple en intelligence artificielle, mais une fois que les hardwares sont fabriqués ils peuvent être utilisés pour toutes sortes de calculs [35].

Dans la plupart des méthodes itératives, le coût est principalement dominé par le coût d'un ou plusieurs produits matrice-vecteur. Dans ces cas, l'utilisation de multiple précisions, c'est-à-dire par exemple Float16, Float32, Float64 ou Float128 est un ingrédient clé pour obtenir des résultats performants [36].

2.12 Différentiation automatique

La différentiation automatique est une technique qui se base sur la règle de la chaîne pour les dérivations. Elle part du principe que le code de n'importe quelle fonction est ultimement découpé en opérations mathématiques élémentaires lorsqu'il exécuté par un processeur. Ces opérations mathématiques élémentaires ont des dérivées connues et faciles à calculer et en combinant cette information avec la règle de la chaîne on peut alors obtenir la dérivée de la fonction elle même. Cette technique a été découverte par plusieurs personnes dans les années 1970 notamment par Linnainmaa et Seppo [37]. Elle a ensuite été remise au goût du jour dans les années 2000 [38, 39].

En appliquant la règle de la chaîne lors de l'évaluation d'une fonction on utilise la technique que l'on appelle "Forward differentiation" pour "Dérivation en avant". Une autre façon de procéder est de garder en mémoire toutes les variables intermédiaires qui apparaissent lors de l'utilisation de la règle de la chaîne quand on évalue une fonction, puis de dériver notre fonction en partant des dernières variables et en remontant jusqu'à l'expression de la fonction originelle. Cette technique est appelée "Reverse differentiation" pour "Dérivation en arrière". Chacune de ces 2 techniques possède un avantage selon les opérations que l'on souhaite effectuer comme nous le reverrons dans la partie 3.8.

Nous pouvons illustrer cette définition avec l'exemple tiré du livre Automatic Differentiation in MATLAB Using ADMAT with Applications [40].

Soit $F(x) = \begin{bmatrix} 3x_1 + 2x_2 \\ 5x_1^2 + 4x_1 + x_2 \end{bmatrix}$ et supposons que nous soyons au point $\hat{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$. Nous avons donc $F(\hat{x}) = \begin{bmatrix} 7 \\ 11 \end{bmatrix}$ et sa jacobienne $J(\hat{x}) = \begin{bmatrix} 3 & 2 \\ 14 & 1 \end{bmatrix}$. Une suite de variable intermédiaires qui nous permettrait d'appliquer la règle de la chaîne est

$$y_1 = 3x_1, y_2 = 2x_2, y_3 = x_1^2, y_4 = 5y_3, y_5 = 4x_1, y_6 = y_5 + x_2, \tag{2.54}$$

et finalement notre résultat est obtenu grâce à

$$z_1 = y_1 + y_2, z_2 = y_4 + y_6. \tag{2.55}$$

On peut bien voir $z = F(\hat{x}) = \begin{bmatrix} 7 \\ 11 \end{bmatrix}$. Pour obtenir notre jacobienne on dérive nos variables des équations intermédiaires (2.54) et (2.55).

$$A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \\ 2 & 0 \\ 0 & 0 \\ 4 & 0 \\ 0 & 1 \end{bmatrix}, \quad L = \begin{bmatrix} -1 & & & & & \\ & -1 & & & & \\ & & -1 & & & \\ & & 5 & -1 & & \\ & & & & -1 & \\ & & & & 1 & -1 \end{bmatrix} \tag{2.56}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad M = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}. \tag{2.57}$$

On peut finalement créer une matrice

$$C = \begin{bmatrix} A & L \\ B & M \end{bmatrix} \tag{2.58}$$

La matrice C correspond à la jacobienne de notre règle de la chaîne qui décompose notre fonction F en plusieurs équations intermédiaires (2.54) et (2.55). La jacobienne de F en $\hat{x}$ peut alors être obtenue en calculant le complément de Schur sur (2.58). Et on obtient bien $J = B - ML^{-1}A = \begin{bmatrix} 3 & 2 \\ 14 & 1 \end{bmatrix}$.

CHAPITRE 3 LEVENBERG-MARQUARDT EN PRÉCISION MIXTE POUR LES PROBLÈMES D'AJUSTEMENTS DE FAISCEAUX

Nous souhaitons améliorer la résolution de problèmes aux moindres carrés non linéaires pour les problèmes d'ajustement de faisceaux. Pour cela nous avons exploré diverses pistes :

1. utilisation de l'algorithme de Levenberg-Marquardt ;
2. méthodes de résolution du sous-problème ;
3. utilisation de précision multiple ;
4. utilisation du GPU ;
5. utilisation de différentiation automatique ;
6. optimisation des paramètres et conditions d'arrêt.

Ces différents point constituent le squelette du mémoire. Nous allons donc commencer par explorer l'utilisation de l'algorithme de Levenberg-Marquardt.

3.1 Algorithme de Levenberg-Marquardt

Dans la section 2.3 de la revue de littérature, nous avons vu qu'il existe différentes versions de Levenberg-Marquardt. Dans notre cas, nous allons commencer par utiliser la version de [11] qui diffère légèrement de celui de Algorithme 2.2 de [11] par rapport aux mises à jour de λ. Au lieu de diminuer indéfiniment vers 0, on choisit un seuil λ_{min} à partir duquel si $\lambda < \lambda_{min}$ alors $\lambda = \lambda_{min}$. Ce changement permet en pratique de limiter la détérioration du conditionnement du problème et à λ de plus facilement remonter à des valeurs plus importantes en quelques mauvaises itérations. On obtient de meilleurs résultats numériques comme on le verra dans la figure 3.3 par exemple. La convergence de l'algorithme telle que donnée dans la partie 2.3.4 n'est pas impactée si on choisit une valeur de λ_{min} suffisamment petite.

Comme mentionné dans la revue de littérature, on peut résoudre exactement ou approximativement le sous-problème de la ligne 9 de l'algorithme de Levenberg-Marquardt 2.3.1. Les méthodes directes sont plus couramment utilisées pour résoudre les problèmes dont la jacobienne est petite et dense tandis que les méthodes itératives sont plutôt utilisées pour les problèmes dont la jacobienne est grande et creuse. Nous allons donc observer les inconvénients et avantages de ces deux approches dans le cas des problèmes d'ajustement de faisceaux.

Algorithme 3.1.1 La méthode de Levenberg-Marquardt par régularisation

On choisit les constantes $0 < \eta_1 \leq \eta_2 < 1$, $0 < \sigma_2 < 1 < \sigma_1$, $\epsilon_a^g > 0$ et $\epsilon_r^g > 0$
On choisit $x_0 \in \mathbb{R}^n$, $\lambda_0 \geq 0$ et λ_{min}
On évalue $F(x_0)$ et $J(x_0)$
On calcule $\|F(x_0)\|$ et $\|J(x_0)^T F(x_0)\|$
pour k = 0,1,... **faire**
    si $\|J(x_k)^T F(x_k)\| \leq \epsilon_a^g + \epsilon_r^g \|J(x_0)^T F(x_0)\|$ **alors**
        On sort de l'algorithme
    fin si
    On calcule le pas d_k tel que

$$d_k \approx \arg\min_{d \in \mathbb{R}^n} \frac{1}{2} \left\| \begin{bmatrix} J(x_k) \\ \lambda_k I \end{bmatrix} d + \begin{bmatrix} F(x_k) \\ 0 \end{bmatrix} \right\|^2 \text{ (Approximativement)}$$

    On calcule le ratio

$$\rho_k = \frac{\frac{1}{2}(\|F(x_k)\|^2 - \|F(x_k + d_k)\|^2)}{\frac{1}{2}(\|F(x_k)\|^2 - \|J(x_k)d_k + F(x_k)\|^2 - \lambda_k^2 \|d_k\|^2)}$$

    si $\rho_k < \eta_1$ **alors**
        On fixe $x_{k+1} = x_k$
        si $\lambda_k < \lambda_{min}$ **alors**
            $\lambda_k = \lambda_{min}$
        sinon
            $\lambda_k = \sigma_1 \lambda_k$
        fin si
    sinon
        On fixe $x_{k+1} = x_k + d_k$
        On évalue $F(x_{k+1})$ et $J(x_{k+1})$
        si $\rho_k \geq \eta_2$ **alors**
            $\lambda_k = \sigma_2 \lambda_k$
        fin si
        si $\lambda_k < \lambda_{min}$ **alors**
            $\lambda_k = \lambda_{min}$
        fin si
    fin si
fin pour

3.2 Méthodes directes et CG (fonctionnement de Ceres)

Comme on peut le constater dans la littérature, le solveur Ceres [25] est utilisé à l'origine pour modéliser et résoudre des problèmes d'ajustement de faisceaux. Afin de justifier notre approche nous commençons par comprendre et expliquer la méthode utilisée dans Ceres et

ses limitations.

Notons $H_\lambda(x_k) = J(x_k)^T J(x_k) + \lambda^2 I$ où $\lambda > 0$. H_λ est symétrique définie positive et est extraite des équations normales (2.16). Il s'avère en pratique que comme $J(x_k)$ est une concaténation horizontale de deux matrices, comme on peut le voir dans la figure 2.1, donc H_λ a la forme

$$H_\lambda = \begin{bmatrix} B & E \\ E^T & C \end{bmatrix}. \tag{3.1}$$

En particulier H_λ a même une forme de matrice flèche comme on peut le voir dans la figure 3.1. À cause de notre implémentation de la fonction résidu et jacobienne, la matrice H_λ est une flèche orientée vers le bas à droite et dans l'implémentation de Ceres elle est orientée vers le haut à gauche mais les résultats sont identiques et il ne s'agit que d'une permutation des éléments.

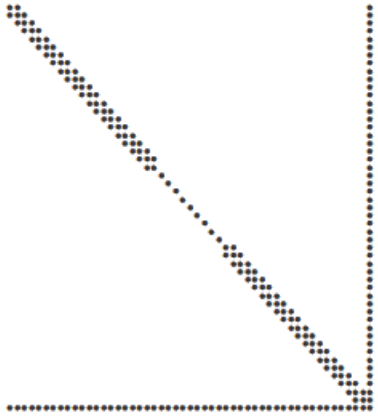

FIGURE 3.1 Matrice H_λ du problème 49-7776.

D'après la partie 2.7 et la figure 2.1 de la revue de littérature, on a déterminé que les $p \times s$ premières colonnes constituent une matrice quasiment bloc-diagonale et les $q \times c$ dernières colonnes constituent une matrice quasiment dense, avec p qui vaut le nombre de points du modèle et q qui vaut le nombre de caméras du modèle. Comme nous traitons des points en 3 dimensions $s = 3$ et c correspond au nombre de paramètres des caméras qui vaut entre 6 et 9 mais dans notre cas $c = 9$. On va noter $n_1 = p \times s$ et $n_2 = q \times c$. Ainsi, pour la matrice H_λ de (3.1), on détermine que $B \in \mathbb{R}^{n_1 \times n_1}$ est diagonale par blocs. De même, $C \in \mathbb{R}^{n_2 \times n_2}$ est diagonale par blocs et $E \in \mathbb{R}^{n_1 \times n_2}$ est creuse. Par exemple pour le problème 49-7776 de la figure 3.1, le bloc B est carré de taille $23328 = 7776 \times 3$ et le bloc C est carré de taille $441 = 49 \times 9$. On peut alors réécrire les équations normales (2.16) sous la forme

$$\begin{bmatrix} B & E \\ E^T & C \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} v \\ w \end{bmatrix}. \tag{3.2}$$

La matrice B est bloc diagonale donc on peut facilement calculer son inverse. On peut ensuite facilement déterminer Δz à partir de Δy :

$$\Delta y = B^{-1}(v - E\Delta z). \tag{3.3}$$

Le système qu'il nous reste à résoudre est

$$\left[C - E^T B^{-1} E\right] \Delta z = w - E^T B^{-1} v. \tag{3.4}$$

Dans la suite on notera les éléments de (3.4) comme suit :

$$S = C - E^T B^{-1} E, \tag{3.5}$$

et

$$u = w - E^T B^{-1} v. \tag{3.6}$$

La matrice dans (3.5) présente dans le système (3.4), est le complément de Schur S de C dans H_λ. La matrice S est symétrique définie positive. Comme H_λ est symétrique définie positive (DP), B et C le sont aussi, et elles sont donc inversibles. De plus, le fait que H_λ est DP permet de prouver que C a un meilleur conditionnement que H_λ comme mentionné dans [1].

On peut résoudre (3.4) en utilisant des méthodes directes comme une factorisation de Cholesky ou une méthode itérative comme le gradient conjugué préconditionné par la matrice C^{-1}. Enfin, une dernière amélioration apportée par Ceres est d'utiliser les opérateurs linéaires pour ne pas former la matrice S. Cette méthode de résolution des problèmes d'ajustement de faisceaux semble donc efficace mais plusieurs problèmes se posent et le premier concerne le conditionnement des matrices.

3.2.1 Conditionnement des matrices

Afin d'utiliser la technique du complément de Schur, nous avons besoin d'effectuer le produit $J_\lambda^T J_\lambda$. De manière plus générale, toutes les méthodes de résolution du problème proposées

par Ceres comme mentionnées dans la parties 2.6 nécessitent la création de H_λ. Et comme il est directement mentionné sur le site de Ceres, en pratique le conditionnement de H_λ est mauvais [41]. En effet, le conditionnement de la matrice jacobienne J d'un problème d'ajustement de faisceaux est souvent déjà très élevé. Or, si $J \in \mathbb{R}^{m\times n}$ avec $m > n$, et si

$$J = U \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^T \tag{3.7}$$

est la décomposition en valeurs singulières de J, alors $J^T J = V\Sigma^2 V^T$. Les valeurs propres de $J^T J$ sont exactement les carrés des valeurs singulières de J et on a le conditionnement $\kappa(J^T J) = \kappa(J)^2$. De plus, $H_\lambda = J^T J + \lambda^2 I = V(\Sigma^2 + \lambda^2 I)V^T$, et donc les valeurs propres de H_λ sont $\sigma_i^2 + \lambda^2$. Cela signifie qu'en effectuant le produit $J_\lambda^T J_\lambda$ on a le résultat sur le conditionnement

$$\kappa(H_\lambda) = \kappa(J_\lambda)^2. \tag{3.8}$$

En formant H_λ, on élève le conditionnement du problème au carré ce qui peut devenir problématique.

Tableau 3.1 Conditionnement des matrices pour différents problèmes d'ajustement de faisceaux avec $\lambda = 1$.

Problème	49-7776	174-50489	287-182023	1778-993923
Catégorie	ladybug	trafalgar	dubrovnik	venice
$\kappa(J)$	∞	∞	∞	∞
$\kappa(J_\lambda)$	2.21e5	7.32e5	2.01e6	3.78e12
$\kappa(H_\lambda)$	4.90e10	5.37e11	4.05e12	1.43e25
$\kappa(S)$	3.33e9	1.34e11	2.43e12	1.21e22
$\kappa(J_\lambda^s)$	1.57e4	2.76e5	1.7e4	5.63e6
$\kappa(H_\lambda^s)$	2.49e8	7.66e10	3.15e8	3.18e13
$\kappa(S^s)$	8.54e6	4.60e8	2.12e7	9.17e10

Pour les différents problèmes utilisés dans le tableau 3.1, la matrice jacobienne J_λ n'est pas de rang maximal, sa valeur propre minimale est donc nulle et son conditionnement est infini. De plus on aperçoit aussi dans le tableau 3.1 que le résultat théorique $\kappa(S) \leq \kappa(H_\lambda)$ est bien visible en pratique même si $\kappa(S)$ est très proche de $\kappa(H_\lambda)$. Enfin, même si ça n'est pas vrai pour tous les problèmes, on constate en règle générale que plus le problème est grand, c'est à dire que plus J_λ est grand, plus le conditionnement de la matrice se détériore et ce même si l'on choisit $\lambda = 1$.

Une solution qui est utilisée en pratique pour réduire la valeur de $\kappa(J_\lambda)$ est une mise à l'échelle de J_λ, on note alors la matrice mise à l'échelle J_λ^s. En particulier, comme $\kappa(H_\lambda) = \kappa(J_\lambda)^2$, le gain sur le conditionnement de H_λ est d'autant plus grand. Nous avons donc utilisé le scaling de Ruiz [42], qui consiste à mettre à l'échelle les lignes et colonnes de J. On note $J_\lambda^s = D_1 J_\lambda D_2$ où J_λ et $J_\lambda^s \in \mathbb{R}^{(m+n)\times n}$, $D_1 \in \mathbb{R}^{(m+n)\times(m+n)}$ et $D_2 \in \mathbb{R}^{n\times n}$. Or, lorsqu'on forme H_λ, on a

$$H_\lambda = J_\lambda^T J_\lambda \tag{3.9}$$
$$= D_2^{-1} J_\lambda^T D_1^{-1} D_1^{-1} J_\lambda D_2^{-1}. \tag{3.10}$$

On ne peut donc utiliser qu'une mise à l'échelle sur les colonnes de J_λ. Dans le tableau 3.1, on constate que le conditionnement d $\kappa(S^s)$ set à peine plus élevé que $\kappa(J_\lambda)$ pour le problème 49-7776, mais cet écart se creuse lorsque la taille de J_λ augmente.

