Optimisation numérique et différentiation automatique pour un problème industriel

M. Barrault & C. Le Bris
CERMICS, Ecole Nationale des Ponts & Chaussées,
6 et 8 avenue Blaise Pascal, Cité Descartes,
Champs-sur-Marne, 77455 Marne-La-Vallée Cedex 2,
barrault@eleves.enpc.fr, lebris@cermics.enpc.fr

15 novembre 1999

Abstract

We consider an optimal control problem arising in an industrial context. As a first step towards the complete treatment of a coupled thermohydraulic problem, we define two simplified purely thermal problems on which we test different optimization techniques, and different strategies for the computation of the gradient of the criterion. The first problem is stationary and can be attacked in various manners : we compare three ways of calculating gradients and two classes of gradient algorithms. The second problem is a time-dependent one. In view of the large number of parameters we optimize upon, adjoint differentiation is the only feasible strategy for gradient computations. We then compare optimization algorithms. Conclusions are drawn on the basis of our numerical results, in order to tackle the industrial problem in the most efficient fashion.

Table des matières

1 Problématique générale **4**

2 Cas traités **5**

3 Présentation de quelques méthodes d'optimisation **8**
3.1 Méthodes stochastiques 9
3.2 Méthodes de gradient 10
3.2.1 Calcul de la direction de descente 10
3.2.1.1 Méthodes du premier ordre 10
3.2.1.2 Méthodes du second ordre et du pseudo second ordre 12
3.2.1.2.1 Décomposition du hessien 12
3.2.1.2.2 Les méthodes de Quasi-Newton 12
3.2.1.2.3 Différentes mises a jour 13
3.2.1.2.4 Mise a jour en jouant sur les colonnes . 15
3.2.1.2.5 La méthode de Quasi-Newton à mémoire limitée 15
3.2.1.2.6 Gauss-Newton 15
3.2.1.2.7 Iterated Subspace Minimization Methods (ISM) 16
3.2.1.2.8 Newton tronqué 16
3.2.1.2.9 Méthode de Newton Discret 17
3.2.1.2.10 Méthode de Newton Discret Tronqué avec mémoire 17
3.2.1.2.11 Cas des fonctions partiellement séparables 18
3.2.1.3 Méthodes de tenseur 19
3.2.2 Calcul de la longueur de descente 19
3.2.2.1 Conditions de Wolfe 20
3.2.2.2 Condition d'Armijo 20
3.2.2.3 Goldstein & Price 21
3.2.2.4 Interpolation cubique et parabolique 21
3.2.2.5 Recherche non monotone 21
3.2.2.6 Recherche avec courbure négative 22
3.2.2.7 Recherche avec mémoire 22
3.2.3 La méthode des régions de confiance 23
3.2.3.1 Algorithme de Newton 24
3.2.3.2 Décomposition du hessien 24
3.2.3.3 Dogleg algorithms 25
3.2.3.4 Gradient Conjugué tronqué 25
3.2.3.5 Régions de confiance non monotone 25
3.2.3.6 Combinaison des régions de confiance et du backtracking 25
3.2.3.7 Propriétés 26
3.3 Méthodes mixtes 27

3.4 Méthodes déterministes sans utilisation du gradient 28

4 Calcul du gradient **29**
4.1 Différences finies 30
4.2 Introduction et résolution de l'état adjoint 30
4.2.1 Adjoint du problème 30
4.2.2 La méthode One-Shot 31
4.2.3 Pseudo-Time method 31
4.3 La différentiation automatique 32
4.3.1 Mode direct 32
4.3.2 Mode inverse 33
4.3.3 Post-traitements du code adjoint construit par ODYSSEE 33
4.3.4 Le cas stationnaire 34
4.3.5 Le cas transitoire 35

