# Implémentation d'ENLSIP en Julia : Rapport Technique

Pierre Borie
ENAC

8 Juin 2020 -11 Septembre 2020

**Résumé**

J'ai effectué mon stage d'insertion professionnelle pour HydroQuebec Transénergie, plus précisément la division de prévision de la demande, en collaboration avec Mr Fabian Bastin de l'Université de Montréal, qui était mon maître de stage. J'ai travaillé sur la transcription de l'algorithme d'optimisation ENLSIP. Cet algorithme, initialement codé en Fortran77, est utilisé en production afin de prédire la demande en électricité dans une période de 24 heures. Ce rapport illustre les résultats de mes travaux sur la compréhension du fonctionnement de l'algorithme et sur son implémentation en Julia.

L'ensemble de mes travaux est disponible sur le répertoire git https://github.com/pierre-borie/nlcls.

## Table des matières

# 1 Introduction

## 1.1 Présentation du problème à résoudre

Pour la suite, $q, l, m$ et $n$ sont des entiers naturels tels que $q \leq l \leq n \leq m$.

On dispose de $m$ observations réelles $(t_i, y_i)$ et l'on souhaite ajuster un modèle $h$ de paramètre $x \in \mathbb{R}^n$ qui approche au mieux nos observations, le tout en satisfaisant $l$

contraintes, dont $q$ sont des contraintes d'égalité. Pour cela, on cherche le paramètre $x^*$ qui minimise la somme des écarts entre prédictions et observations au carré, soit :

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{m} (h(t_i, x) - y_i))^2$$

Pour $i = 1, \ldots, m$, $r_i : x \to y_i - h(t_i, x)$ est le $i$-ème résidu. L'algorithme sur lequel j'ai travaillé a pour finalité la résolution d'un problème d'optimisation de la forme :

$$\begin{cases} \min\limits_{x \in \mathbb{R}^n} \dfrac{1}{2} \|r(x)\|^2 \\ \text{sous contraintes} \\ \qquad c_i(x) = 0 i = 1, \ldots, q \\ \qquad c_j(x) \geq 0 j = q+1, \ldots, l \end{cases} \tag{1.1}$$

On a $r = (r_1, \ldots, r_m)^T : \mathbb{R}^n \mapsto \mathbb{R}^m$ la multi-fonction des résidus et $c = (c_1, \ldots, c_l)^T : \mathbb{R}^n \mapsto \mathbb{R}^l$ la multi-fonction des contraintes dont les $q$ premières composantes sont des contraintes d'égalité et les $l - q$ suivantes sont des contraintes d'inégalité. On suppose les $r_i$ et $c_j$ deux fois continûment différentiables. Finalement, $f : x \mapsto \dfrac{1}{2} \|r(x)\|^2$ est la fonction objectif à minimiser.

## 1.2 Idée générale de la résolution

L'algorithme implémenté est un algorithme itératif où, partant d'un point $x_0$, à une itération $k$, on calcule :

— $p_k \in \mathbb{R}^n$ une direction de descente

— $\alpha_k \in \mathbb{R}$ une longueur de pas

On met à jour le point courant par $x_{k+1} = x_k + \alpha_k p_k$. On décrit ci-dessous, en pseudo-code, le déroulement général de l'algorithme. Les différents termes seront précisés au fur et à mesure. Les noms de fonctions sont ceux utilisés dans les notebooks du répertoire git.

**Require:** $x_0$ point de départ, $r$ et $c$ multi-fonctions des contraintes et leurs matrices jacobiennes respectives $J$ et $A$

**begin**

init_working_set() {Initialisation du premier ensemble de travail}

$k = 0$

not_terminated=check_termination_criterias()

**while** not_terminated **and** $k < maxiter$ **do**

$p_{GN}, \lambda$ = working_set() {Calcul des multiplicateurs de Lagrange $\lambda$}

{Mise à jour de l'ensemble de travail}

{Calcul de la direction de descente $p_{GN}$ par Gauss-Newton rang plein}

$p_k$ = gn_direction_analysis()

{Analyse de la qualité de la direction $p_{GN}$}

{Si c'est une bonne direction de descente alors $p_k = p_{GN}$}

{Sinon, on calcule à nouveau une direction de descente soit par minimisation de sous-espace ($p_k = p_{SUB}$), soit par une méthode de Newton ($p_k = p_N$)}

$\alpha_k, w_k$ = compute_steplength() {Calcul des pénalités $w_k$ et de la longueur de pas $\alpha_k$}

**if** $p_k = p_N$ **then**

$x_{k+1} = x_k + p_k$

**else**

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

**end if**

{Mise à jour du point courant}

check_violated_constraints()

{Place les nouvelles contraintes nouvellement actives dans l'ensemble de travail}

not_terminated=check_termination_criterias()

$k = k + 1$

**end while**

**return** $x_k$

**end**

# 2 Calcul de la direction de descente

## 2.1 Méthode de Gauss-Newton

Les calculs effectués ci-dessous correspondent à ce qui est fait par la fonction SUBDIR du fichier dblreduns.f

### 2.1.1 Modélisation du sous-problème

Les explications ci-dessous se basent sur celles fournies par Lindström and Wedin (1988).

Soit $\tilde{x}$ une approximation de la solution du problème 1.1 et $\delta x \in \mathbb{R}^n$ pris dans un voisinage de $x$. À partir du développement de Taylor d'ordre 1, on a pour $i = 1, \ldots, m$ l'approximation suivante du $i$-ème résidu :

$$r_i(\tilde{x} + \delta x) \approx r_i(\tilde{x}) + \left( \frac{\partial r_i(\tilde{x})}{\partial x_1}, \ldots, \frac{\partial r_i(\tilde{x})}{\partial x_n} \right) \delta x \tag{2.1}$$

De même pour la $j$-ième contrainte, $j = 1, \ldots, l$ :

$$c_j(\tilde{x} + \delta x) \approx c_j(\tilde{x}) + \left( \frac{\partial c_j(\tilde{x})}{\partial x_1}, \ldots, \frac{\partial c_j(\tilde{x})}{\partial x_n} \right) \delta x \tag{2.2}$$

Ce qui, en notant $J$ et $A$ les matrices jacobiennes respectives des multi-fonctions $r$ et $c$, donne les linéarisations :

$$r(\tilde{x} + \delta x) = J(\tilde{x})\delta x + r(\tilde{x}) \tag{2.3}$$

$$c(\tilde{x} + \delta x) = A(\tilde{x})\delta x + c(\tilde{x}) \tag{2.4}$$

Ces linéarisations seront injectées dans la formulation du problème (1.1) mais avant, nous allons nous intéresser au caractère des contraintes.

On souhaite pouvoir distinguer les contraintes actives des contraintes inactives. Les contraintes d'égalité étant par définition actives, le problème se pose pour les contraintes d'inégalité.