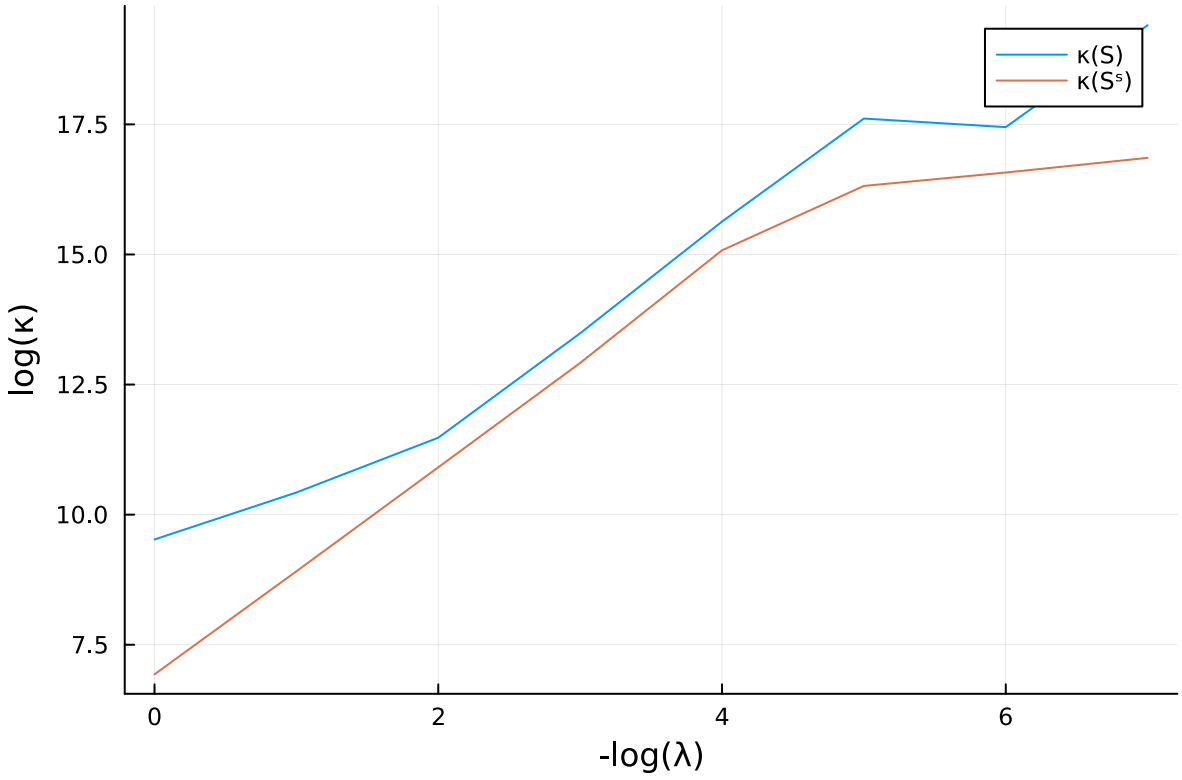

FIGURE 3.2 Conditionnement du complément de Schur de H_λ du problème 49-7776

Les valeurs du tableau 3.1 ont été obtenues avec $\lambda = 1$, or le but de l'algorithme de Levenberg-Marquardt est de réduire la valeur de λ lorsqu'on approche d'un minimum local. Dans la

figure 3.2, on constate que $\kappa(S)$ augmente de manière exponentielle lorsque λ diminue de manière exponentielle. Lorsque $\lambda < 10^{-4}$, $\kappa(S) > 10^{15}$ et même lorsque S est mis à l'échelle $\kappa(S^s) > 10^{15}$. Cela signifie donc que pour toutes les méthodes actuellement utilisées par Ceres, le conditionnement de H_λ devient trop mauvais lorsque la taille du problème est trop importante. Explorer des méthodes pour lesquelles on ne forme pas H_λ permet alors théoriquement de résoudre un système dont le conditionnement est la racine carrée de celui avec H_λ.

Une factorisation de Cholesky ou l'algorithme de CG appliqué au complément de Schur des équations normales présente donc l'avantage de réduire énormément la taille du problème à résoudre. Pour le problème 49-7776, le système à résoudre est environ 50 fois plus petit. Cependant, le problème principal est que cela nécessite de former les équations normales et donc détériorer considérablement le conditionnement du système.

De plus, même si le mauvais conditionnement des matrices est le principal moteur qui motive l'exploration d'autres méthodes de résolutions pour les problèmes d'ajustement de faisceaux, Ceres présente plusieurs autres limitations.

3.2.2 Limitations de Ceres

Les autres limitations de Ceres sont plutôt d'ordre informatique. En effet, l'API de Ceres ne permet pas d'interfacer le code avec Julia, or dans notre cas on souhaite profiter des avantages de Julia comme le multiple dispatch qui permet facilement un changement de précision en cours d'exécution du code. En créant une librairie open-source complétant celle de Julia-Smooth-Optimizers [43], nous encourageons aussi le développement de l'écosystème de Julia.

Enfin, pour que le code de Ceres soit optimal il est nécessaire de segmenter la fonction résidu en plusieurs blocs ce qui fonctionne relativement bien pour les problèmes d'ajustement de faisceaux mais qui n'est pas toujours le cas. Le code que j'ai développé permet de résoudre un problème de moindres carrés non-linéaires pour tous les problèmes d'ajustement de faisceaux qui respectent la structure de NLPModels [44]. Cela signifie qu'on a uniquement besoin d'une fonction résidu, d'un produit jacobien-vecteur et des informations sur la taille du problème.

Nous allons donc explorer les différentes méthodes qui n'utilisent pas le complément de Schur de H_λ et en particulier qui ne nécessitent pas l'évaluation de H_λ afin d'éviter la détérioration du conditionnement du problème.

3.3 LSMR

Si notre but est de ne pas utiliser H_λ, la première idée qu'on peut avoir est d'utiliser la méthode LSMR au lieu du gradient conjugué pour résoudre le sous-problème. Les résultats de la littérature indiquent que LSMR est meilleur que CG en théorie et en pratique. Dans notre cas on constate que pour un nombre d'itérations donnés de Levenberg-Marquardt avec les mêmes critères d'arrêt pour LSMR et CG, la version avec LSMR résout environ 50% de problèmes en plus que CG comme on peut le voir dans les tableaux B.1 et B.2, même si dans l'ensemble, les 2 algorithmes résolvent un nombre assez faible de problèmes selon les critères donnés.

L'algorithme itératif utilisé pour la résolution du sous-problème dans l'article [11] est LSQR, dans notre cas nous utilisons LSMR car $\|Ar_k\|$ et $\|r_k\|$ décroissent de manière monotone contrairement à LSQR où seul $\|r_k\|$ décroît de manière monotone. Cependant LSQR permet de résoudre un problème des moindres carrés linéaires avec plusieurs valeurs de λ à la fois. En effet, chaque valeur de λ supplémentaire ne requiert que n rotations supplémentaires par itération et un stockage de $2n$ valeurs supplémentaires [23]. Si le choix de λ est problématique, LSQR pourrait à ce moment être plus intéressant.

En pratique, utiliser LSMR sur le jacobien $J(x_k)$ et la fonction résidu $F(x_k)$ avec un paramètre de régularisation λ n'est qu'une légère amélioration par rapport au fait de former H_λ, car le processus le fait de manière indirecte. De plus, sans former H_λ, on ne peut pas utiliser la structure de H_λ et créer un préconditionneur adapté. L'exploration des différents préconditionneurs n'utilisant pas la structure de H_λ pour le sous-problème est intéressante et aussi très présente dans la littérature [45], cependant il semblent difficile de trouver mieux que les matrices blocs diagonales B et C en préconditionneurs pour LSMR pour les problèmes d'ajustement de faisceaux comme on peut la voir dans (3.2) à cause de la forte concordance au problème et du faible coût de création de B et C.

Pour résumer, LSMR et LSQR ont sensiblement les mêmes avantages et inconvénients lorsqu'ils sont appliqué sur les problèmes d'ajustement de faisceaux. Ils ne nécessitent pas la formation des équations normales ni-même celle de la jacobienne si on dispose d'un opérateur produit jacobien-vecteur. L'inconvénient principal est que la marge d'amélioration sur ces méthodes est très complexe si on ne dispose pas de la jacobienne pour créer un préconditionneur adapté.

Nous allons donc continuer à explorer d'autres moyens de résoudre le sous-problème de la ligne 9 de Algorithme 3.1.1 sans former H_λ.

3.4 MINRES

Une autre façon de résoudre le sous-problème de l'algorithme de Levenberg-Marquardt sans former H_λ consiste à créer un nouveau système à partir des conditions de KKT du sous-problème comme on peut le voir ci-dessous. On réécrit le sous-problème

$$\arg\min_{d\in\mathbb{R}^n} \tfrac{1}{2}\|J(x_k)d + F(x_k)\|^2 + \tfrac{1}{2}\lambda_k^2\|d\|^2, \tag{3.11}$$

puis on définit $r(x_k) := J(x_k)d+F(x_k)$, ce qui nous permet de reformuler le système ci-dessous

$$\arg\min_{d\in\mathbb{R}^n} \tfrac{1}{2}\|r(x_k)\|^2 + \tfrac{1}{2}\lambda_k^2\|d\|^2 \quad \text{s.c. } r(x_k) - J(x_k)d = F(x_k). \tag{3.12}$$

On définit le Lagrangien

$$\mathcal{L}(y, r, d) = \tfrac{1}{2}\|r(x_k)\|^2 + \tfrac{1}{2}\lambda_k^2\|d\|^2 - y^T(r(x_k) - J(x_k)d - F(x_k)), \tag{3.13}$$

qui permet alors d'écrire les conditions de KKT dans l'équation suivante

$$\begin{aligned} r(x_k) - J(x_k)d &= F(x_k) \\ r(x_k) - y &= 0 \\ \lambda_k^2 d + J(x_k)^T y &= 0. \end{aligned}$$

En remplaçant $r(x_k) = y$, on obtient le système

$$\begin{bmatrix} I & -J(x_k) \\ J(x_k)^T & \lambda^2 I \end{bmatrix} \begin{bmatrix} r(x_k) \\ d \end{bmatrix} = \begin{bmatrix} F(x_k) \\ 0 \end{bmatrix}. \tag{3.14}$$

Pour obtenir une matrice symétrique, on peut le reformuler

$$\begin{bmatrix} I & J(x_k) \\ J(x_k)^T & -\lambda^2 I \end{bmatrix} \begin{bmatrix} -r(x_k) \\ d \end{bmatrix} = -\begin{bmatrix} F(x_k) \\ 0 \end{bmatrix}. \tag{3.15}$$

Pour la suite, on va noter

$$A_\lambda(x_k) = \begin{bmatrix} I & J(x_k) \\ J(x_k)^T & -\lambda^2 I \end{bmatrix}, \tag{3.16}$$

et

$$F_0(x_k) = \begin{bmatrix} F(x_k) \\ 0 \end{bmatrix}. \tag{3.17}$$

Pour la suite on va désormais résoudre le système (3.15) à la ligne 9 de Algorithme 3.1.1.

TABLEAU 3.2 Comparaison entre le conditionnement du complément de Schur mis à l'échelle et du système augmenté pour différents problèmes.

Problème	49-7776	174-50489	287-182023	1778-993923
Catégorie	ladybug	trafalgar	dubrovnik	venice
$\kappa(S^s)$	8.54e6	4.60e8	2.12e7	9.17e10
$\kappa(A)$	2.21e5	7.3e5	2.01e6	3.79e12

Comme on peut le voir dans le tableau 3.2, pour une partie des problèmes testés le conditionnement du système augmenté est meilleur que celui du complément de Schur de H_λ avec J_λ mis à l'échelle. Cet écart a lieu alors qu'on choisit le conditionnement du complément de Schur mis à l'échelle comparé avec celui du système augmenté sans modifications. Une approche qui n'a été que peu ou pas explorée est l'utilisation de multiples précisions et de GPUs pour la résolution du problème (3.15). En particulier, on pourrait résoudre le système en faible précision calculé sur CPU et l'utiliser en préconditionneur d'une méthode itérative en plus haute précision sur GPU. Cette technique peut paraître contre-intuitive étant donné que les calculs sur GPU sont plus performants en faible précision. Cependant, les méthodes directes ne peuvent pas être exécutées sur GPU, il s'agit donc du meilleur compromis que l'on peut avoir pour l'instant.

On peut effectuer une factorisation LDL^T et LDL^T incomplètes de A_λ et l'utiliser en préconditionneur de MINRES appliqué sur (3.15). Lorsqu'on teste sur quelques problèmes avec une factorisation LDL^T ou LDL^T incomplètes, les valeurs de la diagonale D augmentent fortement au bout de quelques itérations et ont tendance à détruire le système, en particulier si λ_{min} est petit.

La figure 3.3 montre l'évolution des valeurs du maximum de la diagonale D de la factorisation LDL^T sur le problème 49-7776 selon les valeurs de λ_{min} que l'on choisit. On constate que lorsque λ_{min} diminue trop, la valeur maximale de D augmente de manière exponentielle. En particulier si $\lambda_{min} = 10^{-3}$, au bout de 60 itérations la valeur maximale de D dépasse 10^{15} ce

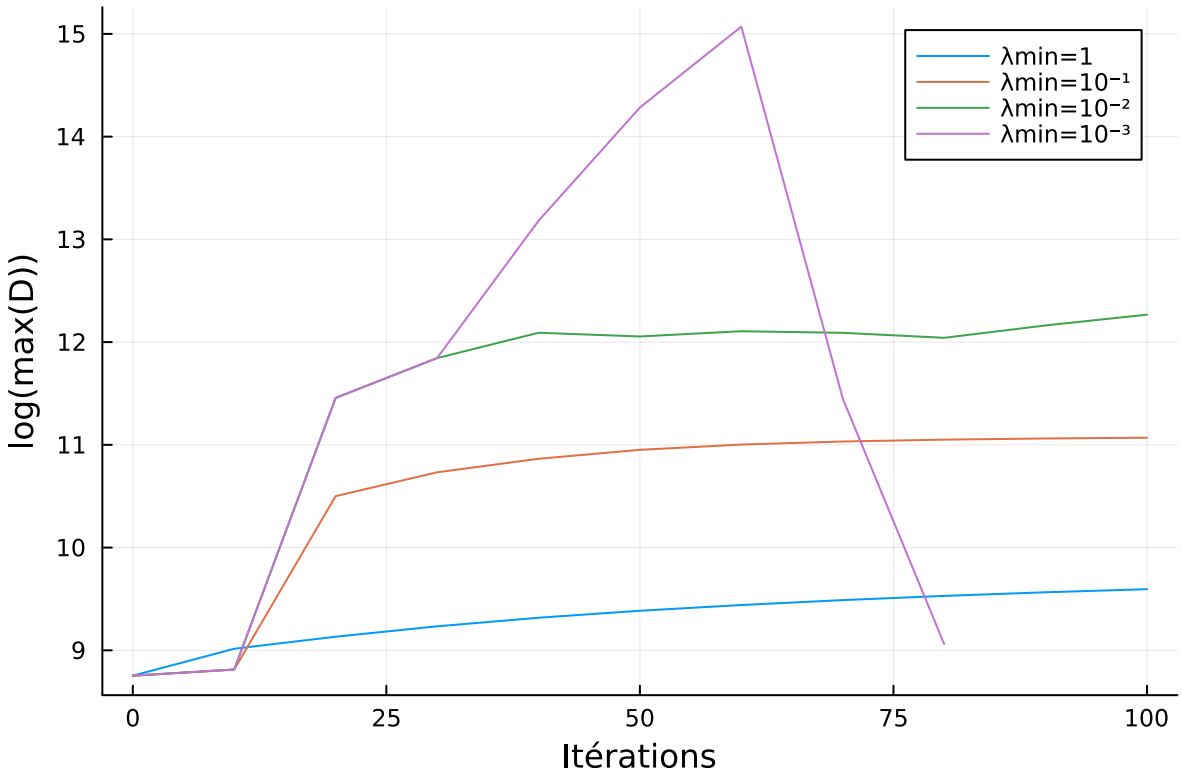

FIGURE 3.3 Valeur maximale sur la diagonale d'une factorisation LDL^T de A_λ pour le problème 49-7776.

qui introduit des erreurs numériques et fait échouer l'algorithme.