5 Résultats numériques **36**
5.1 Problème $\mathcal{P}_1$ 37
5.1.1 Plusieurs algorithmes de Gradient Conjugué non Linéaire: Tableaux 5, 6 37
5.1.2 Plusieurs stratégies de "scaling": Tableaux 7, 8, 9 37
5.1.3 Plusieurs mises à jour: Tableaux 10, 11, 12, 13, 14 38
5.1.4 Plusieurs stratégies de calcul du gradient: Tableaux 15, 16, 17, 18 39
5.2 Problème $\mathcal{P}_2$ 40
5.2.1 Plusieurs algorithmes de Gradient Conjugué: Tableaux 19, 20 40
5.2.2 Plusieurs stratégies de "scaling": Tableaux 22, 23, 24 . . 42
5.2.3 Plusieurs mises à jour 42
5.2.3.1 Sans Scaling: Tableaux 25,26,27,28 42
5.2.3.2 Avec Scaling: Tableaux 29,30,31,32 43
5.2.4 Plusieurs recherches linéaires: Tableaux 33,34,35 43
5.2.5 Le paramètre α vaut 10^{-2} 44
5.2.5.1 Plusieurs algorithmes de Gradient Conjugué non Linéaire Tableaux 36, 37 44
5.2.5.2 Plusieurs stratégies de "scaling": Tableaux 38, 39, 40 44
5.2.5.3 Plusieurs mises à jour 44
5.3 Tableaux récapitulatifs 45

6 Conclusions et Perspectives **48**
6.1 Conclusions 48
6.2 Perspectives 49
6.3 Remerciements 50

Bibliographie **51**

1 Problématique générale

Un problème d'intérêt pratique est celui du contrôle de la température dans un véhicule. Prenons l'exemple d'une personne qui rentre dans son véhicule le matin. L'intérieur du véhicule est supposé assez froid du fait d'une température extérieure basse. L'utilisateur réchauffe l'habitacle par la mise en route d'un flux d'air à une certaine vitesse et une température constante. Le problème qui se pose au constructeur est de déterminer les données du flux d'air afin que l'utilisateur ne reste ni longtemps transi de froid, ni brutalement incommodé par une trop forte chaleur durant la prise en main du véhicule. Ce problème est d'un point de vue mathématique un problème d'optimisation d'une fonctionnelle par rapport à des paramètres (ici, ceux du flux d'air insufflé).

Les méthodes d'optimisation les plus courantes sont fondées sur l'usage des dérivées d'une fonctionnelle par rapport à des paramètres de contrôle. Quand on ne peut les calculer explicitement (ce qui est presque toujours le cas), on peut parfois évaluer une approximation de ces dérivées par différences finies, mais cette approche se révèle peu efficace voire impossible quand le nombre de paramètres est grand.

Une autre approche est alors d'avoir recours à la différentiation explicite du système d'équations discrétisé, ce qui exige un travail considérable d'analyse et de codage. Or, il se trouve que les logiciels de différentiation automatique ont gagné en performance et fiabilité, révélant le fort potentiel cette technique [63]. La différentiation automatique est le procédé qui, à partir d'un code informatique évaluant une fonction f, produit un code qui évalue les valeurs exactes (aux erreurs d'arrondi près) des dérivées partielles de f. Cette technique utilisée en mode adjoint présente l'avantage de calculer les dérivées de la fonctionnelle à minimiser en un temps indépendant du nombre de paramètres par rapport auxquels on dérive. De plus, les dérivées calculées ne posent pas de problèmes de consistance avec les équations d'état résolues, contrairement à celles issues des autres techniques. Enfin, la différentiation automatique permet à l'utilisateur de supprimer facilement n'importe quelle partie du gradient, et donc d'analyser les différentes contributions des parties du code de simulation. Dans le même esprit, la prise en compte des modifications du code à différentier pour le code adjoint est plus aisée.

La différentiation automatique paraît donc pouvoir fournir avec le mode adjoint un outil précis, rapide et simple de mise en œuvre, d'obtention du gradient d'une fonctionnelle dépendant d'un grand nombre de paramètres. En contrepartie, comme c'est le cas lors de la résolution de l'équation adjointe, on a besoin pour le calcul du gradient de sauvegarder toutes les parties de la trajectoire intervenant non linéairement dans le code direct ce qui représente *grosso modo* tous les états intermédiaires par lequel passe le système durant la simulation. Ceci a pour effet de rendre la méthode *a priori* ingérable lors du traitement de cas industriels.