Tout d'abord, une contrainte d'inégalité est considérée comme active si elle égale à 0 et inactive si elle est strictement positive. Cependant, Lindström and Wedin (1988) ne mentionnent pas les hypothèses de qualification des contraintes. On peut légitimement supposer qu'il s'agit de la Linear Independance Constraint Qualification (LICQ) afin de pouvoir utiliser les conditions KKT comme conditions nécessaires d'optimalité. On rappelle que si $x^*$,un minimum local du problème (1.1), satisfait les conditions KKT,

alors il existe $\lambda^* \in \mathbb{R}^l$ tel que :

$$\begin{aligned}
&\nabla f(x^*) - \sum_{i=1}^{l} \lambda_i^* c_i(x^*) = 0 \\
&c_i(x^*) = 0, \text{ pour } i = 1, \ldots, q \\
&c_j(x^*) \geq 0, \text{ pour } j = q+1, \ldots, l \\
&\lambda_j^* \geq 0, \text{ pour } j = q+1, \ldots, l \\
&\lambda_j^* c_j(x^*) = 0, \text{ pour } j = q+1, \ldots, l
\end{aligned} \tag{2.5}$$

La méthode implémentée dans ENLSIP est une méthode d'ensemble actif. Le principe est de ne prendre en compte que des contraintes actives pour les calculs de direction de descente et de pas à une itération donnée. Les contraintes d'inégalité actives peuvent être considérées comme des contraintes d'égalité. Cela permet de ramener le problème initial à un problème sous contraintes d'égalités. On pourrait penser qu'il suffit de prendre les inégalités qui sont égales à 0 et de mettre de côté les autres. Or ce n'est pas ce qui est fait dans l'algorithme. L'idée est de travailler avec un ensemble de contraintes $\mathcal{W}$ appelé ensemble de travail (working set en anglais) mis à jour à chaque début et fin d'itération. $\mathcal{W}$ comprend toujours toutes les contraintes d'égalité.

En début d'itération, $\mathcal{W}$ contient également les contraintes d'inégalité actives au point courant. Sa mise à jour consiste à lui retirer au plus une contrainte parmi celles qui vont devenir inactives une fois la direction de descente calculée. En fin d'itération, lorsque le point courant est mis à jour suite au calcul de la direction de descente et de la longueur de pas, on ajoute à $\mathcal{W}$ les contraintes d'inégalité devenues actives au nouveau point courant. On retire également les contraintes d'inégalité devenues inactives.

En somme, l'ensemble $\mathcal{W}$ contient les contraintes d'inégalité dont les linéarisations vont rester actives le long de la direction de recherche. Ce sont ces contraintes qui servent de base au calcul de la direction de descente.

Les détails quant à la détermination de l'ensemble $\mathcal{W}$ se trouvent dans la section 3. Le lagrangien du problème s'écrit comme :

$$\mathcal{L} : (x, \lambda) \mapsto f(x) - \sum_{i \in \mathcal{W}} \lambda_i c_i(x) \tag{2.6}$$

Les composantes du vecteur $\lambda$ désignent les multiplicateurs de Lagrange. C'est à partir de leurs valeurs, en particulier du signe des multiplicateurs associés aux inégalités, que l'ensemble $\mathcal{W}$ est mis à jour au début de chaque nouvelle itération.

Finalement, tout cela implique que l'on réduit le nombre de contraintes considérées

à $t$ ($q \leq t \leq l$).

Parmi ces $t$ contraintes, on retrouve :

— toutes les contraintes d'égalité

— les contraintes d'inégalité supposées actives le long de la direction de recherche, alors considérées comme des contraintes d'égalité.

On note $\hat{c}$ la restriction de $c$ à $t$ composantes et $\hat{A}$ sa matrice jacobienne. La linéarisation (2.4) de $c$ reste valable pour $\hat{c}$.

A l'itération $k$, le point courant $x_k$, les vecteurs $r(x_k)$ et $\hat{c}(x_k)$ ainsi que les matrices $J_k = J(x_k)$ et $\hat{A}_k = \hat{A}(x_k)$ sont fixés. On cherche alors la direction de descente $p_k$ qui nous rapproche au mieux d'un minimum de la fonction objectif sous les contraintes $c$ satisfaisant les conditions KKT (2.5).

En injectant les linéarisations de $r$ et $\hat{c}$, on obtient le sous-problème suivant :

$$\begin{cases} \min\limits_{p \in \mathbb{R}^n} \dfrac{1}{2} \|J_k p + r(x_k)\|^2 \\ \text{s.c. } \hat{A}_k p + \hat{c}(x_k) = 0 \end{cases} \tag{2.7}$$

C'est-à-dire un problème de moindres carrés linéaires sous contraintes d'égalité linéaires.

### 2.1.2 Factorisation QR

Le cœur de la résolution du problème (2.7) réside dans la factorisation QR de différentes matrices. On rappelle ci-dessous la factorisation QR d'une matrice disposant de plus de lignes que de colonnes.

**Théorème** (Factorisation QR)**.** *Soit $M \in \mathcal{M}_{m \times n}$ avec $m$ et $n$ des entiers quelconques tel que $m \geq n$.*

*Alors il existe $Q$ une matrice orthogonale $m \times m$, $R$ de taille $n \times n$ une matrice triangulaire supérieure et $P$ une matrice de permutation $n \times n$ tel que :*

$$MP = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$$

*avec $|r_{11}| \geq |r_{22}| \geq \ldots \geq |r_{nn}|$ où les $r_{ii}$ désignent les éléments diagonaux de $R$.*

Cette factorisation n'est pas unique mais dispose de nombreux avantages pratiques pour les problèmes de moindres carrés, ce que nous verrons tout au long de ce document. Notons d'ores et déjà que la décomposition ci-dessus est très facile à obtenir en Julia puisqu'il suffit d'appliquer la fonction `qr` du package `LinearAlgebra` à la matrice que

l'on souhaite factoriser. un argument optionnel permet même d'obtenir directement la matrice de permutation associée.

### 2.1.3 Décomposition en sous-systèmes

On souhaite d'abord factoriser la matrice $\hat{A}_k$ de taille $t \times n$ mais comme $t \leq n$, on passe par la factorisation QR de $\hat{A}_k^T$ :

$$\hat{A}_k^T P_a = Q_a \begin{pmatrix} R_a \\ 0 \end{pmatrix} \tag{2.8}$$

avec :

— $Q_a$ matrice orthogonale $n \times n$,

— $R_a$ matrice triangulaire supérieure $t \times t$ à éléments diagonaux décroissants en valeur absolue

— $P_a$ matrice de permutation $n \times n$

**N.B.** : Les matrices issues de la décomposition QR ci-dessus ne sont pas indexées par $k$ par soucis de lisibilité mais sont bien dépendantes de l'itération en cours. Il en va de même pour les différentes factorisations QR présentées dans la suite de ce document.

Comme $P_a$ est est une matrice de permutation, $P_a P_a^T = I_t$, et nous obtenons

$$\hat{A}_k = P_a \begin{pmatrix} L_a & 0 \end{pmatrix} Q_a^T \tag{2.9}$$

où $L_a = R_a^T$ est une matrice triangulaire inférieure $t \times t$.