Pour limiter les valeurs extrêmes sur la diagonale on peut utiliser un mise à l'échelle de A_λ. En pratique, on va utiliser le scaling de Ruiz [42]. De plus, pour nous assurer qu'il s'agit de la factorisation LDL^T qui est problématique et non l'algorithme MINRES, j'ai aussi implémenté une version de Levenberg-Marquardt qui résout exactement le système 3.15 pour lequel le même problème se pose. Comme A_λ est un matrice symétrique, le scaling de Ruiz a la forme suivante

$$A^s_\lambda(x_k) = D_3 A_\lambda(x_k) D_3, \tag{3.18}$$

avec $A_\lambda(x_k)$, $A^s_\lambda(x_k)$ et $D_3 \in \mathbb{R}^{(m+n)\times(m+n)}$. Ce qui revient à résoudre le système

$$A^s_\lambda(x_k) d^s_k = -D_3 F^s_0(x_k). \tag{3.19}$$

Ensuite, on isole d_k

$$\begin{bmatrix} -r(x_k) \\ d_k \end{bmatrix} = D_3 d_k^s \tag{3.20}$$

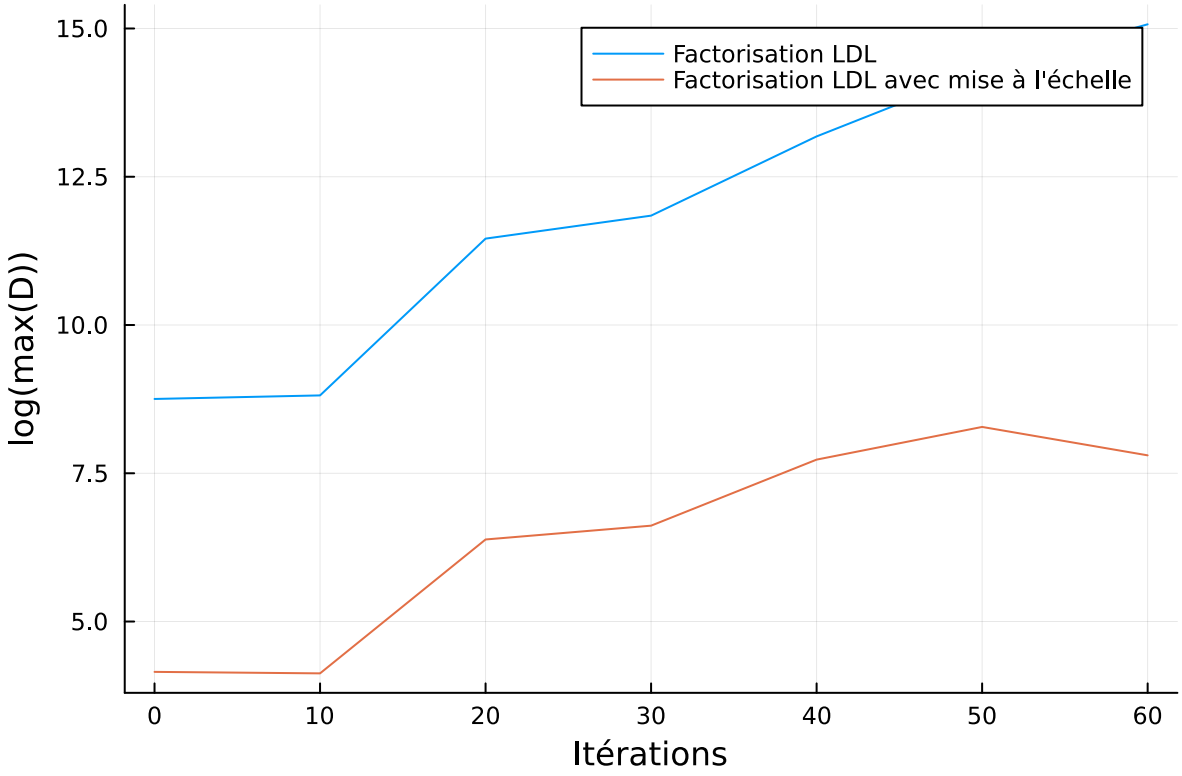

FIGURE 3.4 Comparaison avec et sans scaling des valeurs maximale sur la diagonale d'une factorisation LDL^T de A_λ pour le problème 49-7776 avec $\lambda_{min} = 10^{-3}$

Comme on peut le voir dans la figure 3.4, la mise à l'échelle de A_λ réduit le problème mais pas suffisamment, parce que sur certains problèmes on constate encore $\frac{1}{2}\|F(x_k)\|^2 - \|J(x_k)d_k + F(x_k)\|^2 < 0$ ce qui signifie que d_k n'est pas une direction de descente et ce même lorsqu'on résout le système uniquement avec une méthode directe LDL^T ce qui n'est pas possible en arithmétique exacte. On en conclut qu'il y a un trop grand nombre d'erreurs numériques qui s'introduisent au fil des itérations.

Pour corriger ce problème on va utiliser l'algorithme ma57 [46] qui est une factorisation LBL^T [47]. Cela signifie qu'au lieu d'avoir une matrice diagonale D, on obtient une matrice diagonale par blocs B. Les valeurs extrêmes de B sont ainsi mitigées sur les blocs 2x2 de B.

Comme on peut constater sur la figure 3.5, les deux problèmes ont une évolution quasiment identique jusqu'à l'itération 37 et à partir de là, la version avec factorisation LBL^T arrive à converger mais pas celle avec factorisation LDL^T.

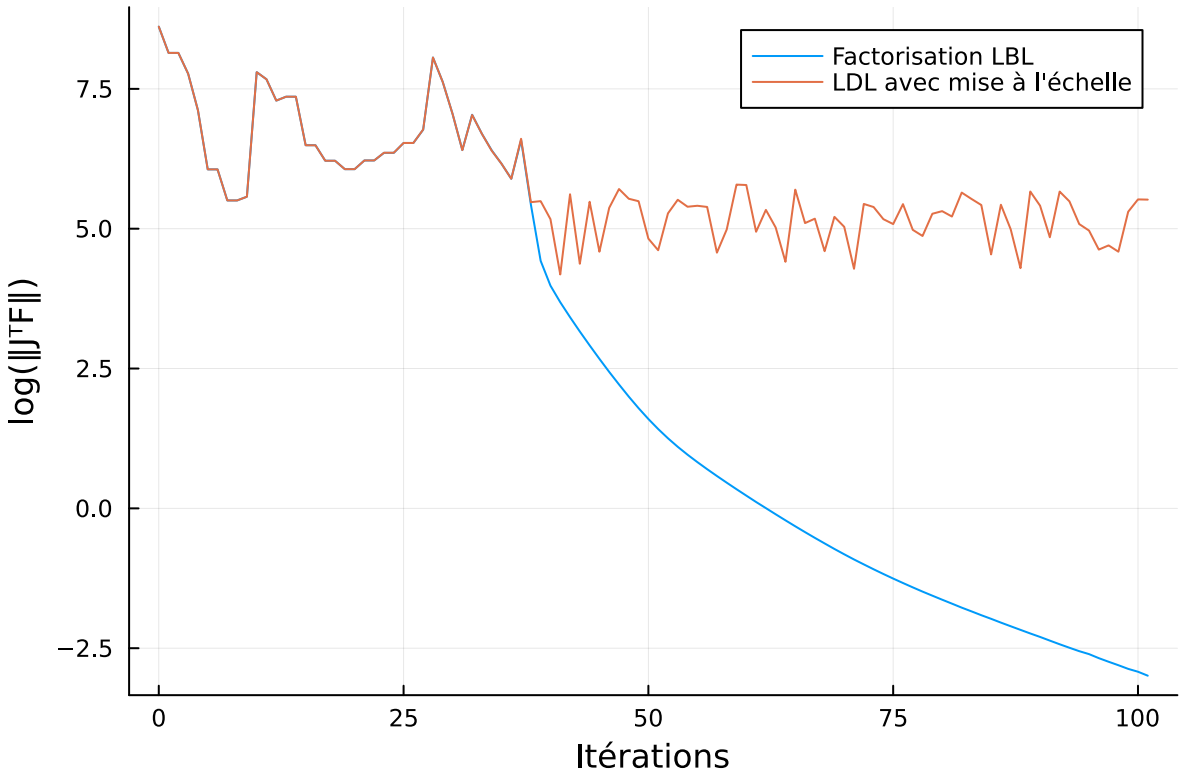

FIGURE 3.5 Évolution du log de la norme du gradient pour le problème 174-50489 avec une factorisation LDL^T et un scaling de Ruiz et une factorisation LBL^T.

Par contre, si on veut utiliser cette factorisation comme préconditionneur de MINRES il faut déterminer une façon de régulariser la matrice diagonale par blocs B pour que la matrice LBL^T soit définie positive. Nous avons déterminé 3 différentes façons d'effectuer cela. La première, de manière similaire à la factorisation LDL^T, consiste simplement à prendre la valeur absolue de la matrice diagonale par blocs :

$$P = L|B|L^T. \tag{3.21}$$

Les résultats numériques en effectuant cette régularisation sont trop mauvais, ce qui nous empêche de l'utiliser en préconditionneur de MINRES car le code ne fonctionne plus. La seconde technique consiste à additionner à la diagonale la valeur propre maximale de B.

$$P = L(B + \lambda_{max} I)L^T \tag{3.22}$$

La valeur propre maximale pour ces méthodes est trop élevée, le préconditionneur est donc

très mauvais pour MINRES, ce qui a tendance à faire diverger l'algorithme de Levenberg-Marquardt.

Algorithme 3.4.1 Procédure de régularisation de B

```
1:  B ∈ ℝ^{n×n}
2:  i = 1
3:  while i ≤ n faire
4:      si |B[i, i+1]| ≤ ϵ alors
5:          B[i, i] = |B[i, i]|
6:          i = i + 1
7:      sinon
8:          det = B[i, i] * B[i+1, i+1] − B[i, i+1]^2
9:          trace = B[i, i] + B[i+1, i+1]
10:         λ1 et λ2 solutions de −x^2 + x × trace − det
11:         si λ1 < 0 ou λ2 < 0 alors
12:             λ_max = max(|λ1|, |λ2|) + τ
13:             B[i, i] = B[i, i] + λ_max
14:             B[i+1, i+1] = B[i+1, i+1] + λ_max
15:         fin si
16:         i = i + 2
17:     fin si
18: fin while
```

La dernière technique est plus subtile pour éviter de perdre trop d'information et est décrite dans Algorithme 3.4.1. Cet algorithme va procéder en une disjonction des cas, comme B est bloc diagonale avec au maximum des blocs de taille 2. Si le bloc est de taille 1, on prend la valeur absolue du bloc comme on le voit à la ligne 5. Si le bloc est de taille 2 on calcule les 2 valeurs propre du bloc et on ajoute le maximum des 2 valeurs à la diagonale si une de ces valeurs est négative plus une valeur τ à la ligne 12. C'est la seule régularisation qui a permis d'obtenir des résultats avec MINRES.

En particulier, plus on choisit une valeur de τ petite, moins on va modifier l'information mais plus le système est proche d'être indéfini. Comme on le voit en pratique dans le tableau 3.3, les performances sont meilleures avec une valeur de $\tau \approx 1$.

Comme on a pu le voir précédemment, en résolvant le système augmenté avec MINRES et en le formant de manière explicite on permet l'utilisation de différents préconditionneurs simples à créer. Par contre, d'une part nous ne sommes pas assurés de trouver une direction de descente avec MINRES et surtout, on a besoin de stocker une matrice $A_\lambda \in \mathbb{R}^{(m+n)\times(m+n)}$. Cependant, on pourrait mitiger le 2e problème étant donné qu'une grande part des valeurs de A_λ sont nulles.

TABLEAU 3.3 Tableau présentant la première itération où $Pred < 0$ avec une factorisation LBL^T sur le problème 49-7776 avec 100 itérations au maximum.

τ	Itération ou $Pred < 0$
1e-5	50
1e-3	88
1e-1	aucune
1	aucune

3.5 TriMR et TriCG

Les méthodes TriMR et TriCG [48] ont été conçues spécialement pour résoudre des systèmes linéaires avec une matrice de la forme A_λ. Cependant, en théorie il est impossible de les utiliser pour résoudre le problème de la ligne 9 de Algorithme 3.1.1 car le second membre doit être de la forme suivante :

$$\begin{bmatrix} b \\ c \end{bmatrix}, \tag{3.23}$$

avec $b \in R^m$ et $c \in \mathbb{R}^n$, $b \neq 0$ et $c \neq 0$. Dans notre cas, $c = 0$ comme on peut le voir dans (3.17). Ce problème peut être résolu si on fournit à l'algorithme une solution initiale. En choisissant une valeur proche de 0 pour la valeur de d_k, on réussit à faire fonctionner l'algorithme mais les résultats ne sont pas très satisfaisants. Cependant, si on arrive à avoir une bonne estimation initiale de d_k à moindre coût, les performances pourraient être bien meilleures.

On constate dans la figure 3.6 que même si la norme du gradient diminue plus lentement en utilisant TriMR, l'évolution semble plus stable qu'avec LSMR.

Le grand avantage de TriMR est que cette méthode est optimale pour les problèmes de la forme du système augmenté. Ce qui en fait aussi son principal défaut étant donné que la marge d'amélioration est très faible, par exemple on ne peut pas donner de meilleur préconditionneur. Le deuxième inconvénient et qu'il faut fournir une valeur initiale de notre solution pour que la méthode fonctionne sur notre système. Cette estimation initiale pourrait être obtenue avec l'utilisation d'approximation rétrospective.

3.6 Précision mixte

Comme mentionné dans la revue de littérature, le langage Julia permet la compilation à la volée d'un code pour plusieurs précisions et un changement de précision de variables en cours

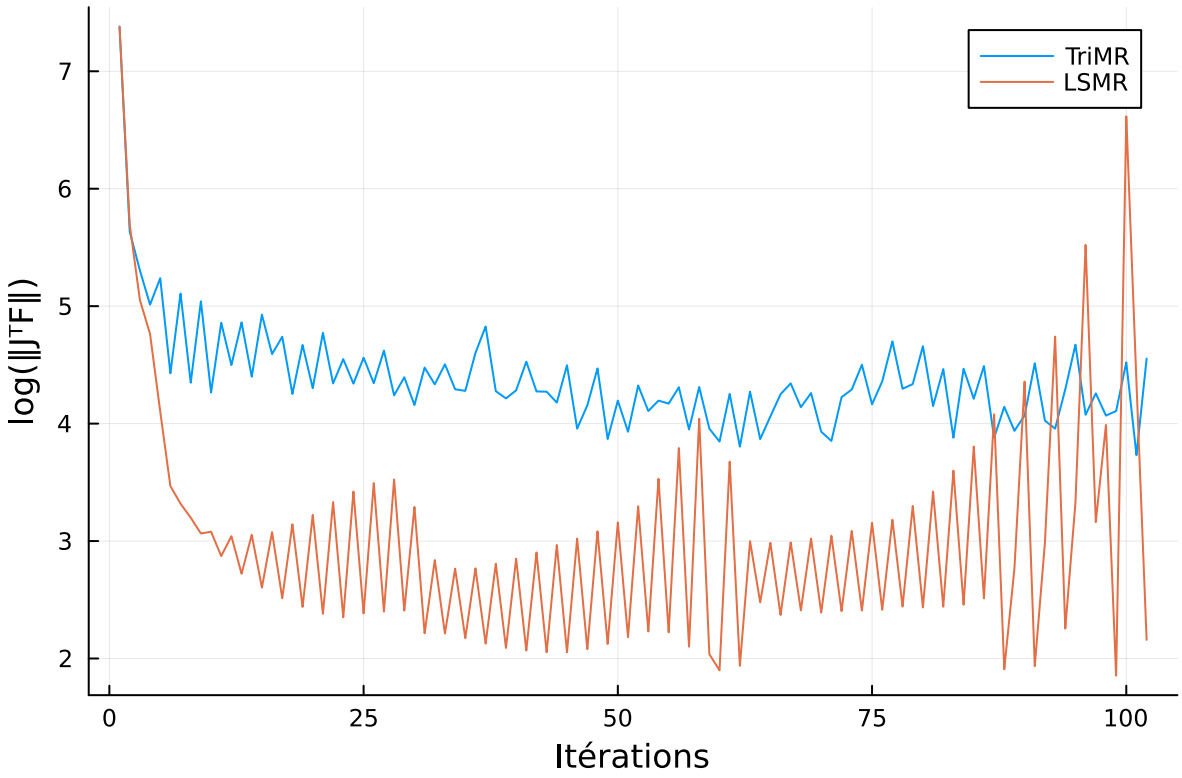

FIGURE 3.6 Évolution du log du gradient de LSMR et TriMR pour le problème 49-7776.

d'exécution. Partant de ce constat, la première idée naïve consiste à résoudre entièrement un problème des moindres carrés non linéaire en précision simple puis de le résoudre en précision double en utilisant comme point de départ la solution du problème précédent. Cependant, comme mentionné précédemment, si on utilise des méthodes itératives, on résout de manière incomplète un problème linéaire à chaque itération de Levenberg-Marquardt. On pourrait alors utiliser la précision multiple de manière plus fine et résoudre à chaque itération le problème en plus faible précision puis augmenter la qualité du résultat pour résoudre les dernières itérations dans une plus grande précision.

Pour effectuer cela on se base sur le principe du raffinement itératif [49]. Dans le cas d'un problème non régularisé on a

$$J(x_k)^T J(x_k) d_k^1 = -J(x_k)^T F(x_k). \tag{3.24}$$

À partir de d_k^1, on définit

$$r^1(x_k) := F(x_k) + J(x_k) d_k^1. \tag{3.25}$$

On forme à nouveau les équations normales mais cette fois ci on remplace $F(x_k)$ par $r^1(x_k)$

$$J(x_k)^T J(x_k) d_k^2 = -J(x_k)^T r^1(x_k). \tag{3.26}$$

On développant $r^1(x_k)$ dans (3.26) on obtient

$$J(x_k)^T J(x_k) d_k^2 = -J(x_k)^T F(x_k) - J(x_k)^T J(x_k) d_k^1, \tag{3.27}$$

et enfin

$$J(x_k)^T J(x_k)(d_k^1 + d_k^2) = -J(x_k)^T F(x_k), \tag{3.28}$$

avec $d_k = d_k^1 + d_k^2$.