En réalité, dans le cadre d'un problème stationnaire, une propriété intéressante précisément propre à l'optimisation stationnaire permet de réduire significativement les sauvegardes effectuées. De même, en optimisation transitoire, l'ap-

proche du calcul de l'adjoint par morceaux a donné naissance à des algorithmes très efficaces qui laissent entrevoir des perspectives intéressantes en contexte industriel.

Notre travail a pour but de confirmer et d'étayer ces idées générales sur l'étude de deux problèmes issus de la thermique. Ces problèmes sont simplifiés par rapport au problème réel mentionné ci-dessus (chauffage de l'habitacle du véhicule) en ce qu'ils s'affranchissent de la plupart des artefacts techniques (on travaille sur une géométrie simplifiée avec des conditions aux bords canoniques), mais ils sont d'une taille raisonnable de sorte de donner une bonne idée de la faisabilité de l'approche sur le problème industriel complet.

2 Cas traités

Nous avons mis en œuvre les techniques d'optimisation qui seront exposées à la section suivante sur deux problèmes de thermique, en traitant l'écoulement fluide comme un paramètre (une étape ultérieure consisterait à considérer le problème couplé thermohydraulique où seule la vitesse du fluide sur les bords est traitée comme un paramètre, la vitesse à l'intérieur du domaine étant *calculée*). Le problème industriel étant trop compliqué, nous choisissons de traiter ces deux problèmes simplifiés dont les ordres de grandeur, s'ils ne sont pas ceux du problème initial, permettent malgré tout de se rendre compte des difficultés spécifiques à l'optimisation pour des problèmes de grande taille.

Plus précisément, la géométrie de l'écoulement est la même pour les deux problèmes. Il s'agit d'une cavité parallépipédique (rectangle) sur laquelle on applique des conditions limites de Dirichlet (température fixée en entrée T_0) et de Neumann (cœfficient d'échange C_1 et température d'échange T_1). On appelle V_0 la vitesse de l'écoulement constante dans la cavité.

Problème 1 (stationnaire)

Soit Ω le domaine borné défini par la cavité ci-dessus dans lequel on cherche à connaître la température. On se donne quatre variables de contrôle (appelées les *paramètres* dans la suite):

- la température d'entrée T_0 (supposée homogène sur le bord Γ_1),
- la vitesse de l'écoulement V_0 (supposée homogène dans le domaine, colinéaire à $\vec{e_X}$),
- le cœfficient d'échange C_1 et la température d'échange T_1 sur la partie du bord Γ_3.

On résout le problème statique suivant :

$$(\mathcal{P}_1) = \begin{cases} -\Delta T_{stat} + V_0.\nabla T_{stat} = 0 \quad \text{sur} \quad \Omega, \\ T_{stat} = T_0 \quad \text{en entrée}, \\ \partial_n T_{stat} = 0 \quad \text{sur la surface latérale inférieure } \Gamma_2, \\ \partial_n T_{stat} = -C_1(T_{stat} - T_1) \quad \text{sur les autres surfaces latérales } \Gamma_3. \end{cases}$$

On veut optimiser les paramètres pour approcher le mieux possible l'état de référence décrit par une température constante appelée T_{ref} ($T_{ref} = 20$ degrés dans la pratique). Le critère s'écrit

$$J(T_0, V_0, C_1, T_1) = \| T_{stat} - T_{ref} \|^2_{L^2(\Omega)} .$$

Pour l'optimisation de ce critère, nous comparons trois calculs de gradient différents sur plusieurs algorithmes d'optimisation (de type Gradient Conjugué ou Quasi-Newton),

- approximation par différences finies,
- gradient donné par la différentiation automatique,
- formule simplifiée (spécifique au cas stationnaire [1]) pour le calcul du gradient automatique.

Problème 2 (dépendant du temps)

On considère l'état stationnaire T_{res} obtenu par un jeu "optimisé" de paramètres de contrôle. On note T_0^{opt}, V_0^{opt}, C_1^{opt} et T_1^{opt} ces paramètres.

A $t = 0$, on se donne arbitrairement un nouveau jeu de ces quatre paramètres. A $t = 100$, on souhaite avoir l'état de température aussi proche possible que l'état T_{res}. Pour les tests numériques, nous avons considéré ce problème indépendamment du premier. Ainsi, on a considéré des valeurs "banales" pour les deux jeux de variables (T_0, V_0, C_1, T_1) à $t = 0$ et $t = 100$. Entre les deux, on a 396 valeurs qui sont "libres" : elles correspondent aux valeurs des 4 paramètres aux 99 instants $t = 1, 2, \ldots, 99$. On optimise sur ces 396 valeurs pour être aussi proche que possible à $t = 100$ de l'état stationnaire cible.