Injectant cette factorisation dans (2.7), nous obtenons :

$$\hat{A}_k p = -c(x_k) \Longleftrightarrow P_a \begin{pmatrix} L_a & 0 \end{pmatrix} Q_a^T p = -c(x_k)$$

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

$$P_a L_a p_1 = -\hat{c}(x_k) \Longleftrightarrow L_a p_1 = -P_a^T \hat{c}(x_k) = b$$

On remarque que $p_1$, soit les $t$ premiers éléments de $p$, est totalement déterminé par les contraintes tandis que $p_2$, soit les $n-t$ derniers éléments de $p$, peut être choisi librement. Cela se comprend en introduisant l'espace nul de $\hat{A}_k$. Notons $Y$ le bloc des $t$ premières

colonnes de $Q_a$ et $Z$ celui des $n-t$ dernières colonnes de $Q_a$.[1] [2] ; on a $\hat{A}_k Z = 0$ d'où :

$$\hat{A}_k p = \hat{A}_k Q_a \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \hat{A}_k Y p_1$$

La stratégie va maintenant être de calculer $p_1$ par rapport aux contraintes puis, $p_1$ fixé, de calculer $p_2$ de sorte à minimiser la fonction objectif du problème linéarisé (2.7). En injectant $p_1$ et $p_2$ dans cette dernière, on peut alors décomposer notre problème en deux nouveaux sous-problèmes :

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

en introduisant $JQ_a = \begin{pmatrix} J_1 & J_2 \end{pmatrix}$, $J_1 \in \mathcal{M}_{m \times t}$ et $J_2 \in \mathcal{M}_{m \times (n-t)}$, puisque $Jp = JQ_a Q_a^T p$, $Q_a$ étant orthogonale.

Avant d'entamer les calculs, calculons le rang de $\hat{A}_k$. Comme $Q_a$ et $P_a$ sont orthogonales, on a $\text{rang}(\hat{A}_k) = \text{rang}(L_a)$ et $L_a$ étant triangulaire d'éléments diagonaux décroissants en valeur absolue, on définit son rang comme étant :

$$\bar{t} = \max_{1 \leq j \leq t} \{ j \mid |l_{jj}| \geq \varepsilon_{rank} \} \tag{2.10}$$

où $\varepsilon_{rank}$ est la racine carrée de la précision relative et les $l_{ii}$ sont les éléments diagonaux de $L_a$.

Cette valeur sera comparée au nombre de contraintes actives $t$ et modifiera la méthode de calcul de $p_1$ expliquée dans les sections 2.1.4 et 2.1.5.

### 2.1.4 Résolution avec stabilisation

Si $\bar{t} < t$, autrement dit si la matrice $\hat{A}_k$ est de rang déficient, il faut réaliser ce qui est appelé dans Lindström and Wedin (1988) une stabilisation. Cela passe par la factorisation QR de $L_a$ :

$$L_a P_\ell = Q_\ell R_\ell \tag{2.11}$$

avec :

— $Q_\ell$ matrice orthogonale $t \times t$

1. From Fabian: de (2.7) ?
2. From Pierre: Oui, c'est pour faire référence à la jacobienne des $t$ contraintes de l'ensemble de travail au point courant $x_k$

— $R_\ell$ matrice triangulaire supérieure $t \times t$ à éléments diagonaux décroissants en valeur absolue

— $P_\ell$ matrice de permutation $t \times t$

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

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

[3] [4] Pour $\omega_1 \leq t$, on définit $R_\ell^{(\omega_1)}$ comme la sous-matrice de $R_\ell$ constituée des éléments $r_{ij}$, avec $i \leq \omega_1, j \leq \omega_1$. $b_1^{(\omega_1)}$ désigne le vecteur constitué des $\omega_1$ premiers éléments de $b_1$.

Par suite, posant $\delta p_1^{(\omega_1)}$ solution du système d'inconnue $y \in \mathbb{R}^{\omega_1}$ $R_\ell y = b_1^{(\omega_1)}$, on obtient :

$$p_1 = P_\ell \begin{pmatrix} \delta p_1^{(\omega_1)} \\ 0 \end{pmatrix} \tag{2.12}$$

$p_1$ étant désormais calculé, il reste à résoudre pour $p_2$ $\min_{p_2} \frac{1}{2} \| J_2 p_2 + (J_1 P_\ell p_1 + r(x_k)) \|^2$.

On se sert alors de la factorisation QR de $J_2$ :

$$J_2 = Q_2 \begin{pmatrix} R_2 \\ 0 \end{pmatrix} P_2 \tag{2.13}$$

avec :

— $Q_2$ matrice orthogonale $m \times m$

— $R_2$ matrice triangulaire supérieure $(n-t) \times (n-t)$ à éléments diagonaux décroissants en valeur absolue

— $P_2$ matrice de permutation $(n-t) \times (n-t)$

On pose $d_2 = -Q_2^T (r(x_k) + J_1 P_\ell p_1)$ et pour $\omega_2 \leq n - t$, on définit $R_2^{(\omega_2)}$ comme le bloc triangulaire supérieur composé des $\omega_2$ premières lignes et colonnes de $R_2$ et $d_2^{(\omega_2)}$ le vecteur constitué des $\omega_2$ premiers éléments de $d_2$. Par suite, posant $\delta p_2^{(\omega_2)}$ solution de

3. From Fabian: Un peu difficile à suivre. On avait à la page précédente $L_a p_1 = b$ ce qui ne donne pas l'équation ci-dessous si on remplace $L_a$ en utilisant (2.11).

4. From Pierre: J'ai pourtant repris les mêmes équations que celle de l'article en modifiant seulement les indices...

$R_2^{(\omega_2)} y = d^{(\omega_2)}$, on obtient :

$$p_2 = P_2^T \begin{pmatrix} \delta p_2^{(\omega_2)} \\ 0 \end{pmatrix} \tag{2.14}$$

### 2.1.5 Résolution sans stabilisation

Si $\bar{t} = t$, on résout directement le système $L_a p_1 = b$ comme suit :

Pour $\omega_1 \leq t$, on définit $L_a^{(\omega_1)}$ comme le bloc triangulaire inférieur composé des $\omega_1$ premières lignes et colonnes de $L_a$ et $b^{(\omega_1)} = (b_1, b_2, \ldots, b_{\omega_1})^T$. Par suite, posant $\delta p_1^{(\omega_1)}$ comme étant la solution de $L_a^{(\omega_1)} y = b^{(\omega_1)}$, on obtient :

$$p_1 = \begin{pmatrix} \delta p_1^{(\omega_1)} \\ 0 \end{pmatrix} \tag{2.15}$$

$p_1$ étant désormais calculé, il reste à résoudre pour $p_2$ $\min\limits_{p_2} \frac{1}{2} \|J_2 p_2 + (J_1 p_1 + r(x_k))\|^2$.

On procède comme dans le cas avec stabilisation en se servant de la factorisation QR de $J_2$. On pose $d = -Q_2^T (r(x_k) + J_1 p_1)$ et pour $\omega_2 \leq n - t$, on définit $R_2^{(\omega_2)}$ comme le bloc triangulaire supérieur composé des $\omega_2$ premières lignes et colonnes de $R_2$ et $d^{(\omega_2)} = (d_1, d_2, \ldots, d_{\omega_2})^T$. Par suite, posant $\delta p_2^{(\omega_2)}$ solution de $R_2^{(\omega_2)} y = d^{(\omega_2)}$, on obtient :

$$p_2 = P_2^T \begin{pmatrix} \delta p_2^{(\omega_2)} \\ 0 \end{pmatrix} \tag{2.16}$$

### 2.1.6 Solution du sous-problème

Finalement, dans un cas comme dans l'autre, la solution du problème sous contraintes (2.7) de dimension $(\omega_1, \omega_2)$ est donnée par :

$$p^{(\omega_1, \omega_2)} = Q_a \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} \tag{2.17}$$

Les valeurs sont déterminées avant d'appliquer la méthode décrite dans cette section 2.1 et de deux façons différentes. C'est l'objet des sections 2.2 et 2.3.