Dans le cas d'un système régularisé le problème est

$$\min \ \tfrac{1}{2}(\|J(x_k) d_k^1 + F(x_k)\|^2 + \lambda_k^2 \|d_k^1\|^2) \tag{3.29}$$

qui a pour conditions d'optimalité les équations normales

$$(J(x_k)^T J(x_k) + \lambda_k^2 I) d_k^1 = -J(x_k)^T F(x_k). \tag{3.30}$$

On suite le même modèle que pour (3.25)

$$r_\lambda^1(x_k) = \begin{bmatrix} J(x_k) \\ \lambda I \end{bmatrix} d_k^1 + \begin{bmatrix} F(x_k) \\ 0 \end{bmatrix}. \tag{3.31}$$

On résout le nouveau système

$$(J(x_k)^T J(x_k) + \lambda^2 I) d_k^2 = -\begin{bmatrix} J(x_k)^T & \lambda I \end{bmatrix} r_\lambda^1(x_k) \tag{3.32}$$

et on obtient un résultat similaire à (3.28)

$$(J(x_k)^T J(x_k) + \lambda^2 I)(d_k^1 + d_k^2) = -J(x_k)^T F(x_k) \tag{3.33}$$

Le problème c'est qu'en formant (3.31), alors le système (3.32) constitue les équations normales du problème

$$\min \ \tfrac{1}{2} \left\| \begin{bmatrix} J \\ \lambda I \end{bmatrix} d_2 + \begin{bmatrix} r_1 \\ \lambda d_1 \end{bmatrix} \right\|^2 . \tag{3.34}$$

Or ce problème n'est pas un problème de moindre carrés linéaires régularisé. En pratique on va donc plutôt utiliser

$$d_k^1 = J(x_k)d_k^1 + F(x_k). \tag{3.35}$$

Les résultats provisoires obtenus ne semblent pas concluants car le second système (3.27) nécessite autant d'itérations pour être résolu que si on résolvait directement le sous-problème

$$\arg\min_{d \in \mathbb{R}^n} \tfrac{1}{2}\|J(x_k)d + F(x_k)\|^2 + \tfrac{1}{2}\lambda_k^2\|d\|^2. \tag{3.36}$$

Une autre façon d'utiliser la précision mixte est de calculer la factorisation LBL^T en simple précision et de l'utiliser en préconditionneur de MINRES en double précision. Comme les matrices jacobiennes des problèmes d'ajustement de faisceaux sont extrêmement mal conditionnées, il semblerait aussi judicieux d'effectuer un changement de précision mais cette fois ci de passer de double précision à quadruple. Comme le nombre d'itérations de MINRES lorsqu'on utilise la factorisation LBL^T en préconditionneur est très réduit, on pourrait économiser un temps précieux plutôt que de directement résoudre avec MINRES en quadruple précision.

La figure 3.7 présente le temps nécessaire pour chaque itération de LSMR et LSMR en multi-précision avec un raffinement itératif. On constate que le raffinement itératif n'offre pas a priori de meilleures performances. En pratique, ça se traduit par le fait que le système (3.26) nécessite approximativement autant d'itérations que si on résolvait le sous-problème de la ligne 9 de Algorithme 3.1.1.

3.6.1 Choix des critères d'arrêt

Une des difficultés du travail en précision mixte est le choix de critères d'arrêt entre la première version en faible précision et la deuxième en meilleure précision. Dans le cas des méthodes itératives, pour le sous-problème de l'algorithme de Levenberg-Marquardt, le raisonnement par défaut consiste à appliquer les mêmes critères d'arrêt peu importe la précision mais ces critères sont relatifs à l'epsilon machine de la précision utilisée.

Par exemple, une tolérance d'arrêt sur la norme de du résidu pour MINRES que l'on établit

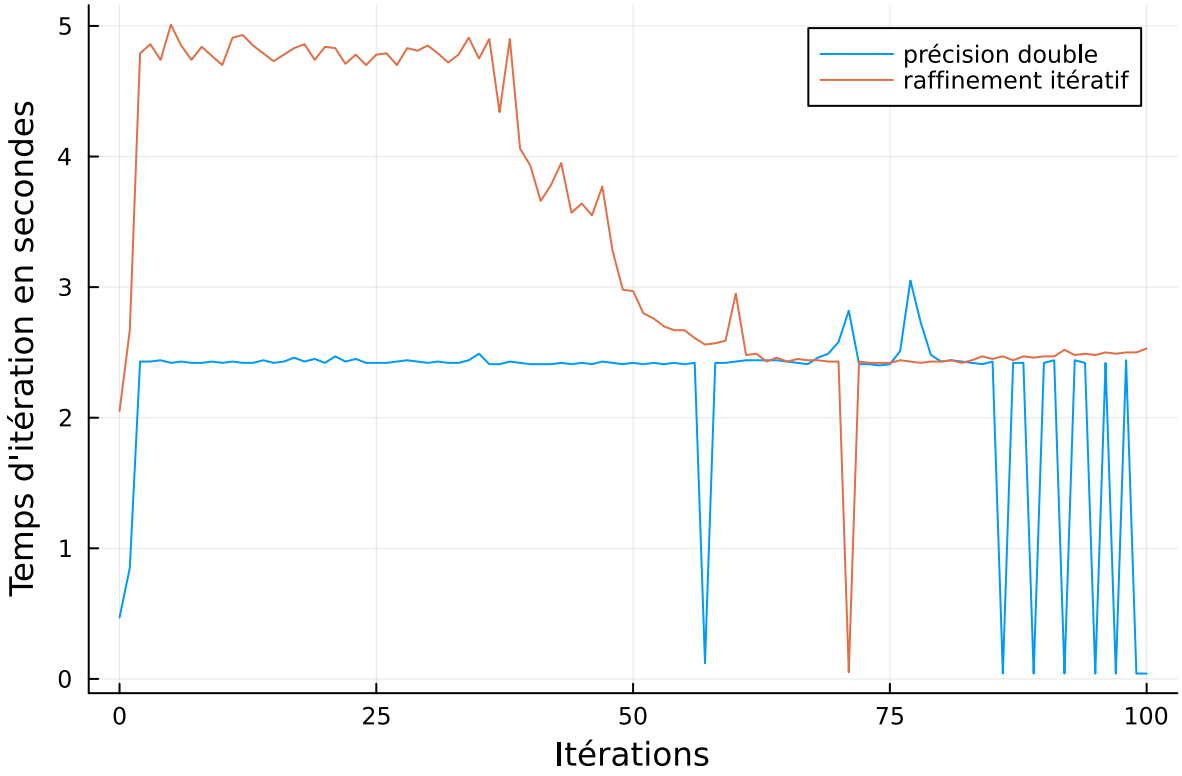

FIGURE 3.7 Temps par itération avec LSMR en Float64 et LSMR en Float32 et Float64 avec raffinement itératif (PC).

à $\sqrt{\epsilon}$ sera de 3.45×10^{-4} en Float32 et 1.49×10^{-8} en Float64. Il n'y a pas d'autres critères d'arrêt qui ont donné de meilleurs résultats jusqu'à présent. Une autre méthode dont le fonctionnement est n'est pas trop éloigné mais qui ne pose pas les mêmes problèmes de critères et l'approximation rétrospective [50] et on pourrait approfondir cette approche pour obtenir de meilleurs résultats qu'avec la précision mixte.

3.7 GPU

Comme mentionné dans la revue de littérature, l'utilisation de GPU permet dans certains cas d'améliorer les performances d'un même code à cause des différences d'architecture par rapport à un CPU. Dans le cas de l'optimisation numérique, les méthodes directes ne peuvent pas, par construction, être portées sur GPU car elle nécessitent l'indexation de vecteurs ce qui ralentit très largement la résolution sur GPU. Les méthodes itératives peuvent quant à elles être portées sur GPU. De plus, dans leur implémentation actuelle, l'évaluation du résidu et de la jacobienne des problèmes d'ajustement de faisceaux ne peuvent pas être faits sur

GPU. J'ai donc implémenté un code qui évalue une partie du code sur CPU et qui résout le problème sur GPU.

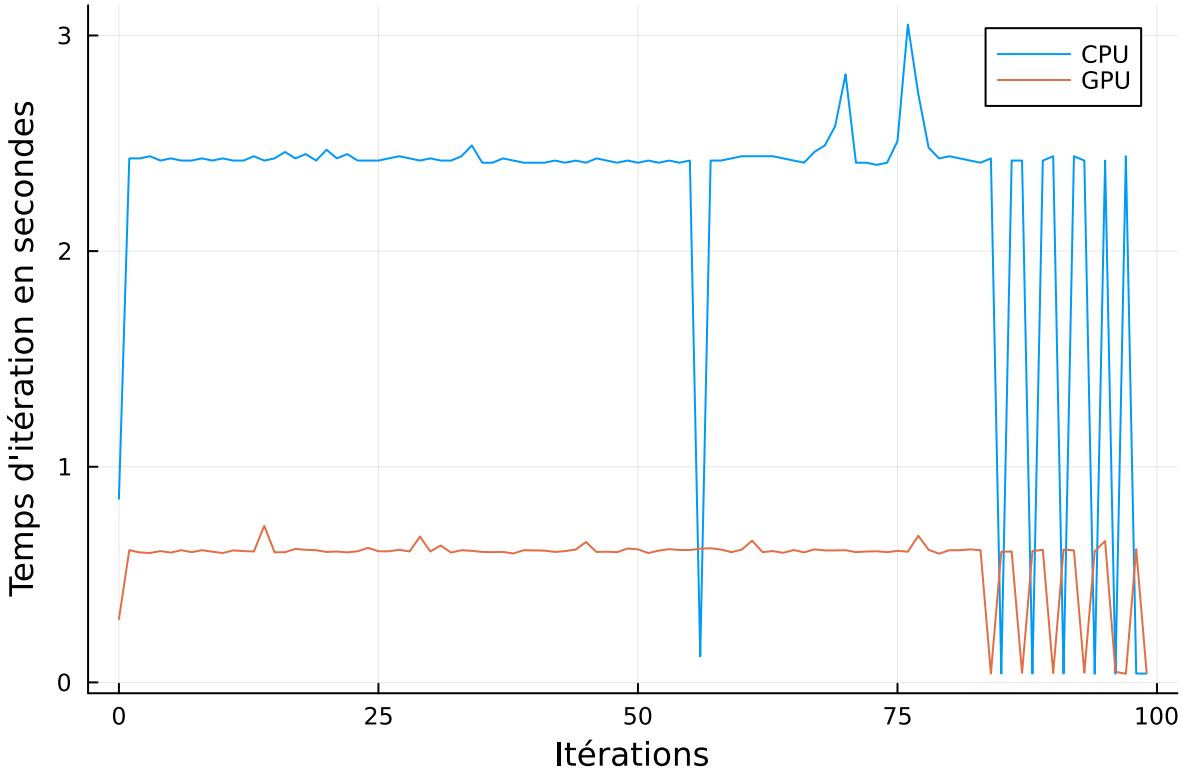

FIGURE 3.8 Temps par itération avec LSMR sur GPU et LSMR sur CPU avec les mêmes critères d'arrêt (PC).

D'après la figure 3.8, on constate qu'avec des critères d'arrêt similaires, chaque itération est environ 3.6 fois plus rapide sur GPU que sur CPU.

La figure 3.9 est d'autant plus intéressante car on constate qu'en résolvant avec le raffinement itératif en multi-précision certaines itérations vont jusqu'à être environ 5 fois plus rapide. Cela est dû au fait que les GPU ont une architecture qui supporte beaucoup mieux les différents précisions par rapport au CPU. Ce qui signifie que si on réussit à tirer du potentiel de la précision mixte, alors ce dernier sera décuplé si on peut exécuter le code sur GPU.

3.8 Différentiation automatique

Lors d'un projet de stage en été 2020, Célestine Angla a codé une fonction résidu pour les problèmes d'ajustement de faisceaux. Cette fonction permet, avec un fichier de données pour chaque problème, de résoudre des problèmes réels d'ajustement de faisceaux. De plus

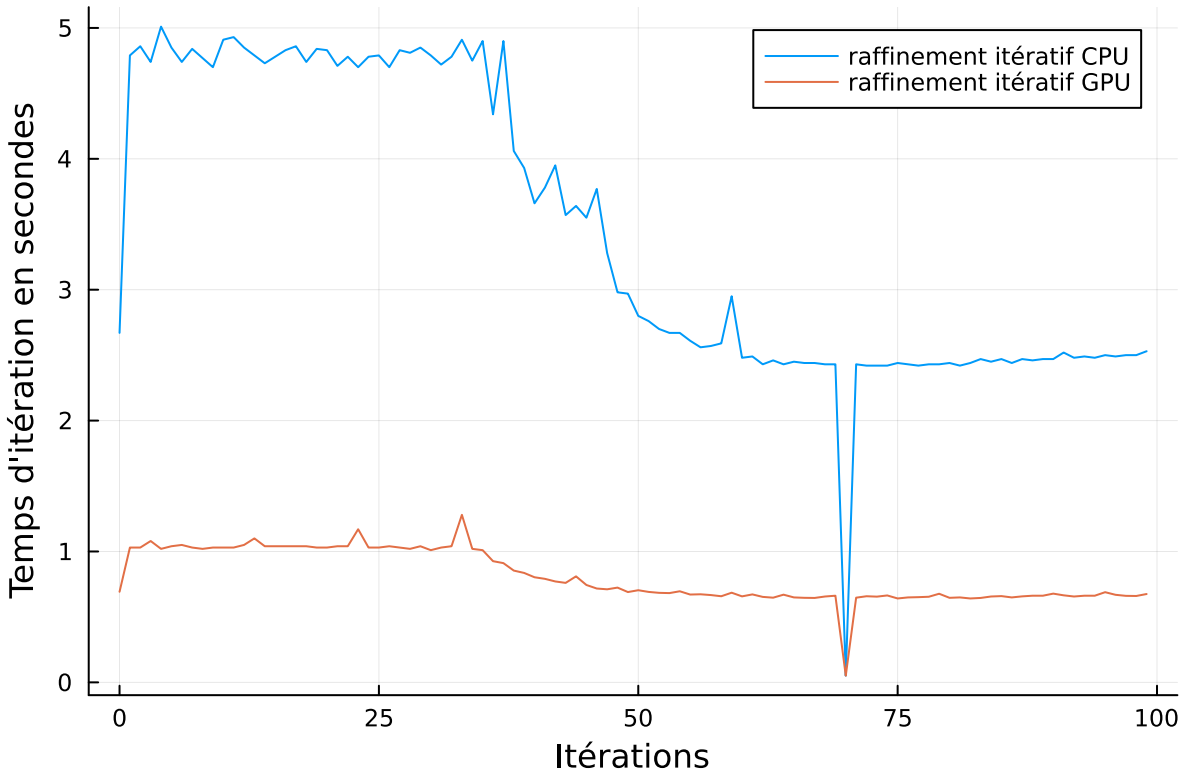

FIGURE 3.9 Temps par itération avec LSMR en Float32 et Float64 avec raffinement itératif sur CPU et GPU avec les mêmes critères d'arrêt (PC).

il est aussi nécessaire de déterminer la jacobienne du problème. Il s'avère qu'elle a aussi été implémentée à la main dans le code du stage. Cependant, calculer cette matrice à la main requiert beaucoup de temps et doit être fait pour chaque problème différent.

Une des options est donc d'utiliser la différentiation automatique pour déterminer l'opérateur linéaire de la jacobienne [51]. Grâce à cela nous ne formons jamais la jacobienne et nous pouvons appliquer notre algorithme à d'autres problèmes où nous ne connaissons pas la jacobienne.

De plus, comme nous avons un détail exact du calcul de la jacobienne, nous pourrons comparer les résultats avec et sans différentiation automatique pour constater l'efficacité de la différentiation automatique dans ce cas précis.

Nous avons effectué des tests sur la différentiation automatique qui ont mis en évidence 2 éléments majeurs visibles dans le tableau 3.4. Tout d'abord, le produit Jb, avec $b \in \mathbb{R}^n$ un vecteur quelconque, est plus rapide que le produit $J^T c$ avec $c \in \mathbb{R}^m$ un vecteur quelconque aussi. De plus, il semblerait que l'utilisation de ReverseDiff soit plus efficace pour un produit $J^T c$. D'après la documentation et le code des librairies ForwardDiff et ReverseDiff, il

Tableau 3.4 Performances de la différentiation automatique pour le problem-49-7776.

Calcul effectué	Jb	$J^T c$	$J^T c$
Méthode utilisée	ForwardDiff	ForwardDiff	ReverseDiff
Temps (s)	1.79	321	5.01
Allocations (GB)	0.376	300	0.732

semblerait que ForwardDiff est plus performant pour les matrices rectangulaires $J \in \mathbb{R}^{m \times n}$ telles que $m > n$ à l'opposé ReverseDiff semble plus performant pour les matrices telles que $n > m$. Comme pour tous les problèmes d'ajustement de faisceaux $J \in \mathbb{R}^{m \times n}$ avec $m > n$, on utilise ForwardDiff pour le produit Jb et ReverseDiff pour $J^t c$.

Enfin, comme on a pu le voir dans la revue de littérature dans la partie 2.7, la fonction résidu et jacobiennes sont très structurées. Il semble donc possible de différentier chaque bloc indépendamment pour gagner encore en performances.

3.8.1 Limitations de la différentiation automatique

Il est théoriquement possible d'utiliser la différentiation automatique sur GPU mais l'implémentation actuelle de la fonction résidu nécessite l'indexation de vecteurs ce qui diminue grandement les performances du code. De plus, l'utilisation de la librairie ReverseDiff nécessite aussi de l'indexation ce qui signifie que les librairies liées à la différentiation automatique sur Julia doivent encore être améliorées avant de pouvoir accéder à ces avantages.

Il y a encore énormément d'allocations lors de l'utilisation de différentiation automatique pour calculer le sous-problème de Levenberg-Marquardt et ce même en fournissant une fonction résidu sans allocations. Une amélioration de la différentiation automatique dans Julia aurait donc un impact direct sur les performances du code.