Pour cela, on résout le problème d'évolution suivant

$$(\mathcal{P}_2) = \begin{cases} \partial_t T + V_0(t).\nabla T = \Delta T \quad \text{sur} \quad \Omega, \\ T(t) = T_0(t) \quad \text{en entrée}, \\ \frac{\partial T}{\partial n}(t) = 0 \quad \text{sur la surface latérale inférieure}, \\ \frac{\partial T}{\partial n}(t) = -C_1(t)\Big(T(t) - T_1(t)\Big) \quad \text{sur les autres surfaces latérales.} \end{cases}$$

Afin de tenir compte du coût du contrôle, on ajoute au critère quatre termes supplémentaires (norme H^1 des quatre contrôles). On a préféré la norme H^1 pour obtenir des courbes de contrôle plus régulières. Plus précisément, le critère utilisé est

$$J\Big(\Big(T_0(t)\Big)_{t\in[1,99]}, \Big(V_0(t)\Big)_{t\in[1,99]}, \Big(C_0(t)\Big)_{t\in[1,99]}, \Big(T_1(t)\Big)_{t\in[1,99]}\Big) =$$
$$\| T(100) - T_{res} \|_{L^2(\Omega)}$$
$$+\alpha\Big(\| T_0(t) \|^2_{H^1} + \| V_0(t) \|^2_{H^1} + \| C_1(t) \|^2_{H^1} + \| T_1(t) \|^2_{H^1} \Big).$$

Plusieurs algorithmes d'optimisation seront testés sur ce problème, sachant que le gradient est toujours calculé par différentiation automatique, pour deux

1. voir le détail de cette technique à la section 4.

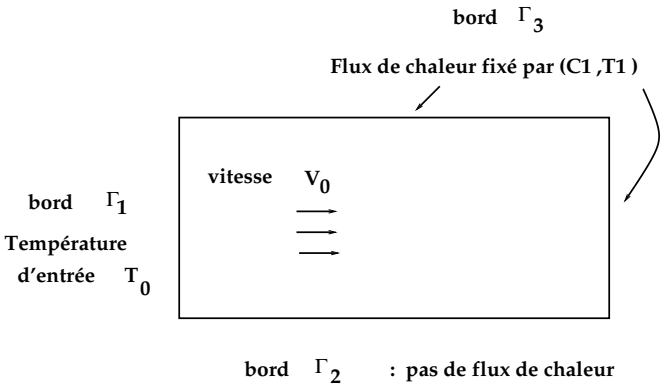


FIG. 1 – **Schéma des conditions aux limites. La boite parallélépipédique est 3D. Les bords sont partitionnés en Γ_1 (bord latéral gauche d'entrée du flux à température homogène T_0), Γ_2 (bord inférieur, pas de flux de chaleur), Γ_3 (4 autres parois latérales, flux de chaleur fixé par C_1, T_1).**

valeurs de pénalisation α dans la fonctionnelle critère J, $\alpha = 10^{-2}$ et $\alpha = 10^{-4}$. Le calcul par différences finies n'est pas envisageable dans ce cas, et l'approche de discrétisation des équations adjointes est laissée de côté pour des raisons que nous détaillerons plus loin.

Pour chacun de ces deux problèmes, le calcul de l'état pour un jeu de paramètres donnés est effectué par le code industriel PAMFLOW^{TM} [2]. Plus précisément, la résolution se fait par éléments finis sur un maillage tétraédrique non structuré. Pour le cas stationnaire, le laplacien est inversé par Gradient Conjugué. Dans le cas dépendant du temps, le schéma de discrétisation en temps est un schéma prédicteur-correcteur. La partie implicite de diffusion est aussi inversée par Gradient Conjugué. Le maillage défini sur notre géométrie comporte 4500 nœuds et 20100 éléments.