## 2.2 Méthode de Gauss-Newton avec rang plein

[5] [6] Dans ce cas, on prend $\omega_1 = \text{rang}(\hat{A}_k)$, en le calculant comme en (2.10) et $\omega_2 = \text{rang}(J_2)$. Le rang de $J_2$ peut être déterminé comme dans la relation (2.10) en se servant des éléments diagonaux de $R_3$.

Cette variante de la méthode de Gauss-Newton est utilisée juste après avoir déterminé quel est l'ensemble de travail pour l'itération en cours et aboutit au calcul du vecteur noté $p_{GN}$ pour la suite. Ce dernier constitue une première proposition de direction de descente. Il est ensuite examiné via différents critères décrits ci-après. Si ces critères ne sont pas respectés, l'algorithme réalise un nouveau calcul de direction de descente. Pour cela, on utilise soit une nouvelle variante de Gauss-Newton avec de nouvelles valeurs de $\omega_1$ et $\omega_2$, soit une méthode de Newton (voir section 2.4).

## 2.3 Méthode de Gauss-Newton avec minimisation de sous-espace

### 2.3.1 Principe général

Cette méthode suit le schéma de résolution détaillé dans la section 2.1 avec des valeurs de $\omega_1$ et $\omega_2$, calculées spécifiquement et inférieures à celles choisies dans la méthode Gauss-Newton rang plein. L'idée générale pour calculer ces entiers est de déterminer la dimension $\omega$ à utiliser pour résoudre un système triangulaire supérieur d'inconnue $u$ et de second membre $v$ de la forme :

$$Ru = v \tag{2.18}$$

Cela définit comment découper la matrice $R$ afin de résoudre ce système comme en (2.15) ou (2.16).

Par suite, $\omega_1$ est la dimension à utiliser pour résoudre le système d'inconnue $u = P_\ell^T p_1$ :

$$R_a u = -Q_\ell^T P_a^T c(x_k)$$

$\omega_2$ est celle à utiliser pour résoudre celui d'inconnue $p_2$ :

$$R_2 p_2 = -Q_2^T (r(x_k) + J_1 P_\ell p_1)$$

5. From Pierre: On calcule les rangs des matrices $\hat{A}_k$ et $J_2$ en calculant celui des matrices triangulaires de leurs décompositions QR (ou LQ pour $\hat{A}_k$)

6. From Pierre: Après examen du code et de l'article, il n y a pas vraiment de détails supplémentaires sur cette méthode. L'approximation de la hessienne de la fonction objectif $f$ n'est jamais mentionnée...

### 2.3.2 Calcul de la dimension

On se place dans le cadre du système $Ru = v$

On note $\bar{r} = \text{rang}(R)$ et on définit pour $1 \leq i \leq \bar{r}$ :

$$workset_i = \|v_1, \ldots, v_i\| \text{ et } h_i = \left\| \frac{v_1}{R_{11}}, \ldots, \frac{v_i}{R_{ii}} \right\|$$

$workset_i$ est la norme du second membre du système (2.18) en dimension $i$, tandis que $h_i$ peut être vu comme une estimation de la norme de la solution de ce même sous-système. Ces grandeurs servent à déterminer la plus petite dimension à utiliser pour résoudre (2.18).

Celle-ci, notée mindim, est définie comme $\underset{1 \leq i \leq \bar{r}}{\text{argmax}}\ h_i |R_{ii}|$.

On définit ensuite l'entier dim, déterminé de deux façons différentes selon la variante de Gauss-Newton utilisée à l'itération précédente.

Si c'était Gauss-Newton rang plein, on choisit :

$$\text{dim} = \max \{1 \leq k \leq \bar{r} \mid h_k < \gamma_h h_{\bar{r}} \text{ et } workset_k > \gamma_w workset_{\bar{r}}\}$$

où $\gamma_h$ et $\gamma_w$ sont des constantes pré-définies. (respectivement 0.2 et 0.5 dans le code Fortran).

Au cas où cet indice ne serait pas défini, on prend $\text{dim} = \bar{r} - 1$.

Si la variante utilisée était la minimisation de sous-espace, on détermine d'abord par différents critères si l'itération précédente était satisfaisante ou pas (voir ces critères dans la fonction `subspace_minimization`). Si oui, on prend $\text{dim} = \omega^{(k-1)}$ où $\omega^{(k-1)}$ est la dimension utilisée à l'étape précédente. Sinon, on prend $\text{dim} = \max(1, \omega^{(k-1)} - 1)$.

Finalement, on renvoie

$$\omega = \max(\text{mindim}, \text{dim})$$

.

Un des problèmes rencontrés vis-à-vis de cette partie est que je ne suis pas parvenu à trouver de justifications théoriques quand le calcul de $\omega$ et les différents critères évoqués. Pour l'implémentation, je me suis donc contenté de reprendre exactement ce qui était fait dans le code Fortran. Pour les détails d'implémentation, voir les fonctions `compute_solving_dim` et `subspace_dimension` du notebook.

### 2.3.3 Commentaires sur la méthode

Les indications du code Fortran (voir fonction GNDCHK du fichier dblmod2nls.f) semblent indiquer que cette méthode est utilisée dans le cas où on a ajouté une contrainte à l'ensemble de travail au début de l'itération, ou bien si un des multiplicateurs de Lagrange associés aux contraintes d'inégalités actives est négatif. Lindström and Wedin (1988) font mention page 7 que cette méthode permettrait de "stabiliser" la méthode de Gauss-Newton rang plein, puisque l'on fait une minimisation dans un sous-espace de $\mathbb{R}^n$.

On peut également supposer, compte tenu de l'époque du code, cette variante était utile pour réduire les erreurs numériques qui peuvent arriver dans la variante rang plein. Cela ne reste qu'une supposition mais diminuerait la pertinence de ce calcul de dimension.

## 2.4 Méthode de Newton

Cette section explique les calculs effectués par la fonction NEWTON du fichier dblredunls.f.

### 2.4.1 Problème à résoudre

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

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

$\nabla^2_{xx}\mathcal{L}(x,\lambda)$ désigne la matrice hessienne du lagrangien introduit en (2.6) par rapport à la variable $x$, dont voici le détail de l'expression analytique :

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

où $\nabla^2 r_i$ (resp. $\nabla^2\hat{c}_i$) désigne la matrice hessienne du $i$-ème résidu (resp. de la $i$-ème

contrainte active). On posera $\Gamma = -\sum_{i=1}^{t} \lambda_i \nabla^2 \hat{c}_i(x_k) + \sum_{i=1}^{m} r_i(x_k) \nabla^2 r_i(x_k)$ pour la suite. On peut vérifier que $\Gamma$ est symétrique comme combinaison linéaire de matrices symétriques.