3.9 Conditions d'arrêt et paramètres

Les principales conditions d'arrêt dans le code sont les mêmes que celles déterminées dans la partie 2.3.1

$$\|J(x_k)^T F(x_k)\| \leq atolg + rtolg \times \|J(x_0)^T F(x_0)\|, \tag{3.37}$$

$$f(x_k) \leq atolf + rtolf \times f(x_0). \tag{3.38}$$

Les valeurs de $atolg$, $rtolg$, $atolf$ et $rtolf$ sont relatives à l'epsilon machine de la précision utilisée. On ajoute ensuite un dernier critère pour éviter un trop grand nombre d'évaluations du résidu, de produits jacobien-vecteur, d'itérations ou un temps trop grand au cas où le problème ne converge pas.

Concernant les critères d'arrêt du sous-problème, comme ils étaient aussi difficiles à déterminer j'ai directement demandé aux développeurs de Ceres le raisonnement derrière le choix de ces derniers. Le premier critère d'arrêt est celui basé sur [11] afin d'assurer la convergence de l'algorithme qui est donné dans l'équation 3.39

$$\|J(x_k)d + F(x_k)\| \leq \eta_k \|J(x_k)^T F(x_k)\|. \tag{3.39}$$

Cependant, comme dans le cas de Ceres le gradient conjugué est appliqué à S comme défini dans l'équation 3.5. Or, S est définie positive ce qui permet l'introduction du critère d'arrêt suivant :

$$\frac{k[Q(x_k) - Q(x_{k-1})]}{Q(x_k)} \leq \eta_k, \tag{3.40}$$

avec $Q(x) = x^T Sx - 2u^T x$ avec u provenant de l'équation 3.6. Ce critère n'est cependant pas utilisable si on utilise LSMR, LSQR ou MINRES.

Concernant les différents paramètres η_1, η_2, σ_1, σ_2 et λ_0, malgré plusieurs tentatives d'améliorations, les meilleurs valeurs semblent spécifiques à chaque problème dans le cadre des problèmes d'ajustement de faisceaux. En plus, comme les problèmes d'ajustement de faisceaux demandent beaucoup de ressources pour être résolus, cela rend l'optimisation encore plus difficile.

3.10 Implémentation de l'algorithme

3.10.1 Programmation en Julia

Le développement de l'algorithme en Julia a permis d'exploiter plusieurs techniques propres au langage et principalement le multiple dispatch. Grâce à cela, j'ai conçu l'algorithme de sorte qu'il soit très facile pour n'importe qui de calculer le jacobien et/ou résoudre le problème avec sa propre méthode tant que les fonctions respectent la signature donnée. Il s'agit d'une structure par blocs qui permet d'emboîter très facilement un morceau de code extérieur. Par exemple, pour Algorithme 3.1.1, il y a juste besoin de remplacer la ligne 9 par (3.15). Cela permet une comparaison des différentes versions très fidèle.

L'utilisation de solvers pour préallouer la mémoire nécessaire dans le code diminue les allocations à la volée et améliore les performances tout comme l'utilisation des opérateurs linéaires qui évitent de former les produits matrice-matrice. Enfin, la parallélisation du code qui peut être d'autant plus exploitée sur GPU a permis d'améliorer encore les performances de l'algorithme.

3.10.2 Test du code

Comme les problèmes d'ajustement demandent beaucoup de ressources, je n'ai pas pu les tester sur mon ordinateur personnel. J'ai donc utilisé des machines à distance pour lancer le code et ai dû m'adapter à plusieurs contraintes comme le partitionnement des problèmes sur différents ordinateurs, la redirection des logs dans différents fichiers et la centralisation des scripts vers un seul fichier source pour éviter la perte de temps à lancer le code sur différents ordinateurs un par un.

3.10.3 Bibliothèque de problèmes d'ajustement de faisceaux

Dans la revue de littérature, nous faisons mention des problèmes d'ajustement de faisceaux dont une bibliothèque publique est disponible en ligne [1]. Cependant, les problèmes sont donnés dans un fichier en respectant un format donné dans l'article. Le travail de stage de Célestine Angla a permis d'apprendre à lire et créer une première version de problèmes aux moindres carrés non linéaires à partir de ces fichiers. Nous avons ensuite récupéré ce travail pour en créer une librairie publiée dans Julia. C'est cette bibliothèque de problèmes qui a posé la base de mes tests de performances des différents algorithmes.

Tableau 3.5 Allocations des fonctions pour le problème 1778-993923.

Function	Allocations	Memory
residual!	202747249	7.42 GB
jac_structure!	225330279	4.40 GB
jac_coord!	287779988	12.11 GB

Comme on peut le voir dans le tableau 3.5, la fonction *residual!* qui correspond à $F(x)$ et *jac_structure!* et *jac_coord!* qui correspondent à la création de $J(x)$ allouaient une grande quantité de mémoire en particulier pour les gros problèmes d'ajustement de faisceaux. Nous avons réduit ces allocations jusqu'à 0 en préallouant la mémoire directement dans le modèle du problème.

Comme on peut le voir dans le tableau 3.6, cette drastique réduction d'allocations a permis

TABLEAU 3.6 Facteur de temps gagné pour l'évaluation des fonctions pour différents problèmes.

Problème	Fonction residual!	Fonction jac_structure!	Fonction jac_coord!
49-7776 ladybug	7,73	15,17	5,52
174-50489 trafalgar	8,00	15,75	5,56
273-176305 dubrovnik	9,90	18,21	6,49
1778-993923 venice	7,80	14,83	5,41

de diminuer d'un très grand facteur le temps d'évaluation de $F(x)$ et $J(x)$. L'évaluation de la fonction résidu est environ 8 fois plus rapide et la mise à jour de la jacobienne est plus de 5 fois plus rapide.

3.11 Comparaison des résultats avec Ceres

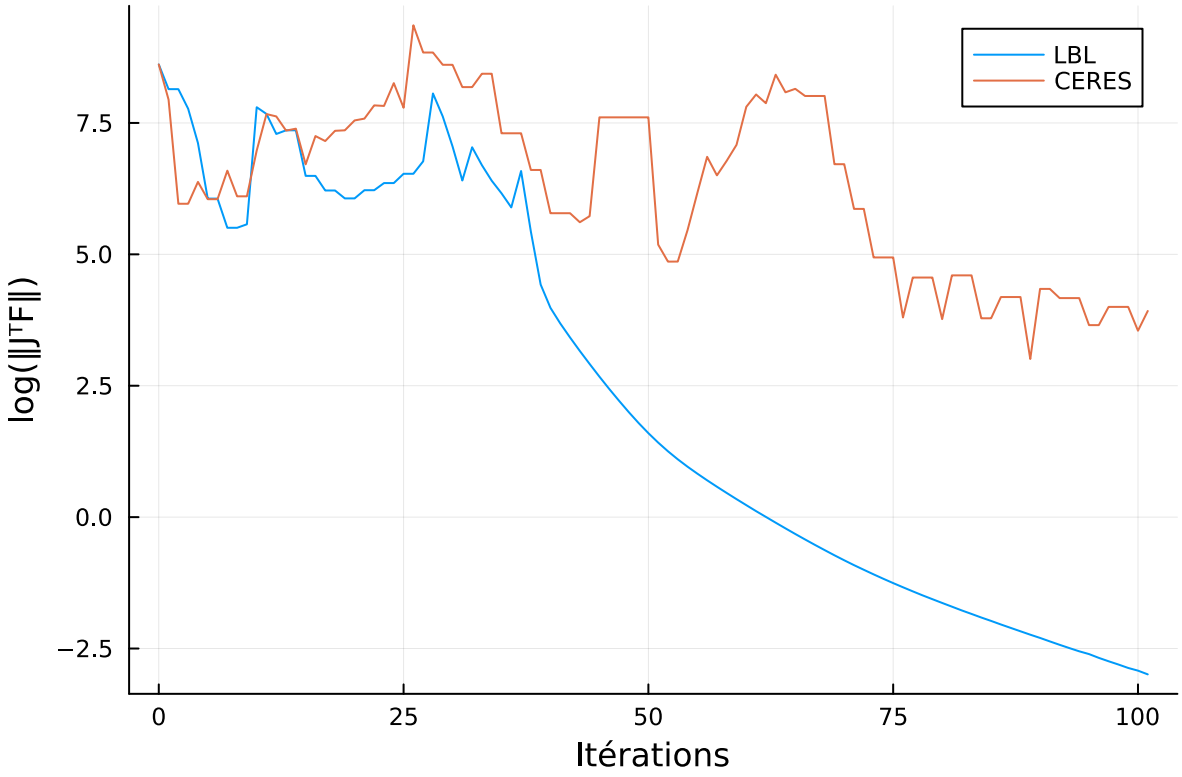

FIGURE 3.10 Évolution du log de la norme du gradient pour le problème 174-50489 avec une factorisation LBL^T et avec Ceres.

Sur le problème de la figure 3.10 ainsi que sur quelques autres problèmes, on arrive à avoir une convergence plus rapide en temps et en norme du gradient avec notre code en Julia

qu'avec Ceres. Cependant, nous utilisons entre autre un calcul de la jacobienne manuel et optimisé ce qui permet de gagner un temps non négligeable.

3.12 Affichage des résultats

Le vecteur de variables u défini dans l'équation 2.50 de la revue de littérature contient des coordonnées en 3 dimensions de points. Ces mêmes points réorganisés dans le bon format permettent de reproduire les problèmes donnés en 3 dimensions comme dans la figure 1.1 qui correspond dans ce cas au vecteur initial du problème 1778-993923 de la catégorie Venice.

FIGURE 3.11 Fontaine d'Onofrio dans la vieille ville de dubrovnik

La figure 3.11 est une photo de la fontaine d'Onofrio. On peut y voir la fontaine dans une place ainsi que les murs des maisons qui l'entourent. Nous observons ensuite la modélisation initiale et après résolution de cette fontaine qui correspond au problème 16-22106 de la librairie BAL.

On aperçoit une partie des structures sur la modélisation initiale dans la figure 3.12, mais

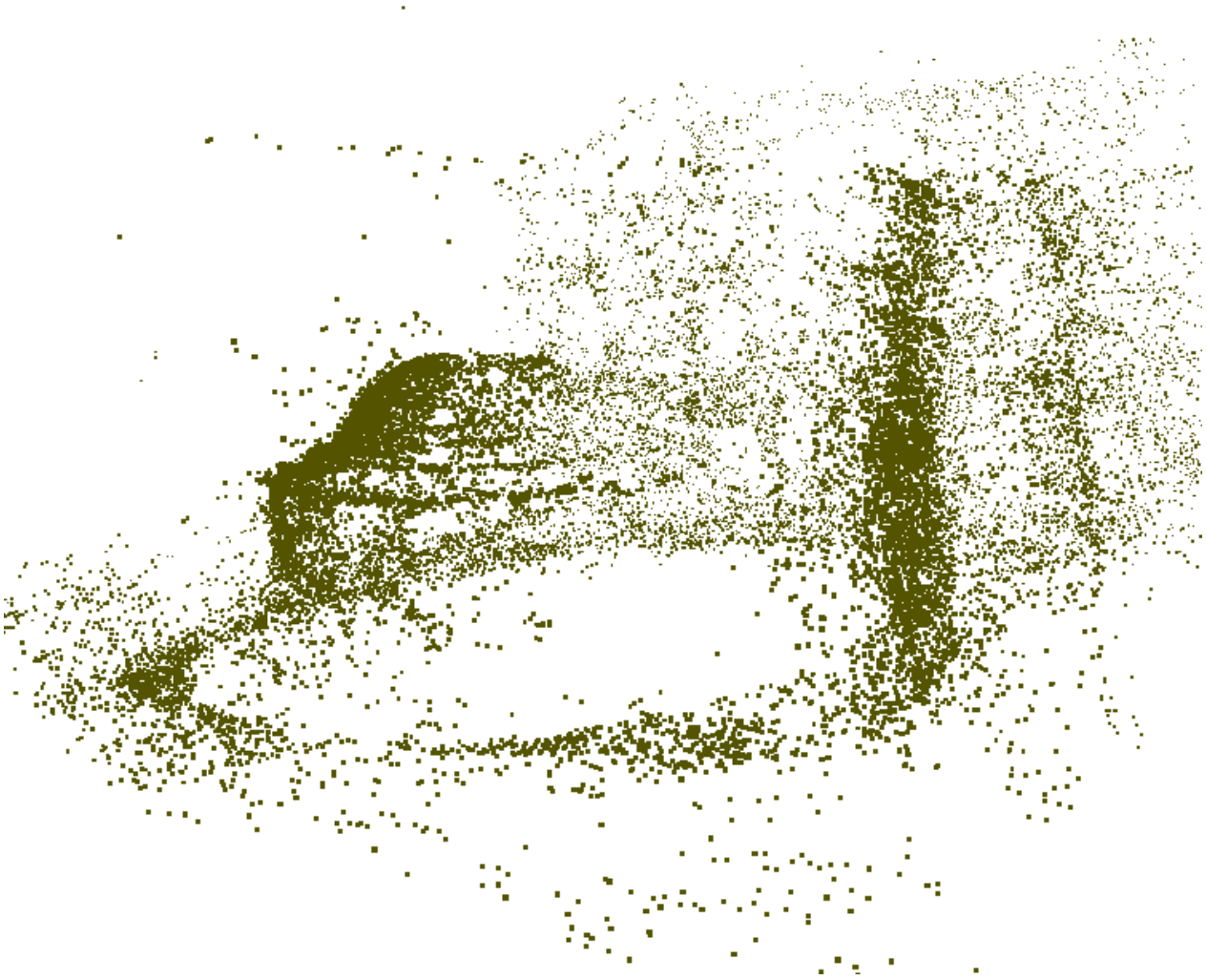

FIGURE 3.12 Modélisation initiale du problème 16-22106 représentant la fontaine d'Onofrio.

elle comporte plusieurs défauts. Les structures ne sont pas nettes et il y a plusieurs points résiduels partout sur l'image.

Une fois que le problème est résolu, comme on peut le voir dans la figure 3.13, on aperçoit que les structures sont beaucoup plus nettes. Par exemple les murs à droite sont droits, on aperçoit l'encadrement des fenêtres et beaucoup moins de points résiduels partout sur l'image.

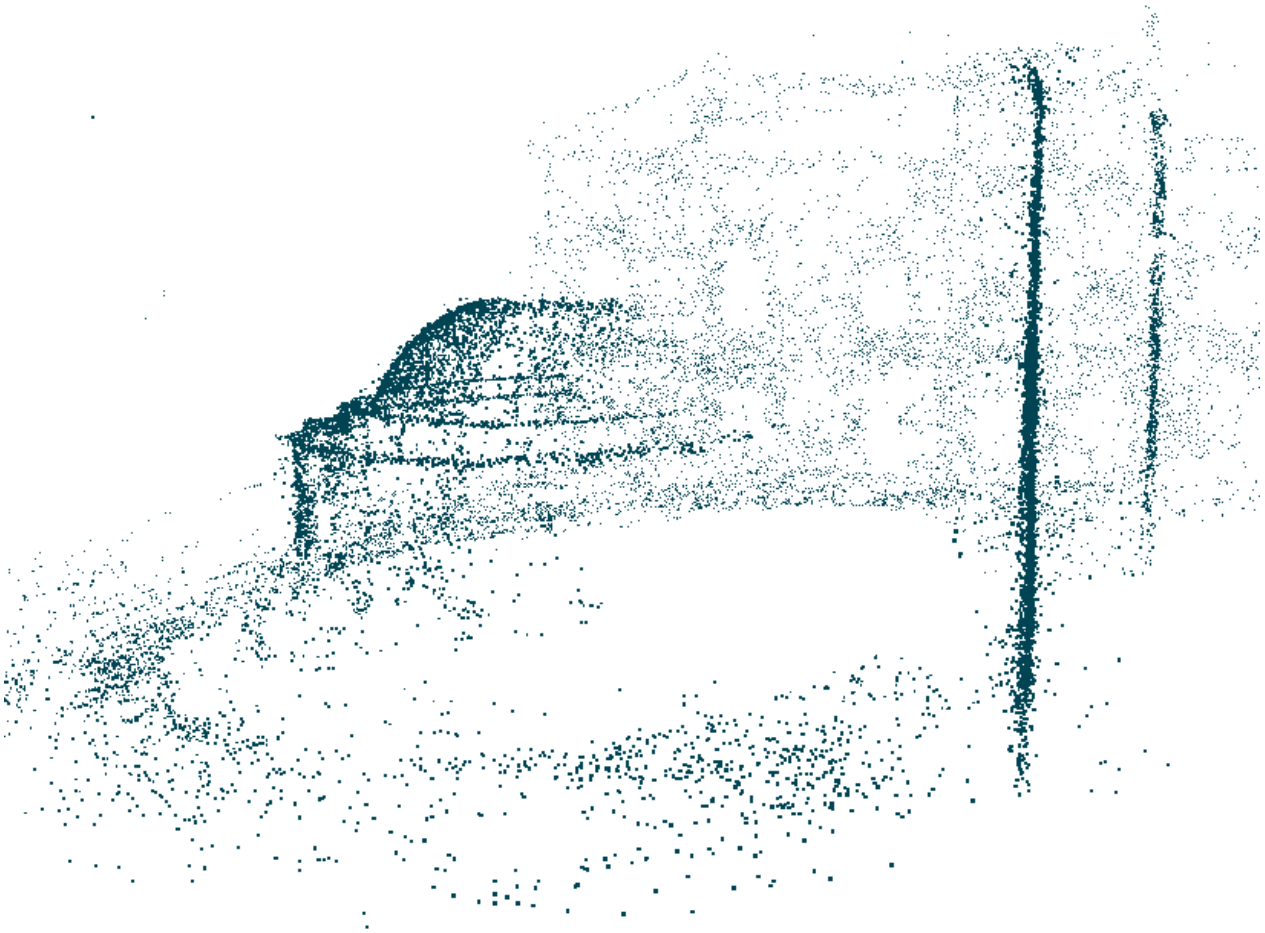

FIGURE 3.13 Modélisation finale du problème 16-22106 représentant la fontaine d'Onofrio.