Signalons que nous résoudrons ces deux problèmes en tant que problèmes d'optimisation sans contraintes. Des contraintes de type bornes inférieures existent (température, vitesse positives), mais elles seront prises en compte par une projection dans la recherche linéaire suivant le calcul de la direction de descente et non par un algorithme sous contraintes particulier.

Pour résoudre les deux problèmes abordés, il existe plusieurs grandes classes de méthodes qui regroupent chacune de nombreuses heuristiques. On se propose de faire une brève revue de ces méthodes dans la section suivante. Comme pour les deux problèmes traités on suppose la fonction critère au moins deux fois différentiable, on insiste plus sur les méthodes de gradient qui sont le centre de notre travail. Les méthodes n'utilisant pas le gradient sont brièvement

2. code commercialisé par ESI Group.

évoquées. Les méthodes spécifiques au cas non différentiable sont, elles, laissées complètement de côté. On renvoie le lecteur par exemple à [4] pour une description didactique de ces méthodes. Bien entendu, dans l'exposé que nous faisons de toutes ces méthodes d'optimisation, comme dans l'exposé que nous ferons à la section 4 des stratégies de calcul du gradient, nous ne prétendons à aucune exhaustivité. Il s'agit simplement de situer les méthodes sur lesquelles notre choix se fixera ensuite dans l'ensemble des méthodes que nous avons pu trouver dans la littérature.

En quatrième partie, nous donnons une sommaire description des différentes techniques développées pour le calcul du gradient. Nous consacrons un traitement un peu plus exhaustif à la différentiation automatique utilisée en mode adjoint, technique que nous avons choisi de mettre en œuvre. Nous indiquerons les éléments qui permettent de gérer le problème de taille mémoire inhérent aux approches adjointes.

En cinquième partie, les résultats numériques obtenus pour les deux problèmes décrits précédemment sont présentés. Le problème statique nous a permis de comparer l'approximation par différences finies (possible quand les paramètres sont peu nombreux) par rapport au gradient issu de la différentiation automatique. Dans le même temps, nous testons aussi différents algorithmes d'optimisation. En revanche, le second problème comprenant 396 variables de contrôle n'est pas traitable si on approche les dérivées partielles par différences finies. Sur ce problème, on a donc comparé plusieurs méthodes d'optimisation à gradient donné (calculé par différentiation automatique).

On résume les conclusions de notre travail en sixième partie. Précisons tout de suite que ces conclusions n'ont d'autre but que de montrer *sur les cas que nous avons mis en œuvre* la supériorité de certaines approches sur certaines autres. Nous espérons que ces conclusions pourront être des enseignements transposables à d'autres contextes, mais nous ne prétendons à aucune généralité. Ces conclusions sont suivies de perspectives qui donnent une idée des travaux à réaliser pour compléter les quelques résultats présentés dans cet article. De plus, on y lance quelques idées qui motivent des études plus poussées.

3 Présentation de quelques méthodes d'optimisation

Dans le domaine de l'optimisation, on dispose *a priori* d'un grand nombre de méthodes. Une hiérarchie générale est impossible, l'efficacité d'une méthode donnée variant selon les problèmes traités et les besoins de l'utilisateur. Cependant, en optimisation sans contraintes on peut distinguer quatre grandes classes.

3.1 Méthodes stochastiques

Les méthodes stochastiques tiennent une place majeure en optimisation non différentiable, en particulier pour les problèmes combinatoires. Elles sont aussi d'un excellent recours pour les problèmes continus qui mettent en jeu une fonction critère présentant un nombre considérable de minima locaux (recherche de configuration optimale en chimie moléculaire [84]). En effet, elles possèdent la propriété de convergence globale[3] vers l'optimum global. On distingue principalement quatre grands types d'algorithmes relevant de ces méthodes.

Tout d'abord, les algorithmes génétiques tentent de simuler le processus d'évolution naturelle dans un environnement hostile. Chaque solution du problème, ou individu, est codée par une chaîne de bits finie à laquelle est associée une "fitness" égale au critère en cette solution. Ensuite, des populations d'individus sont générées itérativement en appliquant des processus de sélection, de croisement et de mutation qui se basent sur la "fitness" des individus [40].