On peut alors reformuler le problème sous la forme :

$$\begin{cases} \min\limits_{p \in \mathbb{R}^n} \frac{1}{2} p^T \left[ J_k^T J_k + \Gamma \right] p + \left[ J_k^T r(x_k) \right]^T p \\ s.c. \\ \hat{A}_k p = -\hat{c}(x_k) \end{cases} \tag{2.19}$$

### 2.4.2 Calcul de la direction de Newton

[7] Comme pour la méthode Gauss-Newton, on pose $Q_a^T p = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$, $p_1 \in \mathbb{R}^t, p_2 \in \mathbb{R}^{n-t}$ et $J_k Q_a = \begin{pmatrix} J_1 & J_2 \end{pmatrix}$. On va également calculer $p_1$ par rapport aux contraintes en premier, puis calculer $p_2$ par la fonction objectif.

Sont connues les factorisations QR des matrices $\hat{A}_k$, $L_a$ et $J_2$, respectivement en (2.8), (2.11) et (2.13). On a également connaissance du rang de $\hat{A}_k$.

Si $\text{rang}(\hat{A}_k) = t$, soit $b = -P_1^T \hat{c}(x_k)$, l'équation des contraintes s'écrit alors $L_a p_1 = b$. Etant de rang $t$, ce système triangulaire inférieur se résout directement.

Si $\text{rang}(\hat{A}_k) < t$, on injecte la factorisation QR de $L_a$ dans l'équation des contraintes. Le premier système à résoudre devient alors $R_\ell P_\ell^T p_1 = b_1$ avec $b_1 = Q_\ell^T b$.

Soit $\omega_r$ le rang de $R_\ell$, $\delta p_1$ la solution du système triangulaire supérieur $R_\ell^{(\omega_r)} \delta p_1 = b_1$, ce qui donne :

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

La suite est indépendante du rang de $\hat{A}_k$ et $p_1$ est désormais fixé.

7. From Pierre: Cette méthode de Newton n'est que mentionnée dans l'article et il n y a pas de justification dans les commentaires du code Fortran, mis à part le fait que c'est dit plus efficace proche de la solution. Aucune mention de l'ignorance ou non des résidus... Peut être plus de détails dans le livre se trouvant uniquement en Suède ? (qui ressemble de plus à une boîte de Pandore contenant tous les mystères de cet algorithme...)

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

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

Avec $W = Q_a^T J_k^T J_k Q_a + Q_a^T \Gamma Q_a$ matrice $n \times n$ qui est trivialement symétrique et $h = (J_1 \ J_2)^T r(x_k) \in \mathbb{R}^n$. $p_1$ étant fixé, on souhaite minimiser $\varphi$ afin de trouver la solution du problème.

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

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

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

Tous les blocs sont évidemment symétriques et on a $W_{12}^T = W_{21}$

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

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

avec K indépendant de $v$. Les blocs de $W$ étant symétriques :

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

Si $W_{22}$ est définie positive, le minimum de $\varphi$ s'obtient en annulant son gradient. $p_2$ est alors donné par la résolution du système d'inconnue $v$ :

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

Encore une fois, si $W_{22}$ est définie positive, on utilise la factorisation de Cholesky de $W_{22}$ pour se ramener à la résolution de deux systèmes triangulaires consécutifs. Sinon, on renvoie une erreur et l'itération en cours est dite "en échec".

Si le calcul aboutit, la solution du problème (2.19), notée $p_N$ pour Newton, est encore une fois donnée par :

$$p_N = Q_a \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$$

# 3 Mise à jour de l'espace de travail

## 3.1 Mise en situation

On suppose qu'à l'optimum $x^*$, solution du problème (1.1), on a un ensemble $\mathcal{A}$ de contraintes saturées, i.e $\forall i \in \mathcal{A},\ c_i(x^*) = 0$. À ces contraintes, on associe des multiplicateurs de Lagrange $\lambda_i^*$ pour tout $i \in \mathcal{A}$.

Les vecteurs $x^*$ et $\lambda^*$ vérifient $\nabla \mathcal{L}(x^*, \lambda^*) = 0$ soit :

$$\hat{A}(x^*)\lambda^* = \nabla f(x^*) \tag{3.1}$$

avec $\lambda_i^* \geq 0$ pour les contraintes d'inégalité.

L'idée est, à chaque itération, de travailler avec des contraintes d'inégalité susceptibles d'être actives à la solution afin de pouvoir les considérer comme des égalités. Cela permet d'utiliser les méthodes de calcul de direction de descente vues dans ce document. Cette prédiction des contraintes actives est représentée par l'ensemble $\mathcal{W}$ introduit plus tôt et que l'on met à jour à chaque itération.

Le premier ensemble de travail est l'ensemble des contraintes actives au point de départ de l'algorithme. Je n'ai pas vu de mention stipulant qu'il faut prendre un point réalisable ou non.

## 3.2 Critère de suppression d'une contrainte

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

Le critère de sélection des contraintes se base sur une approche dite EQP (Equality Quadratic Programming). Avec cette approche, développée dans Gill et al. (1985), on suppose la direction de descente $p_k$ toujours réalisable, i.e $A_k p_k + c(x_k) \geq 0$ pour toute

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

Si notre estimation des multiplicateurs de Lagrange comporte une ou plusieurs composantes négatives, on décide alors de retirer une des contraintes associées à ces composantes. En l'occurence, on choisit celle dont le multiplicateur est le plus petit, soit le plus grand en valeur absolue, parmi ceux qui sont strictement négatifs. Cette contrainte que l'on supprime est celle qui deviendra inactive au prochain itéré.

## 3.3 Calcul des multiplicateurs de Lagrange

Nous sommes en début d'itération $k$, l'ensemble de travail est inchangé par rapport à l'itération précédente.

La première estimation des multiplicateurs de Lagrange se fait en résolvant pour $\lambda$ le système :

$$\hat{A}_k^T \lambda = \nabla f(x_k) \tag{3.2}$$

On résout ce système en utilisant la factorisation QR de $\hat{A}_k^T$ afin récrire le système d'inconnue $v = P_a^T \lambda$ :

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

qui est un système triangulaire supérieur. La fonction correspondant à ce calcul est la fonction MULEST du fichier dblmod2nls.f

Notre première estimation des multiplicateurs de Lagrange est alors $\lambda = P_a v$. On procède comme décrit en 3.2 afin de déterminer quelle contrainte retirer de $\mathcal{W}$. On parle de première estimation car dans le cas ou celle-ci n'implique aucune suppression de contrainte, on effectue une deuxième estimation, supposée meilleure, afin d'appliquer à nouveau notre critère de sélection.

Avant de réaliser cette estimation, il nous faut calculer la direction de descente $p_{GN}$ avec la méthode de Gauss-Newton rang plein (voir section 2.1) correspondant à l'ensemble de travail actuel.

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

$$\hat{A}_k^T \lambda = J_k^T \left[ J_k p_{GN} + r(x_k) \right] \tag{3.3}$$

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

On résout ce système manière analogue à la première estimation. Ces calculs sont effectués au sein de la fonction LEAEST du fichier dblredunls.f.

Si aucune contrainte n'est à supprimer, on maintient l'ensemble de travail tel quel. Sinon, on retire la contrainte de $\mathcal{W}$ et la ligne correspondant à cette contrainte de $\hat{A}_k$. Cette dernière matrice étant modifiée, toutes les décompositions QR qui en découlent le sont également, soit celles de $\hat{A}_k^T$, $L_a$ et $J_2$. Il faut donc les calculer à nouveau afin de pouvoir les utiliser dans la suite de l'itération en cours. Ceci fait, on peut calculer alors à nouveau la direction de descente avec la méthode de Gauss-Newton rang plein, direction qu'on pourra alors analyser puis re-calculer si nécessaire.