CHAPITRE 4 CONCLUSION ET RECOMMANDATIONS

4.1 Synthèse des travaux

Dans ce mémoire nous avons abordé différentes alternatives pratiques à celles déjà implémentées pour la résolution de problèmes aux moindres carrés non linéaires. En particulier nous avons montré les limitations de l'utilisation des méthodes directes et CG sur le complément de Schur dans la partie 3.2 pour les problèmes mal conditionnés. En effet, nous avons démontré que ces différentes méthodes qui forment explicitement la matrice hessienne H_λ ont un très mauvais conditionnement. A partir de ces résultats nous avons proposé plusieurs alternatives qui ne calculent pas H_λ. La première, qui consiste à utiliser LSMR pour résoudre le sous-problème, s'est rapidement avérée difficile à améliorer. Ensuite, la reformulation du problème pour utiliser MINRES a permis l'exploration et l'amélioration d'un préconditionneur LDL^T, puis LBL^T. Nous avons aussi constaté que la résolution sur GPU, qui est actuellement adéquate uniquement sur les méthodes itératives, améliore grandement les performances. Chaque itération est plus de 3 fois plus rapide si on utilise la résolution sur GPU au lieu de CPU avec un résultat identique à la précision machine. De plus, si on arrive à avoir implémenter une version efficace du raffinement itératif, l'utilisation de GPU améliorera encore ces résultats, étant donné les meilleurs performances du GPU en précision simple. Enfin, nous avons réussi à implémenter une version de l'algorithme avec une résolution du sous problème itérative et une factorisation LBL^T en préconditionneur.

4.2 Développement de librairies Julia

Nous avons développé 2 librairies en Julia. La première, LevenbergMarquardt.jl, est utile pour résoudre les problèmes de moindres carrés non linéaires avec l'algorithme de Levenberg-Marquardt. Cette librairie s'intègre avec les autres modules de l'écosystème de JuliaSmoothOptimizers. En particulier, elle fonctionne avec n'importe quel NLPModel. Elle supporte différentes précisions, intègre la différentiation automatique et les calculs sur GPU. La résolution du sous-problème peut-être faite avec différentes méthodes itératives : LSMR, CG, TriMR, TriCG, MINRES, et méthodes directes : LDL^T, LDL^T avec complément de Schur et LBL^T. De plus, elle a été conçue en gardant une structure principale de l'algorithme inchangée de sorte à pouvoir facilement ajouter d'autres méthodes de résolution du sous-problème.

La deuxième librairie, BundleAdjustmentModels.jl, est une banque de problèmes d'ajustement de faisceaux. Une partie du code a été développée par Célestine Angla lors d'un

stage précédent cette maîtrise. Il s'agit d'un dépôt Julia accessible en open-source. Les tests couvrent 92% du code et le code a été développé avec l'intégration continue, pour pouvoir rester d'actualité avec les prochaines versions de Julia. On y utilise la technologie des artéfacts Julia pour une meilleure gestion des fichiers. La fonction résidu et jacobienne n'allouent pas de mémoire. Enfin, on peut générer des fichiers pour observer l'évolution du résultat en 3 dimensions.

4.3 Limitations de la solution proposée

Comme nous l'avons montré dans le tableau 3.2, les différents alternatives proposées à Ceres sont mieux conditionnées, cependant la taille du système augmenté (3.15) est bien plus importante que celle du système basé sur le complément de Schur (3.4) ce qui influe fortement sur les performances théoriques de l'algorithme et que l'on aperçoit en pratique. De plus, les résultats provisoires obtenus avec l'utilisation de précision mixte sont en deçà de nos espérances.

4.4 Améliorations futures

Comme nous l'avons mentionné ci-dessus, les résultats obtenus avec l'utilisation de précision mixte sont décevants par rapport à ce qu'on pourrait en attendre d'après la littérature. L'utilisation de calculs sur GPU semble cependant très prometteuse et en particulier si elle réussit à être couplée de manière efficace avec un approche en précision mixte. De plus nous avons peu approfondi l'utilisation des différentes méthodes par manque de temps, nous pourrions par exemple chercher de meilleurs préconditionneurs pour LSMR, d'autres façons de donner une première estimation de pas pour TriMR, ou une meilleure façon d'effectuer le raffinement itératif. Le conditionnement très mauvais des problèmes semble aussi indiquer que nous pourrions résoudre le problème en précision quadruple avec une méthode directe en précision double en préconditionneur. On pourrait aussi améliorer le raffinement itératif en utilisant des méthodes itératives flexibles comme FGMRES. Le choix des critères d'arrêt pour le changement de précision du raffinement itératif est aussi un problème difficile à résoudre. Mis à part FGMRES, il existe encore d'autres méthodes de Krylov qui pourraient être intéressantes. On pourrait même tester des méthodes avec une approche stochastique. En particulier, l'approximation rétrospective est une piste intéressante. On pourrait aussi intégrer les améliorations que nous avons abordé directement avec les développeurs de Ceres.

RÉFÉRENCES

[1] S. Agarwal, N. Snavely, S. M. Seitz et R. Szeliski, “Bundle adjustment in the large,” dans *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos et N. Paragios, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010, p. 29–42.

[2] N. Snavely, S. M. Seitz et R. Szeliski, “Photo tourism : Exploring photo collections in 3d,” dans *SIGGRAPH Conference Proceedings*. New York, NY, USA : ACM Press, 2006, p. 835–846.

[3] ——, “Modeling the world from "internet" photo collections,” *International Journal of Computer Vision*, vol. 80, n°. 2, p. 189–210, November 2008. [En ligne]. Disponible : http://phototour.cs.washington.edu/

[4] B. Triggs, P. F. McLauchlan, R. I. Hartley et A. W. Fitzgibbon, “Bundle adjustment — a modern synthesis,” dans *Vision Algorithms : Theory and Practice*, B. Triggs, A. Zisserman et R. Szeliski, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2000, p. 298–372.

[5] H. Cao, P. Tao, H. Li et J. Shi, “Bundle adjustment of satellite images based on an equivalent geometric sensor model with digital elevation model,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 156, p. 169–183, 2019. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0924271619301959

[6] H. Pan, T. Huang, P. Zhou et Z. Cui, “Self-calibration dense bundle adjustment of multi-view worldview-3 basic images,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 176, p. 127–138, 2021. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0924271621001143

[7] Y.-S. Shin, Y. Lee, H.-T. Choi et A. Kim, “Bundle adjustment from sonar images and slam application for seafloor mapping,” dans *OCEANS 2015 - MTS/IEEE Washington*, 2015, p. 1–6.

[8] K. Levenberg, “A method for the solution of certain non-linear problems in least squares,” *Quart. Appl. Math.*, p. 164–168, 1944. [En ligne]. Disponible : https://doi.org/10.1090/qam/10666

[9] D. W. Marquardt, “An algorithm for least-squares estimation of nonlinear parameters,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, n°. 2, p. 431–441, 1963. [En ligne]. Disponible : https://doi.org/10.1137/0111030

[10] K. L. Hiebert, “An evaluation of mathematical software that solves nonlinear least squares problems,” *ACM Trans. Math. Softw.*, vol. 7, n°. 1, p. 1–16, mar 1981. [En ligne]. Disponible : https://doi.org/10.1145/355934.355935

[11] S. J. Wright et J. N. Holt, “An inexact levenberg-marquardt method for large sparse nonlinear least squres,” *The Journal of the Australian Mathematical Society. Series B. Applied Mathematics*, vol. 26, n°. 4, p. 387–403, 1985.

[12] J. E. Dennis, D. M. Gay et R. E. Walsh, “An adaptive nonlinear least-squares algorithm,” *ACM Trans. Math. Softw.*, vol. 7, n°. 3, p. 348–368, sep 1981. [En ligne]. Disponible : https://doi.org/10.1145/355958.355965

[13] M. R. Osborne, “Nonlinear least squares — the levenberg algorithm revisited,” *The Journal of the Australian Mathematical Society. Series B. Applied Mathematics*, vol. 19, n°. 3, p. 343–357, 1976.

[14] A. S. Householder, “Unitary triangularization of a nonsymmetric matrix,” *J. ACM*, vol. 5, n°. 4, p. 339–342, oct 1958. [En ligne]. Disponible : https://doi.org/10.1145/320941.320947

[15] W. Givens, “Computation of plain unitary rotations transforming a general matrix to triangular form,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 6, n°. 1, p. 26–50, 1958. [En ligne]. Disponible : https://doi.org/10.1137/0106004

[16] T. A. Davis, “Algorithm 849 : A concise sparse cholesky factorization package,” *ACM Trans. Math. Softw.*, vol. 31, n°. 4, p. 587–591, dec 2005. [En ligne]. Disponible : https://doi.org/10.1145/1114268.1114277

[17] C. Brezinski, *Lanczos Tridiagonalization Process*. Boston, MA : Springer US, 2002, p. 161–169. [En ligne]. Disponible : https://doi.org/10.1007/978-1-4613-0261-2_7

[18] G. Golub et W. Kahan, “Calculating the singular values and pseudo-inverse of a matrix,” *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis*, vol. 2, n°. 2, p. 205–224, 1965. [En ligne]. Disponible : https://doi.org/10.1137/0702016

[19] M. A. Saunders, H. D. Simon et E. L. Yip, “Two conjugate-gradient-type methods for unsymmetric linear equations,” *SIAM Journal on Numerical Analysis*, vol. 25, n°. 4, p. 927–940, 1988. [En ligne]. Disponible : https://doi.org/10.1137/0725052

[20] M. R. Hestenes et E. Stiefel, “Methods of conjugate gradients for solving linear systems,” *Journal of research of the National Bureau of Standards*, vol. 49, p. 409–435, 1952.

[21] C. C. Paige et M. A. Saunders, “Solution of sparse indefinite systems of linear equations,” *SIAM Journal on Numerical Analysis*, vol. 12, n°. 4, p. 617–629, 1975. [En ligne]. Disponible : https://doi.org/10.1137/0712047

[22] A. Montoison et D. Orban, “Tricg and trimr : Two iterative methods for symmetric quasi-definite systems,” *SIAM Journal on Scientific Computing*, vol. 43, n°. 4, p. A2502–A2525, 2021. [En ligne]. Disponible : https://doi.org/10.1137/20M1363030

[23] C. C. Paige et M. A. Saunders, “Lsqr : An algorithm for sparse linear equations and sparse least squares,” *ACM Trans. Math. Softw.*, vol. 8, n°. 1, p. 43–71, mars 1982. [En ligne]. Disponible : https://doi.org/10.1145/355984.355989

[24] D. C.-L. Fong et M. Saunders, “Lsmr : An iterative algorithm for sparse least-squares problems,” *SIAM Journal on Scientific Computing*, vol. 33, n°. 5, p. 2950–2971, 2011. [En ligne]. Disponible : https://doi.org/10.1137/10079687X

[25] S. Agarwal, K. Mierle et T. C. S. Team, “Ceres Solver,” 3 2022. [En ligne]. Disponible : https://github.com/ceres-solver/ceres-solver

[26] S. Agarwal, “Ceres users.” [En ligne]. Disponible : http://ceres-solver.org/users.html

[27] J. J. Moré, B. S. Garbow et K. E. Hillstrom, “User guide for MINPACK-1,” Rapport technique ANL-80-74, août 1980.

[28] M. I. A. Lourakis et A. A. Argyros, “Sba : A software package for generic sparse bundle adjustment,” *ACM Trans. Math. Softw.*, vol. 36, n°. 1, mars 2009. [En ligne]. Disponible : https://doi.org/10.1145/1486525.1486527

[29] G. Callou, P. Maciel, E. Tavares, E. Andrade, B. Nogueira, C. Araujo et P. Cunha, “Energy consumption and execution time estimation of embedded system applications,” *Microprocessors and Microsystems*, vol. 35, n°. 4, p. 426–440, 2011. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0141933110000529

[30] K. Shiann-Rong, W. Kun-Yi et Y. Kee-Khuan, “Energy-efficient multiple-precision floating-point multiplier for embedded applications,” *Journal of Signal Processing Systems*, 2013. [En ligne]. Disponible : https://doi.org/10.1007/s11265-012-0695-1

[31] S. Huang, S. Xiao et W. Feng, “On the energy efficiency of graphics processing units for scientific computing,” dans *2009 IEEE International Symposium on Parallel Distributed Processing*, 2009, p. 1–8.

[32] E. Masanet, A. Shehabi, N. Lei, S. Smith et J. Koomey, “Recalibrating global data center energy-use estimates,” *Science*, vol. 367, n°. 6481, p. 984–986, 2020. [En ligne]. Disponible : https://www.science.org/doi/abs/10.1126/science.aba3758

[33] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone et J. C. Phillips, “Gpu computing,” *Proceedings of the IEEE*, vol. 96, n°. 5, p. 879–899, 2008.

[34] J. Nickolls, I. Buck, M. Garland et K. Skadron, “Scalable parallel programming with cuda : Is cuda the parallel programming model that application developers have been

waiting for ?" *Queue*, vol. 6, n°. 2, p. 40–53, mar 2008. [En ligne]. Disponible : https://doi.org/10.1145/1365490.1365500

[35] N. J. Higham, S. Pranesh et M. Zounon, "Squeezing a matrix into half precision, with an application to solving linear systems," *SIAM Journal on Scientific Computing*, vol. 41, n°. 4, p. A2536–A2551, 2019. [En ligne]. Disponible : https://doi.org/10.1137/18M1229511

[36] S. Gratton, E. Simon, D. Titley-Peloquin et P. L. Toint, "Minimizing convex quadratics with variable precision conjugate gradients," *Numerical Linear Algebra with Applications*, vol. 28, n°. 1, p. e2337, 2021. [En ligne]. Disponible : https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.2337

[37] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numerical Mathematics*, 1976. [En ligne]. Disponible : https://doi.org/10.1007/BF01931367

[38] M. Bartholomew-Biggs, S. Brown, B. Christianson et L. Dixon, "Automatic differentiation of algorithms," *Journal of Computational and Applied Mathematics*, vol. 124, n°. 1, p. 171–190, 2000, numerical Analysis 2000. Vol. IV : Optimization and Nonlinear Equations. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0377042700004222

[39] A. Griewank et A. Walther, "Introduction to automatic differentiation," *PAMM*, vol. 2, n°. 1, p. 45–49, 2003. [En ligne]. Disponible : https://onlinelibrary.wiley.com/doi/abs/10.1002/pamm.200310012

[40] W. X. Thomas F. Coleman, *Chapter 1 : Fundamentals of Automatic Differentiation and the Use of ADMAT*, 2016, p. 1–14. [En ligne]. Disponible : https://epubs.siam.org/doi/abs/10.1137/1.9781611974362.ch1

[41] S. Agarwal, "Condition number of hessian matrix." [En ligne]. Disponible : http://ceres-solver.org/nnls_solving.html#cgnr

[42] D. Ruiz, "A scaling algorithm to equilibrate both rows and columns norms in matrices 1," 2001.

[43] D. Orban et A. S. Siqueira, "JuliaSmoothOptimizers : Infrastructure and solvers for continuous optimization in Julia," 2019. [En ligne]. Disponible : https://juliasmoothoptimizers.github.io

[44] D. Orban, A. S. Siqueira et contributors, "NLPModels.jl : Data structures for optimization models," https://github.com/JuliaSmoothOptimizers/NLPModels.jl, July 2020.

[45] N. Gould et J. Scott, "The state-of-the-art of preconditioners for sparse linear least-squares problems," *ACM Trans. Math. Softw.*, vol. 43, n°. 4, jan 2017. [En ligne]. Disponible : https://doi.org/10.1145/3014057

[46] I. S. Duff, “Ma57—a code for the solution of sparse symmetric definite and indefinite systems,” *ACM Trans. Math. Softw.*, vol. 30, n°. 2, p. 118–144, jun 2004. [En ligne]. Disponible : https://doi.org/10.1145/992200.992202

[47] I. S. Duff et J. K. Reid, “The multifrontal solution of indefinite sparse symmetric linear,” *ACM Trans. Math. Softw.*, vol. 9, n°. 3, p. 302–325, sep 1983. [En ligne]. Disponible : https://doi.org/10.1145/356044.356047

[48] A. Montoison et D. Orban, “Tricg and trimr : Two iterative methods for symmetric quasi-definite systems,” *SIAM Journal on Scientific Computing*, vol. 43, n°. 4, p. A2502–A2525, 2021. [En ligne]. Disponible : https://doi.org/10.1137/20M1363030

[49] E. Carson et N. J. Higham, “Accelerating the solution of linear systems by iterative refinement in three precisions,” *SIAM Journal on Scientific Computing*, vol. 40, n°. 2, p. A817–A847, 2018. [En ligne]. Disponible : https://doi.org/10.1137/17M1140819

[50] D. Newton, R. Bollapragada, R. Pasupathy et N. K. Yip, “Retrospective approximation for smooth stochastic optimization,” 2021. [En ligne]. Disponible : https://arxiv.org/abs/2103.04392

[51] B. M. Averick, J. J. Moré, C. H. Bischof, A. Carle et A. Griewank, “Computing large sparse jacobian matrices using automatic differentiation,” *SIAM Journal on Scientific Computing*, vol. 15, n°. 2, p. 285–294, 1994. [En ligne]. Disponible : https://doi.org/10.1137/0915020

ANNEXE A ARCHITECTURE DES ORDINATEURS

Tableau A.1 Architecture des ordinateurs sur le noeud de calcul Frontal22 (F22).