### 3.4 Commentaires

Cette section ainsi que la précédente illustrent un point important sur la méthode de l'algorithme : il s'agit d'une approche primale. C'est-à-dire que les itérés ne sont calculés que par rapport au problème primal (1.1) et on ne prend jamais en compte les multiplicateurs de Lagrange impliqués dans le problème dual. Comme nous le verrons plus tard, ils ne sont pas non plus impliqués dans le calcul de la longueur de pas. C'est sûrement une des raisons qui expliquent pourquoi on suppose que seules les directions de descente sont réalisables. La mise en place d'une approche primal-dual, avec une prise en compte plus globale des multiplicateurs de Lagrange, pourrait se révéler intéressante.

Il est également à noter que je ne suis pas parvenu à vraiment comprendre de justification théorique du critère de sélection des contraintes et que j'ai simplement reformulé le fonctionnement de l'algorithme sur ce point.

## 4 Calcul du pas

### 4.1 Principe général

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

On introduit alors la fonction de mérite :

$$\psi(x, w) = \frac{1}{2}\|r(x)\|^2 + \frac{1}{2}\sum_{i\in\mathcal{W}} w_i c_i(x)^2 + \frac{1}{2}\sum_{j\in\mathcal{I}} w_j \min(0, c_j(x))^2 \tag{4.1}$$

où $w \in \mathbb{R}^l$ désigne un vecteur de pénalités que l'on associe à chaque contrainte. $\mathcal{I}$ fait référence aux contraintes "inactives", c'est-à-dire qui ne sont pas dans l'ensemble de travail.

A l'itération $k$, le point courant $x_k$, la direction de descente $p_k$ et les pénalités (voir calcul section 4.2) $w^{(k)}$ sont fixés. La longueur de pas est alors définie comme :

$$\alpha_k = \min_{\alpha \in [\alpha_{min}, \alpha_{max}]} \psi(x_k + \alpha p_k, w^{(k)}) \tag{4.2}$$

Avec $\alpha_{max} = \min_{i \in \mathcal{I}} \left\{ -\frac{c_i(x_k)}{\nabla c_i(x_k)^T p_k} \text{ pour } i \text{ tel que } \nabla c_i(x_k)^T p_k < 0 \right\}$.
Si un tel minimum n'existe pas, on prend $\alpha_{max} = 3$.
$\alpha_{min}$ vaut $\alpha_{max}/3000$.
On note $\phi : \alpha \mapsto \psi(x_k + \alpha p_k, w^{(k)})$ pour la suite.

## 4.2 Calcul des pénalités

Introduites à la section précédente dans la fonction de mérite, ces pénalités servent à diriger la recherche du pas optimal tout en se maintenant dans un espace ou nos approximations linéaires des contraintes sont valables, puisque l'on calcule $p_k$ par rapport à ces dernières. Cela permet également de rechercher le pas dans un espace où l'ensemble $\mathcal{W}$ reste le plus possible réalisable.

Le calcul des pénalités est réalisé à chaque itération avant de démarrer celui du pas et fait l'objet d'un problème d'optimisation à part. Mais avant de s'attarder à ce problème, définissons quelques grandeurs utiles pour la suite.

$\kappa$ est une collection de $\xi$ vecteurs de $\mathbb{R}^l$ ($\xi = 4$ dans le code Fortran). La $i$-ème ligne de $\kappa$ correspond aux 4 plus petites pénalités jusqu'alors calculées pour la contrainte $i$ rangées par ordre décroissant.

$w^{(old)}$ est le dernier élément de $\kappa$, i.e. le vecteur contenant les plus petites pénalités calculées au cours des 4 itérations précédentes. ($\hat{w}^{(old)}$ contient les plus petites pénalités associées aux contraintes actives.)

On définit également le vecteur

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

ainsi que le réel

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

où $\delta$ est un réel valant 0.25 dans le code Fortran.

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

### 4.2.1 Problème d'optimisation à résoudre

$w_k$ est définie comme la solution du problème :

$$\begin{cases} \min\limits_{w\in\mathbb{R}^l} \|w\| \\ \text{s.c.} \\ y^T\hat{w} \geq \tau \text{ (ou } y^T\hat{w} = \tau) \\ w_i \geq w_i^{(old)}, \text{ pour } 1 \leq i \leq l \end{cases} \tag{4.3}$$

$\hat{w}$ désigne les poids associés aux contraintes actives, $y$ et $\tau$ sont définis ci-dessous selon différents cas.

1. Si $z^T\hat{w}^{(old)} \geq \mu$ et $\omega_1 \neq t$ ($\omega_1$ est la dimension définie en (2.15))

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

   Et la première contrainte est une contrainte d'inégalité.

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

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

   Et la première contrainte est une contrainte d'inégalité.

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

   $y = z$ et $\tau = \mu$ et la première contrainte est une contrainte d'égalité.

### 4.2.2 Résolution

Dans le cas où la première contrainte est une inégalité :
Pour $i \in \mathcal{W}$, $w_i^{(k)} = \max(\frac{\tau}{\|y\|^2} y_i, w_i^{(old)})$ (ce qui assure $y^T \hat{w}^{(k)} \geq \tau$)
Pour $i \in \mathcal{I}$, $w_i^{(k)} = w_i^{(old)}$.

Pour le cas où la contrainte est une égalité, l'idée est la même (voir la fonction `minimize_euclidean_norm` du notebook pour les spécificités).

## 4.3 Calcul de la longueur de pas

On rappelle que l'on cherche le minimum de $\phi : \alpha \mapsto \psi(x_k + \alpha p_k, w^{(k)})$ sur l'intervalle $[\alpha_{min}, \alpha_{max}]$

L'idée pour le calcul de $\alpha_k$ est, au lieu de calculer directement le minimum de $\phi$, de calculer le minimum d'approximations polynomiales successives de $\phi$ l'interpolant en deux ou trois points.

La méthode de calcul de pas est expliquée dans l'article Lindström and Wedin (1984), en particulier aux pages 290-291, pour le cas d'un problème initial sans contraintes. Après analyse du code Fortran (fonction LINEC du fichier dblreduns.f) j'ai remarqué que le déroulement du calcul dans le cas avec contraintes est exactement le même. C'est pour cette raison que je ne détaillerai pas la méthode dans ce document, les explications théoriques étant déjà fournies.