OS :	Linux (x86_64-pc-linux-gnu)
CPU :	Intel(R) Xeon(R) CPU E5530 @ 2.40GHz
GPU :	None
RAM :	48Go
Julia Version :	1.7.1

Tableau A.2 Architecture de mon ordinateur personnel (PC).

OS :	Windows 10 (x86_64-w64-mingw32)
CPU :	Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz
GPU :	NVIDIA GeForce GTX 1650
RAM :	8Go
Julia Version :	1.7.3

Les tableaux A.1 et A.2 présentent l'architecture sur lesquels les différents tests ont été effectués. Sur les tableaux ou figures correspondant la mention PC ou F22 détermine sur quel ordinateur le test a été effectué.

ANNEXE B COMPARAISON DE LSMR ET CG

TABLEAU B.1 – Algorithme de Levenberg-Marquardt avec CG (F22).

Nom	N° res	N° jprod	Statut	$\|F(x)\|$	$\|J_0^T F_0\|$	$\|J^T F\|$	Temps
201-54427	997	496980	max_time	4.86e+02	4.45e+08	1.05e+04	2.00e+04
206-54562	971	484963	max_time	4.83e+02	4.44e+08	5.48e+04	2.00e+04
215-55910	998	499909	max_iter	4.84e+02	4.44e+08	5.93e+04	1.98e+04
49-7776	1003	494363	max_iter	1.74e+02	2.40e+07	1.45e+03	2.83e+03
73-11032	504	92525	max_iter	2.34e+02	3.41e+07	1.76e+06	8.70e+02
138-19878	1003	500623	max_iter	3.60e+02	5.55e+07	7.33e+03	8.16e+03
412-52215	835	416923	max_time	4.74e+02	9.76e+07	5.52e+03	2.00e+04
460-56811	919	429431	max_time	5.13e+02	1.08e+08	5.01e+04	2.00e+04
539-65220	763	380986	max_time	5.54e+02	1.09e+08	1.01e+04	2.00e+04
598-69218	7	1005	first_order	1.93e+03	1.68e+13	5.32e+07	6.13e+01
646-73584	9	1011	first_order	2.67e+03	2.12e+14	1.21e+09	6.83e+01
810-88814	345	255504	max_time	1.44e+03	4.44e+09	2.45e+09	2.00e+04
856-93344	282	253369	max_time	2.15e+03	7.31e+10	3.25e+09	2.01e+04
885-97473	262	233410	max_time	1.09e+03	9.98e+10	4.66e+08	2.01e+04
931-102699	320	227706	max_time	2.02e+03	3.61e+11	1.93e+09	2.01e+04
969-105826	320	227776	max_time	2.75e+03	5.15e+11	1.04e+09	2.00e+04
1118-118384	329	185204	max_time	2.33e+03	1.03e+12	3.17e+09	2.01e+04
1152-122269	509	35469	max_iter	6.51e+03	1.11e+13	7.32e+12	4.02e+03
1197-126327	248	176913	max_time	3.89e+03	1.25e+13	4.86e+09	2.01e+04
1266-132593	347	169148	max_time	5.40e+03	1.22e+13	4.96e+10	2.00e+04
1469-145199	291	158971	max_time	3.02e+03	1.30e+13	1.03e+10	2.00e+04
1514-147317	529	40586	max_iter	3.36e+03	1.25e+13	4.54e+10	5.26e+03
1587-150845	578	9377	max_iter	3.35e+03	1.18e+13	1.36e+12	1.39e+03
52-64053	621	309842	max_time	1.09e+03	3.32e+08	4.12e+04	2.01e+04
245-198739	215	107107	max_time	1.88e+03	5.24e+08	1.90e+06	2.01e+04
427-310384	143	71071	max_time	2.12e+03	6.81e+08	6.23e+05	2.02e+04
1408-912229	7	163	first_order	1.60e+04	1.76e+15	1.48e+09	1.48e+02
1350-894716	7	255	first_order	1.38e+04	1.01e+15	8.49e+08	2.20e+02
1288-866452	7	422	first_order	1.02e+04	1.98e+14	4.45e+08	3.32e+02

Tableau B.1 – Algorithme de Levenberg-Marquardt avec CG (F22). (suite)

Nom	N° res	N° jprod	Statut	$\|F(x)\|$	$\|J_0^T F_0\|$	$\|J^T F\|$	Temps
1238-843534	23	4479	first_order	3.35e+03	1.48e+13	6.90e+07	3.23e+03
202-132796	283	140626	max_time	1.16e+03	6.19e+08	1.20e+05	2.01e+04
273-176305	237	117972	max_time	9.50e+02	7.45e+08	1.95e+05	2.00e+04
39-18060	1003	500687	max_iter	3.97e+02	2.25e+08	1.70e+04	6.40e+03
161-48126	1003	500858	max_iter	4.64e+02	3.95e+08	1.76e+04	1.70e+04
193-53101	977	487870	max_time	4.78e+02	4.46e+08	5.26e+04	2.00e+04
257-65132	853	421172	max_time	5.00e+02	4.65e+08	1.56e+04	2.00e+04
372-47423	685	341631	first_order	4.78e+02	2.28e+08	1.31e+03	1.29e+04
744-543562	81	39562	max_time	2.48e+03	7.40e+09	4.85e+06	2.05e+04
783-84444	644	254198	max_time	6.58e+02	1.21e+10	4.12e+07	2.00e+04
1064-113655	754	188391	max_time	1.39e+03	9.55e+11	5.76e+08	2.01e+04
1340-137079	213	162487	max_time	2.63e+03	1.44e+13	3.45e+09	2.01e+04
89-110973	345	172172	max_time	1.09e+03	3.98e+08	1.78e+05	2.00e+04
1778-993923	7	122	first_order	1.82e+04	3.19e+15	2.49e+09	1.39e+02
1642-153820	225	153918	max_time	5.56e+03	1.16e+13	6.15e+10	2.01e+04
1695-155710	210	153122	max_time	7.26e+03	1.16e+13	1.15e+11	2.01e+04
225-57665	945	470880	max_time	4.86e+02	4.46e+08	4.13e+04	2.00e+04
318-41628	83	36915	first_order	4.16e+02	3.21e+08	1.40e+03	1.36e+03
707-78455	520	15681	max_iter	4.28e+03	4.21e+13	6.70e+11	1.18e+03
1031-110968	529	16687	max_iter	2.11e+03	8.53e+11	6.00e+10	1.80e+03
1235-129634	549	22587	max_iter	1.70e+03	1.22e+13	2.84e+10	2.77e+03
1184-816583	7	565	first_order	8.63e+03	2.04e+14	2.85e+08	4.14e+02
1158-802917	7	639	first_order	7.34e+03	2.48e+14	2.55e+08	4.74e+02
1102-780462	7	1005	first_order	5.08e+03	4.28e+13	1.72e+08	6.94e+02
16-22106	207	99414	first_order	3.09e+02	1.71e+08	9.34e+02	1.71e+03
88-64298	517	257415	max_time	1.14e+03	4.15e+08	4.27e+04	2.01e+04
135-90642	367	182408	max_time	1.02e+03	5.06e+08	6.38e+04	2.01e+04
142-93602	355	176329	max_time	8.92e+02	5.31e+08	4.67e+04	2.01e+04
150-95821	375	186341	max_time	7.59e+02	5.28e+08	4.79e+04	2.01e+04
161-103832	343	170370	max_time	7.81e+02	5.29e+08	6.73e+04	2.00e+04
173-111908	329	163429	max_time	8.19e+02	5.22e+08	1.60e+04	2.01e+04
182-116770	319	158584	max_time	9.50e+02	7.25e+08	1.84e+05	2.01e+04
237-154414	257	127655	max_time	1.35e+03	6.91e+08	1.90e+05	2.01e+04
253-163691	247	122900	max_time	1.06e+03	6.98e+08	1.82e+05	2.01e+04

Tableau B.1 – Algorithme de Levenberg-Marquardt avec CG (F22). (suite)

Nom	N° res	N° jprod	Statut	$\|F(x)\|$	$\|J_0^T F_0\|$	$\|J^T F\|$	Temps
262-169354	239	118936	max_time	9.73e+02	7.08e+08	5.39e+05	2.01e+04
287-182023	237	118027	max_time	1.02e+03	7.77e+08	2.12e+05	2.01e+04
308-195089	173	110159	max_time	1.48e+03	9.20e+08	5.46e+07	2.00e+04
21-11315	1003	500603	max_iter	2.64e+02	1.64e+08	7.60e+03	3.55e+03
50-20431	1003	500677	max_iter	3.80e+02	2.37e+08	8.10e+03	6.90e+03
126-40037	1003	500789	max_iter	4.74e+02	3.34e+08	2.39e+04	1.46e+04
138-44033	1003	500775	max_iter	4.66e+02	3.88e+08	4.84e+04	1.73e+04
951-708276	37	10072	first_order	3.19e+03	2.01e+12	1.12e+07	5.89e+03
356-226730	181	89966	max_time	2.02e+03	1.13e+09	1.44e+06	2.01e+04
170-49267	1003	500793	max_iter	4.71e+02	3.95e+08	8.19e+03	1.88e+04
174-50489	1003	500848	max_iter	4.68e+02	4.10e+08	7.21e+03	1.87e+04

Tableau B.2 – Algorithme de Levenberg-Marquardt avec LSMR (F22).

Nom	N° res	N° jprod	Statut	$\|F(x)\|$	$\|J_0^T F_0\|$	$\|J^T F\|$	Temps
201-54427	233	31298	first_order	4.96e+02	4.45e+08	2.56e+03	1.38e+03
206-54562	231	37693	first_order	4.92e+02	4.44e+08	2.58e+03	1.67e+03
215-55910	247	37463	first_order	4.94e+02	4.44e+08	2.68e+03	1.67e+03
49-7776	67	31092	first_order	1.80e+02	2.40e+07	1.40e+02	2.17e+02
73-11032	61	27834	first_order	1.87e+02	3.41e+07	2.01e+02	2.77e+02
138-19878	505	248952	first_order	3.65e+02	5.55e+07	3.35e+02	4.56e+03
412-52215	127	58110	first_order	4.76e+02	9.76e+07	5.86e+02	2.98e+03
460-56811	251	113895	first_order	5.14e+02	1.08e+08	6.41e+02	5.98e+03
539-65220	285	136216	first_order	5.56e+02	1.09e+08	6.56e+02	8.24e+03
598-69218	1003	1002	max_iter	3.95e+03	1.68e+13	2.29e+08	4.44e+02
646-73584	585	584	first_order	4.37e+03	2.12e+14	9.46e+08	2.75e+02
810-88814	1003	1002	max_iter	5.00e+03	4.44e+09	3.66e+09	5.28e+02
856-93344	1003	1002	max_iter	5.63e+03	7.31e+10	1.12e+10	5.54e+02
885-97473	1003	1002	max_iter	5.67e+03	9.98e+10	1.15e+10	5.87e+02
931-102699	1003	1002	max_iter	5.82e+03	3.61e+11	1.12e+10	6.10e+02
969-105826	1003	1002	max_iter	5.85e+03	5.15e+11	1.13e+10	6.30e+02
1118-118384	1003	1002	max_iter	7.30e+03	1.03e+12	3.25e+10	6.92e+02
1152-122269	1003	1002	max_iter	9.18e+03	1.11e+13	3.09e+11	7.09e+02

Tableau B.2 – Algorithme de Levenberg-Marquardt avec LSMR (F22). (suite)

Nom	N° res	N° jprod	Statut	$\|F(x)\|$	$\|J_0^T F_0\|$	$\|J^T F\|$	Temps
1197-126327	1003	1002	max_iter	9.53e+03	1.25e+13	4.16e+11	7.34e+02
1266-132593	1003	1002	max_iter	9.84e+03	1.22e+13	4.87e+11	7.63e+02
1469-145199	1003	1002	max_iter	9.18e+03	1.30e+13	4.10e+11	8.37e+02
1514-147317	1003	1002	max_iter	9.86e+03	1.25e+13	7.78e+11	8.48e+02
1587-150845	1003	1002	max_iter	9.98e+03	1.18e+13	7.79e+11	8.61e+02
52-64053	497	234344	first_order	1.11e+03	3.32e+08	2.00e+03	1.75e+04
245-198739	527	86482	max_time	1.91e+03	5.24e+08	2.01e+04	2.02e+04
427-310384	1003	1002	max_iter	1.07e+04	6.81e+08	1.05e+08	2.22e+03
1408-912229	1003	1002	max_iter	2.18e+04	1.76e+15	8.76e+10	6.01e+03
1350-894716	1003	1002	max_iter	2.13e+04	1.01e+15	2.84e+10	6.27e+03
1288-866452	1003	1002	max_iter	2.09e+04	1.98e+14	5.07e+09	6.05e+03
1238-843534	1003	1002	max_iter	2.08e+04	1.48e+13	5.70e+09	5.94e+03
202-132796	1003	1002	max_iter	1.04e+04	6.19e+08	1.38e+08	1.07e+03
273-176305	1003	1002	max_iter	1.12e+04	7.45e+08	1.34e+08	1.35e+03
39-18060	83	33983	first_order	4.08e+02	2.25e+08	1.33e+03	4.75e+02
161-48126	139	20089	first_order	4.71e+02	3.95e+08	2.27e+03	8.37e+02
193-53101	221	32392	first_order	4.87e+02	4.46e+08	2.58e+03	1.45e+03
257-65132	307	76424	first_order	5.10e+02	4.65e+08	2.78e+03	3.87e+03
372-47423	43	17247	first_order	4.83e+02	2.28e+08	1.38e+03	7.84e+02
744-543562	1003	1002	max_iter	1.57e+04	7.40e+09	3.90e+08	4.28e+03
783-84444	1003	1002	max_iter	5.12e+03	1.21e+10	3.68e+09	5.71e+02
1064-113655	1003	1002	max_iter	6.44e+03	9.55e+11	1.88e+10	7.44e+02
1340-137079	1003	1002	max_iter	9.09e+03	1.44e+13	4.27e+11	8.89e+02
89-110973	451	163350	max_time	1.08e+03	3.98e+08	5.51e+03	2.01e+04
1778-993923	1003	1002	max_iter	2.26e+04	3.19e+15	1.02e+11	7.09e+03
1642-153820	1003	1002	max_iter	1.00e+04	1.16e+13	8.00e+11	9.64e+02
1695-155710	1003	1002	max_iter	1.03e+04	1.16e+13	8.02e+11	9.72e+02
225-57665	219	34027	first_order	4.93e+02	4.46e+08	2.62e+03	1.59e+03
318-41628	37	11873	first_order	4.17e+02	3.21e+08	1.45e+03	4.67e+02
707-78455	1003	1002	max_iter	4.99e+03	4.21e+13	3.94e+08	5.07e+02
1031-110968	1003	1002	max_iter	6.03e+03	8.53e+11	1.13e+10	7.44e+02
1235-129634	1003	1002	max_iter	9.64e+03	1.22e+13	4.36e+11	8.45e+02
1184-816583	1003	1002	max_iter	2.08e+04	2.04e+14	1.57e+09	5.77e+03
1158-802917	1003	1002	max_iter	2.06e+04	2.48e+14	7.85e+09	5.71e+03

Tableau B.2 – Algorithme de Levenberg-Marquardt avec LSMR (F22). (suite)

Nom	N° res	N° jprod	Statut	$\|F(x)\|$	$\|J_0^T F_0\|$	$\|J^T F\|$	Temps
1102-780462	1003	1002	max_iter	2.03e+04	4.28e+13	1.42e+09	5.58e+03
16-22106	55	24135	first_order	4.70e+02	1.71e+08	9.58e+02	4.40e+02
88-64298	215	102261	first_order	1.23e+03	4.15e+08	2.51e+03	8.42e+03
135-90642	79	25180	first_order	1.07e+03	5.06e+08	3.02e+03	2.93e+03
142-93602	83	24443	first_order	9.33e+02	5.31e+08	3.09e+03	2.91e+03
150-95821	65	14466	first_order	7.80e+02	5.28e+08	2.91e+03	1.76e+03
161-103832	95	27225	first_order	8.09e+02	5.29e+08	3.20e+03	3.44e+03
173-111908	257	84268	first_order	8.26e+02	5.22e+08	3.11e+03	1.12e+04
182-116770	1003	1002	max_iter	1.04e+04	7.25e+08	1.75e+08	8.66e+02
237-154414	1003	1002	max_iter	1.07e+04	6.91e+08	1.40e+08	1.16e+03
253-163691	1003	1002	max_iter	1.10e+04	6.98e+08	1.34e+08	1.20e+03
262-169354	1003	1002	max_iter	1.11e+04	7.08e+08	1.34e+08	1.24e+03
287-182023	1003	1002	max_iter	1.13e+04	7.77e+08	1.37e+08	1.27e+03
308-195089	1003	1002	max_iter	1.16e+04	9.20e+08	1.34e+08	1.36e+03
21-11315	81	35852	first_order	2.73e+02	1.64e+08	9.47e+02	2.62e+02
50-20431	55	17668	first_order	3.90e+02	2.37e+08	1.34e+03	2.82e+02
126-40037	125	29988	first_order	4.82e+02	3.34e+08	1.99e+03	9.87e+02
138-44033	135	24256	first_order	4.73e+02	3.88e+08	2.29e+03	9.01e+02
951-708276	1003	1002	max_iter	1.94e+04	2.01e+12	1.09e+09	4.72e+03
356-226730	1003	1002	max_iter	1.27e+04	1.13e+09	1.41e+08	1.78e+03
170-49267	137	18085	first_order	4.78e+02	3.95e+08	2.38e+03	7.59e+02
174-50489	189	24313	first_order	4.75e+02	4.10e+08	2.29e+03	1.02e+03

Les paramètres utilisés sont les suivants :

- $\eta_1 = \epsilon(x)^{\frac{1}{4}}$
- $\eta_2 = 0.99$
- $\sigma_1 = 10$
- $\sigma_2 = 0.1$
- $\lambda_{\min} = 0.1$
- restol $= \epsilon(x)^{\frac{1}{3}}$
- atol $= \sqrt{\epsilon(x)}$
- rtol $= \epsilon(x)^{\frac{1}{3}}$

— maximum d'itérations : 1000
— maximum d'itérations du sous-problème : 1000
— maximum de temps : 20000s

ANNEXE C COMPARAISON DE FACTORISATION LDL^T ET MINRES AVEC PRÉCONDITIONNEUR LDL^T

TABLEAU C.1 – Levenberg-Marquardt avec factorisation LDL^T (F22).