Les seules spécificités concernent le cas où l'on approche $\phi$ par un polynôme l'interpolant en deux points 0 et $\alpha_i$ prédéfini. En effet, les coefficients du polynôme interpolateur changent lorsque l'on ajoute des contraintes.

$$\text{Soit } P(\alpha) = \frac{1}{2}\|v_2\alpha^2 + v_1\alpha + v_0\|^2$$

Notant

$$F(\alpha) = \begin{pmatrix} r_1(x_k + \alpha p_k) \\ \vdots \\ r_m(x_k + \alpha p_k) \\ \sqrt{\hat{w}_1}\hat{c}_1(x_k + \alpha p_k) \\ \vdots \\ \sqrt{\hat{w}_t}\hat{c}_t(x_k + \alpha p_k) \end{pmatrix},$$

$v_0$, $v_1$ et $v_2$ dans $\mathbb{R}^{m+t}$ peuvent s'exprimer comme

$$
\begin{aligned}
v_0 &= F(0) \\
v_1 &= \left[\nabla r_1(x_k)^T p_k, \ldots, \nabla r_m(x_k)^T p_k, \sqrt{\hat{w}_1}\nabla \hat{c}_1(x_k)^T p_k, \ldots, \sqrt{\hat{w}_t}\nabla \hat{c}_t(x_k)^T p_k\right]^T \\
v_2 &= \left[\frac{1}{\alpha_i}\left(\frac{F_i(\alpha_i) - v_0^{(i)}}{\alpha_i} - v_1^{(i)}\right)\right]^T_{1\leq i\leq m+t}
\end{aligned}
$$

Le polynôme $P$ ainsi formé avec les coefficients ci-dessus interpole trivialement$\phi$ en 0 et $\alpha_i$.

On explicitera également une méthode d'Armijo-Goldstein utilisée dans le cas où les approximations polynomiales de $\phi$ ne fourniraient pas un résultat satisfaisant (trop petit par exemple). Cette méthode et surtout les paramètres utilisés sont sans doute spécifiques à l'algorithme.

$\varepsilon_{rel}$ désigne la racine carrée de la précision relative sur les double et $\gamma = 0.25$ dans le code Fortran. On part de $\alpha_0 \in \mathbb{R}$.

$\alpha = \alpha_0$
**while** $\phi(\alpha) \geq \phi(0) + \gamma * \alpha * \phi'(0)$ **and** $(\alpha\|p_k\| > \varepsilon_{rel}$ **or** $\alpha > \alpha_{min})$ **do**
  $\alpha = \alpha/2$
**end while**
**return** $\alpha$

Les critères de la conditionnelle **or** assurent que l'on ne renvoie pas une valeur de pas trop petite.

## 4.4 Commentaires sur le calcul du pas

Tout d'abord, cela n'est pas illustré dans ce document, mais la méthode de calcul du pas ressemble fortement à une méthode de régions de confiance, encore peu développées à l'époque où a été codé l'algorithme ENLSIP. Leur utilisation pourrait s'avérer intéressante pour une amélioration de l'algorithme, non seulement pour la longueur de pas mais aussi pour le calcul de la direction de descente.

On remarque également que les pénalités ne rentrent en ligne compte que dans la fonction de mérite. Un ajout de pénalités dans le calcul de la direction de descente, notamment via l'utilisation du Lagrangien augmenté, peut également être une piste d'amélioration.

Enfin, cette section montre aussi que l'on ne prend pas en compte les multiplicateurs de Lagrange qui confirme l'approche primal de l'algorithme.

# 5 Terminaison de l'algorithme

Cette partie concerne les conditions d'arrêt de l'algorithme. Les différents critères sont évalués par l'appel de la fonction TERCRI du fichier dblmod2nls.f.

## 5.1 Critères de convergence

A la fin de chaque itération, on teste différents critères afin de savoir si l'on peut stopper l'algorithme et renvoyer la valeur de minimum trouvée. On se place en fin de l'itération $k$ avec $x_k$ le nouvel itéré.

On teste d'abord les conditions nécessaires suivantes :

1. $\|\hat{c}(x_k)\| < \varepsilon_h$ et les contraintes inactives doivent être strictement positives
2. $\|\hat{A}_k^T \lambda - \nabla f(x_k)\| < \sqrt{\varepsilon_{rel}}\,(1 + \|\nabla f(x_k)\|)$
3. $\min_{i \in \mathcal{I}} \{\lambda_i \mid \lambda_i > 0\} \geq \varepsilon_{rel} \max_{1 \leq j \leq t} |\lambda_j|$
   ou $\min_{i \in \mathcal{I}} \{\lambda_i \mid \lambda_i > 0\} \geq \epsilon_{rel} \left(1 + \|r(x_k)\|^2\right)$ s'il n y a qu'une seule inégalité

Si ces trois conditions sont vérifiées, on vérifie ensuite si parmi ces conditions suffisantes, l'une d'elles est vérifiée :

1. $\|d\|^2 \leq \varepsilon_{rel}^2 \|r(x_k)\|^2$
2. $\|r(x_k)\|^2 \leq \varepsilon_{abs}^2$
3. $\|x_{k-1} - x_k\| < \varepsilon_x \|x_k\|$
4. $\dfrac{\sqrt{\varepsilon_{rel}}}{\|p_k\|} > 0.25$

Les réels $\varepsilon_{rel}, \varepsilon_x, \varepsilon_{abs}$ et $\varepsilon_h$ désignent des précisions relatives définies par l'utilisateur de l'algorithme. Par défaut, ils ont tous la même valeur qui est la racine carrée de la précision machine sur les double flottants.

## 5.2 Critères d'arrêt anormaux

Ces critères sont d'ordre algorithmique et ne reposent pas sur des principes mathématiques. Je n'ai implémenté qu'un seul d'entre eux : l'algorithme ne doit pas dépasser un nombre donné d'itérations. Les autres critères correspondent à de la gestion d'erreur, ce que je n'ai pas eu le temps de traiter en profondeur.

# 6 Résultats

Cette partie illustre différents résultats obtenus avec mon implémentation de l'algorithme en Julia dans le cas où il n y a que des contraintes d'égalité. Par manque de temps, je n'ai pas pu finir le cas avec contraintes d'inégalités.

Pour réaliser mes tests, je suis parti de données générées par un modèle à paramètres fixés à l'avance. J'ai ensuite perturbé ces données avec un bruit gaussien. Enfin, j'ai appliqué mon algorithme en prenant ces données perturbées comme données initiales afin de retrouver les paramètres du modèle d'origine.

## 6.1 Premier exemple

Vraie fonction $t \mapsto (t-2)(t-6)(t-10)$

Modèle à ajuster :

$g : (t, x_1, x_2, x_3) \mapsto (t-x_1)(t-x_2)(t-x_3)$

s.c.

$x_1 + x_2 + x_3 = 18$

$x_1 x_2 x_3 = 120$

Il s'agit d'un polynôme dont on souhaite chercher les racines avec les relations coefficients-racines comme contraintes.

Point de départ : $x_0 = [1, 0, 0]$

En 13 itérations, on trouve $x^* = \begin{pmatrix} 5,99202 \\ 10,00665 \\ 2,00133 \end{pmatrix}$

Sur la figure 1, la courbe orange représente le modèle à partit duquel on a généré les données. La courbe bleue, qui colle quasiment parfaitement à l'orange, montre les prédictions du modèle avec $x^*$ comme paramètre. Les points verts sont les données perturbées.

Dans ce cas-ci, on voit que l'on a trouvé les paramètres optimaux.