Nom	N° res	Statut	$\|F(x)\|$	$\|J_0^T F_0\|$	$\|J^T F\|$	Temps
262-169354	245	first_order	8.30e+02	7.08e+08	3.87e+03	1.59e+04
308-195089	154	first_order	8.85e+02	9.20e+08	1.78e+03	1.10e+04
21-11315	21	first_order	2.46e+02	1.64e+08	8.22e+01	9.21e+00
1118-118384	19	first_order	7.84e+02	1.03e+12	5.04e+06	5.05e+02
1469-145199	27	first_order	1.64e+03	1.30e+13	6.93e+07	1.65e+03
1238-843534	31	max_time	2.09e+03	1.48e+13	5.63e+15	2.05e+04
173-111908	25	first_order	7.27e+02	5.22e+08	1.88e+03	6.69e+02
783-84444	101	max_time	6.30e+02	1.21e+10	2.22e+09	2.02e+04
1152-122269	119	max_time	8.50e+02	1.11e+13	8.28e+10	2.10e+04
356-226730	68	first_order	9.95e+02	1.13e+09	5.22e+03	7.37e+03
257-65132	816	max_iter	4.49e+02	4.65e+08	2.53e+08	8.09e+03
49-7776	776	max_iter	1.63e+02	2.40e+07	2.91e+06	9.94e+02
931-102699	28	first_order	7.10e+02	3.61e+11	4.74e+05	5.03e+02
273-176305	206	first_order	8.31e+02	7.45e+08	4.45e+03	1.68e+04
1235-129634	68	max_time	8.99e+02	1.22e+13	4.54e+10	2.03e+04
1064-113655	41	first_order	7.51e+02	9.55e+11	2.82e+05	9.91e+02
89-110973	651	first_order	7.53e+02	3.98e+08	2.41e+03	7.82e+03
1102-780462	41	max_iter	6.15e+04	4.28e+13	1.17e+19	2.08e+04
253-163691	170	first_order	8.33e+02	6.98e+08	3.41e+03	9.93e+03
161-103832	56	first_order	6.99e+02	5.29e+08	4.19e+02	1.43e+03
1031-110968	97	max_time	7.42e+02	8.53e+11	2.27e+09	2.08e+04
142-93602	115	first_order	7.48e+02	5.31e+08	3.07e+03	2.74e+03
810-88814	105	max_time	6.38e+02	4.44e+09	5.30e+07	2.02e+04
1158-802917	30	max_time	2.17e+03	2.48e+14	4.67e+15	2.05e+04
856-93344	108	max_time	6.51e+02	7.31e+10	1.73e+08	2.02e+04
460-56811	252	max_time	5.08e+02	1.08e+08	2.13e+07	2.00e+04
73-11032	52	first_order	1.85e+02	3.41e+07	1.10e+02	3.77e+01
88-64298	27	first_order	7.68e+02	4.15e+08	2.16e+03	4.00e+02

Tableau C.1 – Levenberg-Marquardt avec factorisation LDL^T (F22). (suite)

Nom	N° res	Statut	$\|F(x)\|$	$\|J_0^T F_0\|$	$\|J^T F\|$	Temps
412-52215	163	first_order	4.71e+02	9.76e+07	5.33e+02	1.21e+03
1587-150845	89	max_time	3.36e+04	1.18e+13	1.42e+13	2.24e+04
744-543562	49	max_time	1.79e+03	7.40e+09	1.17e+08	2.05e+04
126-40037	976	max_iter	4.28e+02	3.34e+08	1.28e+07	2.70e+03
372-47423	686	max_time	4.90e+02	2.28e+08	1.41e+06	2.00e+04
1695-155710	96	max_time	1.46e+03	1.16e+13	3.74e+11	2.14e+04
969-105826	94	max_time	7.19e+02	5.15e+11	1.65e+08	2.06e+04
1184-816583	38	max_time	1.76e+03	2.04e+14	3.52e+13	2.01e+04
150-95821	31	first_order	7.10e+02	5.28e+08	2.28e+03	7.48e+02
539-65220	42	first_order	5.53e+02	1.09e+08	4.34e+02	3.93e+02
1288-866452	38	max_time	1.83e+03	1.98e+14	1.80e+14	2.01e+04
1340-137079	32	first_order	8.93e+02	1.44e+13	8.15e+07	1.47e+03
225-57665	810	max_iter	1.10e+05	4.46e+08	1.55e+14	9.22e+03
1350-894716	32	max_time	1.78e+03	1.01e+15	1.99e+13	2.01e+04
52-64053	775	max_iter	6.88e+02	3.32e+08	2.09e+04	5.52e+03
1266-132593	20	first_order	8.10e+02	1.22e+13	1.33e+07	5.40e+02
1408-912229	28	max_iter	2.46e+03	1.76e+15	9.37e+13	2.07e+04
646-73584	129	max_time	9.15e+02	2.12e+14	1.46e+09	2.00e+04
885-97473	115	max_time	6.63e+02	9.98e+10	7.46e+07	2.02e+04
135-90642	40	first_order	8.67e+02	5.06e+08	2.26e+03	9.17e+02
1778-993923	32	max_time	1.87e+03	3.19e+15	2.48e+10	2.07e+04
598-69218	18	first_order	5.84e+02	1.68e+13	8.65e+07	2.01e+02
427-310384	104	max_time	1.46e+03	6.81e+08	1.71e+05	2.00e+04
161-48126	exception	∞	∞	∞	∞	
138-19878	835	max_iter	3.54e+02	5.55e+07	2.79e+04	7.37e+03
245-198739	148	max_time	1.41e+03	5.24e+08	6.25e+05	2.01e+04
138-44033	980	max_iter	4.17e+02	3.88e+08	9.89e+06	3.21e+03
50-20431	62	first_order	3.21e+02	2.37e+08	9.93e+02	6.98e+01
202-132796	57	first_order	8.12e+02	6.19e+08	3.68e+03	2.62e+03
174-50489	978	max_iter	4.15e+02	4.10e+08	5.91e+07	3.87e+03
39-18060	65	first_order	3.48e+02	2.25e+08	1.07e+03	6.51e+01
182-116770	71	first_order	7.80e+02	7.25e+08	4.14e+03	2.19e+03
170-49267	977	max_iter	4.15e+02	3.95e+08	4.15e+07	3.95e+03
951-708276	40	max_time	2.01e+03	2.01e+12	3.51e+14	2.08e+04

Tableau C.1 – Levenberg-Marquardt avec factorisation LDL^T (F22). (suite)

Nom	N° res	Statut	$\|F(x)\|$	$\|J_0^T F_0\|$	$\|J^T F\|$	Temps
1642-153820	15	first_order	8.74e+02	1.16e+13	9.01e+06	1.03e+03
206-54562	825	max_iter	4.24e+02	4.44e+08	1.09e+05	1.39e+04
318-41628	84	first_order	4.15e+02	3.21e+08	7.48e+02	3.77e+02
287-182023	173	first_order	8.42e+02	7.77e+08	4.59e+03	1.18e+04
193-53101	829	max_iter	4.36e+02	4.46e+08	8.19e+07	9.01e+03
707-78455	125	max_time	1.00e+03	4.21e+13	6.09e+11	2.00e+04
237-154414	66	first_order	8.49e+02	6.91e+08	3.89e+03	3.26e+03
201-54427	825	max_iter	6.80e+02	4.45e+08	1.64e+09	8.76e+03
16-22106	38	first_order	1.90e+02	1.71e+08	2.16e+01	4.33e+01

Tableau C.2 – Levenberg-Marquardt avec MINRES et LDL^T (F22).

Nom	N° res	Statut	$\|F(x)\|$	$\|J_0^T F_0\|$	$\|J^T F\|$	Temps
262-169354	245	first_order	8.30e+02	7.08e+08	4.12e+03	1.53e+04
308-195089	154	first_order	8.85e+02	9.20e+08	4.59e+03	1.06e+04
21-11315	21	first_order	2.46e+02	1.64e+08	7.43e+01	8.47e+00
1118-118384	19	first_order	7.84e+02	1.03e+12	4.94e+06	5.34e+02
1469-145199	72	max_time	1.80e+06	1.30e+13	6.29e+16	2.11e+04
1238-843534	29	max_time	2.09e+03	1.48e+13	5.63e+15	2.01e+04
173-111908	25	first_order	7.27e+02	5.22e+08	1.88e+03	6.68e+02
783-84444	125	max_time	6.30e+02	1.21e+10	2.22e+09	2.03e+04
1152-122269	24	first_order	7.82e+02	1.11e+13	5.78e+07	7.33e+02
356-226730	66	first_order	9.95e+02	1.13e+09	3.89e+03	7.19e+03
257-65132	273	first_order	4.39e+02	4.65e+08	2.68e+03	1.34e+03
49-7776	0	exception	∞	∞	∞	∞
931-102699	28	first_order	7.10e+02	3.61e+11	4.24e+05	5.28e+02
273-176305	206	first_order	8.31e+02	7.45e+08	4.49e+03	1.64e+04
1235-129634	76	max_time	9.02e+02	1.22e+13	5.15e+10	2.03e+04
1064-113655	41	first_order	7.51e+02	9.55e+11	1.48e+05	1.07e+03
89-110973	651	first_order	7.53e+02	3.98e+08	2.41e+03	7.90e+03
1102-780462	39	max_time	1.60e+05	4.28e+13	6.63e+19	2.07e+04
253-163691	170	first_order	8.33e+02	6.98e+08	3.32e+03	1.14e+04
161-103832	56	first_order	6.99e+02	5.29e+08	4.19e+02	1.52e+03
1031-110968	116	max_time	7.42e+02	8.53e+11	2.24e+09	2.01e+04

Tableau C.2 – Levenberg-Marquardt avec MINRES et LDL^T (F22). (suite)

Nom	N° res	Statut	$\|F(x)\|$	$\|J_0^T F_0\|$	$\|J^T F\|$	Temps
142-93602	115	first_order	7.48e+02	5.31e+08	3.07e+03	2.77e+03
810-88814	125	max_time	6.38e+02	4.44e+09	5.29e+07	2.00e+04
1158-802917	31	max_time	2.17e+03	2.48e+14	4.67e+15	2.04e+04
856-93344	132	max_time	6.51e+02	7.31e+10	1.73e+08	2.01e+04
460-56811	712	max_time	5.08e+02	1.08e+08	2.13e+07	2.00e+04
73-11032	52	first_order	1.85e+02	3.41e+07	1.07e+02	3.42e+01
88-64298	27	first_order	7.68e+02	4.15e+08	2.16e+03	4.05e+02
412-52215	163	first_order	4.71e+02	9.76e+07	5.33e+02	1.14e+03
1587-150845	61	max_time	3.14e+04	1.18e+13	4.14e+12	2.18e+04
744-543562	50	max_time	1.79e+03	7.40e+09	1.17e+08	2.03e+04
126-40037	78	first_order	4.28e+02	3.34e+08	1.17e+03	2.29e+02
372-47423	834	max_iter	4.90e+02	2.28e+08	1.41e+06	1.57e+04
1695-155710	53	max_time	1.46e+03	1.16e+13	8.93e+10	2.05e+04
969-105826	109	max_time	7.19e+02	5.15e+11	1.64e+08	2.00e+04
1184-816583	42	max_time	1.76e+03	2.04e+14	3.85e+13	2.04e+04
150-95821	31	first_order	7.10e+02	5.28e+08	2.28e+03	7.32e+02
539-65220	42	first_order	5.53e+02	1.09e+08	4.33e+02	3.78e+02
1288-866452	38	max_time	1.83e+03	1.98e+14	1.80e+14	2.05e+04
1340-137079	61	max_time	1.19e+03	1.44e+13	2.31e+09	2.03e+04
225-57665	53	first_order	4.27e+02	4.46e+08	8.37e+02	2.61e+02
1350-894716	32	max_time	1.78e+03	1.01e+15	1.99e+13	2.09e+04
52-64053	775	max_iter	6.88e+02	3.32e+08	2.09e+04	5.21e+03
1266-132593	20	first_order	8.10e+02	1.22e+13	1.33e+07	5.90e+02
1408-912229	27	max_time	2.46e+03	1.76e+15	9.37e+13	2.00e+04
646-73584	144	max_time	9.15e+02	2.12e+14	1.46e+09	2.01e+04
885-97473	137	max_time	6.63e+02	9.98e+10	7.44e+07	2.03e+04
135-90642	40	first_order	8.67e+02	5.06e+08	2.26e+03	9.15e+02
1778-993923	30	max_time	1.87e+03	3.19e+15	2.48e+10	2.01e+04
598-69218	18	first_order	5.84e+02	1.68e+13	8.65e+07	1.96e+02
427-310384	106	max_time	1.46e+03	6.81e+08	1.71e+05	2.02e+04
161-48126	77	first_order	4.10e+02	3.95e+08	3.92e+02	3.55e+02
138-19878	835	max_iter	3.54e+02	5.55e+07	2.79e+04	3.95e+03
245-198739	181	max_time	1.41e+03	5.24e+08	6.25e+05	2.01e+04
138-44033	76	first_order	4.17e+02	3.88e+08	2.08e+03	2.60e+02

Tableau C.2 – Levenberg-Marquardt avec MINRES et LDL^T (F22). (suite)

Nom	N° res	Statut	$\|F(x)\|$	$\|J_0^T F_0\|$	$\|J^T F\|$	Temps
50-20431	62	first_order	3.21e+02	2.37e+08	9.93e+02	6.40e+01
202-132796	57	first_order	8.12e+02	6.19e+08	3.65e+03	2.66e+03
174-50489	75	first_order	4.14e+02	4.10e+08	8.21e+02	3.18e+02
39-18060	65	first_order	3.48e+02	2.25e+08	6.78e+01	6.21e+01
182-116770	71	first_order	7.80e+02	7.25e+08	4.19e+03	2.18e+03
170-49267	72	first_order	4.15e+02	3.95e+08	1.54e+03	3.00e+02
951-708276	40	max_iter	2.01e+03	2.01e+12	3.52e+14	2.09e+04
1642-153820	15	first_order	8.74e+02	1.16e+13	8.85e+06	1.08e+03
206-54562	201	first_order	4.24e+02	4.44e+08	2.66e+03	9.51e+02
318-41628	84	first_order	4.15e+02	3.21e+08	7.38e+02	3.54e+02
287-182023	173	first_order	8.42e+02	7.77e+08	4.56e+03	1.14e+04
193-53101	829	max_iter	4.36e+02	4.46e+08	8.19e+07	5.44e+03
707-78455	144	max_time	9.98e+02	4.21e+13	6.01e+11	2.05e+04
237-154414	66	first_order	8.49e+02	6.91e+08	3.80e+03	3.22e+03
201-54427	277	first_order	4.24e+02	4.45e+08	2.63e+03	1.31e+03
16-22106	38	first_order	1.90e+02	1.71e+08	2.16e+01	3.77e+01

Les paramètres utilisés sont les suivants :

- $\eta_1 = \epsilon(x)^{\frac{1}{4}}$
- $\eta_2 = 0.99$
- $\sigma_1 = 10$
- $\sigma_2 = 0.1$
- $\lambda_{\min} = 10^{-5}$
- restol $= \epsilon(x)^{\frac{1}{3}}$
- atol $= \sqrt{\epsilon(x)}$
- rtol $= \epsilon(x)^{\frac{1}{3}}$
- maximum d'itérations : 1000
- maximum d'itérations du sous-problème : 1000
- maximum de temps : 20000s

ANNEXE D CRITÈRES D'ARRÊT POUR LES FIGURES DU CHAPITRE 3.

Les critères d'arrêt de Levenberg-Marquardt pour les figures 3.6, 3.7, 3.8 et 3.9 sont les suivants :

- $\eta_1 = \epsilon(x)^{\frac{1}{4}}$
- $\eta_2 = 0.99$
- $\sigma_1 = 10$
- $\sigma_2 = 0.1$
- $\lambda_{\min} = 10^{-6}$
- restol $= \epsilon(x)^{\frac{1}{3}}$
- atol $= 0$
- rtol $= \epsilon(x)$
- in_rtol $= 10^{-2}$
- maximum d'itérations : 1000
- maximum d'itérations du sous-problème : 1000
- maximum de temps : 20000s