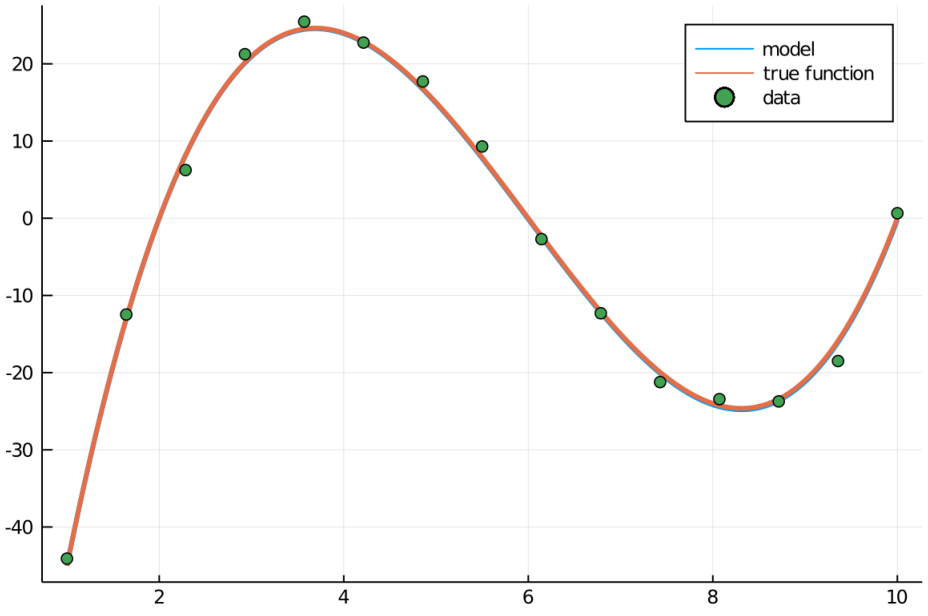


FIGURE 1 – Illustration graphique

### 6.2 Second exemple

Vraie fonction $t \mapsto 1 - \frac{t^2}{2} + \frac{t^4}{24}$

Modèle à ajuster :

$g : (t, x_1, x_2) \mapsto 1 + x_1 t^2 + {x_2}^3 \frac{t^4}{3}$

s.c.

$x_1 + 2x_2 = \frac{1}{2}$

On cherche $[-0.5, 0.5]$ comme minimum.

Point de départ : $x_0 = [1, 0]$

En 9 itérations, on trouve $x^* = \begin{pmatrix} 0.71843 \\ -0.10921 \end{pmatrix}$

Ici, comme l'illustre la figure 2, le modèle de paramètre $x^*$ ne correspond pas au modèle initial. Le point trouvé est cependant bien un minimum respectant bien les critères de convergence. Seulement il s'agit très probablement d'un minimum local dont le modèle résultant est assez éloigné de ce que l'on espérait. Dans un contexte d'ajustement de modèle, cela ne donne évidemment pas un résultat satisfaisant car une différence dans

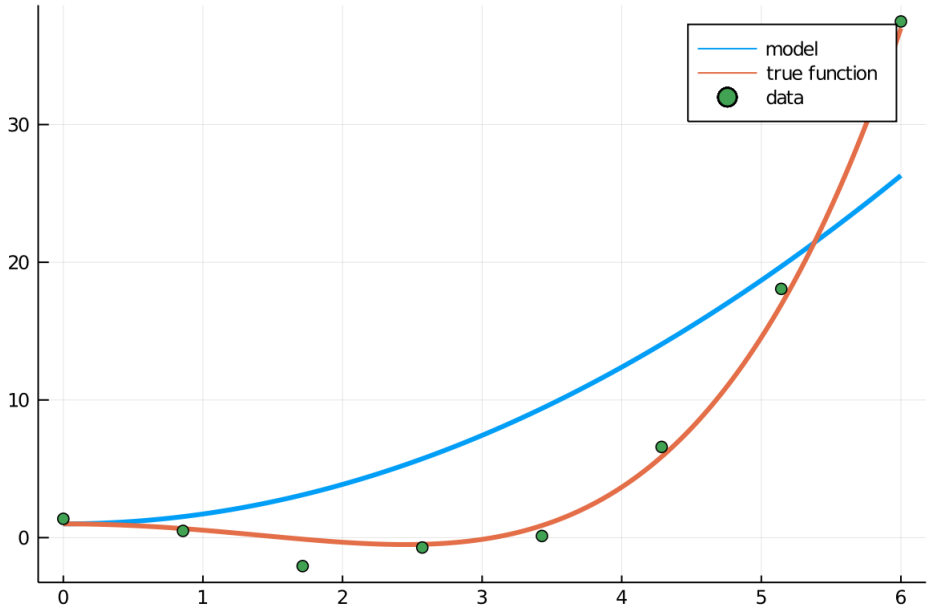


FIGURE 2 – Illustration graphique

les paramètres peut complètement modifier l'allure de la fonction résultante.

On peut néanmoins, en modifiant le point de départ, trouver le résultat souhaité. En effet en partant de $x_0 = [-0.2, 0.1]$, on trouve en 10 itérations :

$$x^* = \begin{pmatrix} -0.499987 \\ 0.50882 \end{pmatrix}$$

Ce résultat, illustré à la figure 3, est bien plus proche du résultat que l'on souhaite obtenir. Il montre également, bien que le modèle soit simple, la sensibilité au point de départ dont le choix est un élément important.

## Références

Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright. Model building and practical aspects of non linear progamming. *Computational Mathematical Programming*, F15 :210–247, 1985.

Per Lindström and Per-Åke Wedin. A new linesearch algorithm for nonlinear least squares problems. *Mathematical Programming*, Vol. 29 :268–296, 1984.

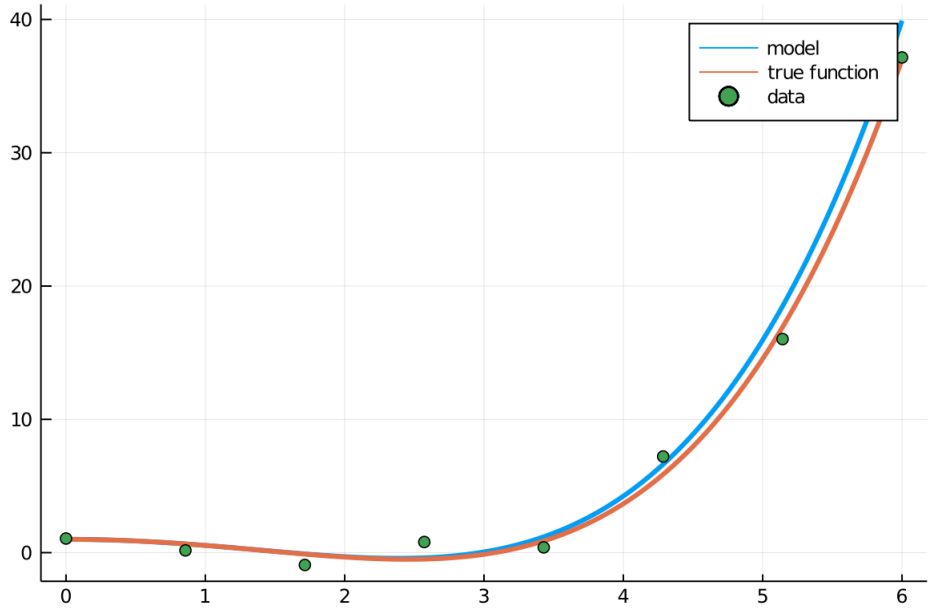


FIGURE 3 – Illustration graphique

Per Lindström and Per-Åke Wedin. Gauss-Newton based algorithms for constrained non linear least squares problems. *Institute of Information procressing, University of Umeå, S-910, Umeå, Sweden*, 1988.