L'algorithme de recuit simulé (ou "simulated annealing") est né d'une analogie thermodynamique avec le refroidissement d'un métal. Le refroidissement lent et régulier d'un métal permet aux atomes de se stabiliser peu à peu dans une position d'énergie minimale en dépit du nombre immense de configurations que peuvent prendre ces atomes. Metropolis a proposé d'introduire un paramètre ou température qui diminue au cours de l'optimisation. Au début du processus, la température élevée autorise les transitions vers un état d'énergie plus élevée. Au cours du processus, la température diminuant, la transition vers un état d'énergie plus élevée devient de plus en plus improbable [56, 57, 58].

La méthode Tabou est une procédure heuristique moins populaire que les deux précédentes. Elle suppose qu'on puisse définir un voisinage de solutions pour chaque solution. A chaque itération, la procédure se déplace vers la solution du voisinage $N(x_k)$ de x_k qui diminue au mieux la fonction critère. Alors que la plupart des méthodes d'exploration ne gardent comme information que la valeur minimale de la fonction critère calculée jusqu'alors, cette procédure garde en mémoire les solutions rencontrées ou plus généralement l'itinéraire effectué lors des dernières itérations dans une liste dite tabou [39]. La liste tabou a pour rôle d'interdire le choix des déplacements (respectivement des solutions) à ceux (respectivement celles) qui ramènent à une solution visitée précédemment. Ainsi, l'algorithme peut s'échapper des minima locaux. Au cours de la procédure, la liste tabou ainsi que les mouvements admissibles pour la solution sont régis par des processus d'intensification, de diversification et d'aspiration [94].

Enfin, la technique appelée GRASP (ou "Greedy Randomized Adaptative Search Procedures) est une heuristique très simple utilisée surtout pour les problèmes combinatoires. Il s'agit d'un processus itératif, qui à chaque itération répète les mêmes phases. La première phase est une phase de construction pendant laquelle une solution réalisable est exhibée. La seconde consiste à réaliser une optimisation locale à partir de la solution construite . La meilleure solution est gardée en mémoire jusqu'à ce que le processus soit arrêté (stagnation, nombre maximal d'itérations atteint) [93].

3. c'est-à-dire qu'elles convergent quelque soit la donnée initiale.

Pour mieux tirer parti des avantages de chaque méthode, des méthodes hybrides ont été développées ainsi que des algorithmes parallélisés [90].

Une méthode stochastique requiert au minimum quelques centaines de simulations directes. Dans notre cas où chaque évaluation de la fonction est coûteuse, typiquement quelques heures de calcul sur un supercalculateur, une telle méthode est donc inutilisable. On peut mentionner toutefois quelques exemples d'utilisation d'algorithmes génétiques en Optimisation de forme pour des formes définies par quelques dizaines de nœuds [33, 74].

3.2 Méthodes de gradient

Les méthodes de gradient interviennent lorsque la fonction à optimiser est différentiable (ou qu'on la postule comme telle ...). Elles utilisent les informations données par les dérivées partielles de f pour calculer les itérés du processus, ce qui a pour objectif d'économiser sur le nombre total d'évaluations de la fonction. Dans notre cas, on suppose que la fonction f est différentiable au moins deux fois.

Un algorithme de minimisation consiste à chaque itération en la succession de 3 étapes [27]

- le choix d'un modèle local,
- le choix d'une direction de descente,
- le choix d'une longueur de descente.

Certains algorithmes effectuent les deux dernières étapes simultanément. Ce sont les algorithmes de régions de confiance. Dans la suite, on donne une description sommaire des principaux algorithmes correspondant à ces étapes, émaillée de nombreuses références bibliographiques. On s'efforcera de faire ressortir les avantages et les inconvénients génériques de chaque méthode, et on mentionnera si oui ou non nous avons mis en œuvre la méthode sur notre cas précis.

3.2.1 Calcul de la direction de descente

On peut distinguer trois grands groupes de techniques. Le premier est constitué des méthodes dites du premier ordre car seules les informations relatives au gradient de la fonction sont utilisées. Le second est constitué des méthodes dites du second ordre car les informations relatives au gradient et au hessien de la fonction sont prises en compte. Enfin, le troisième est constitué de méthodes qu'on peut appeler de pseudo-second ordre car elles utilisent une approximation du hessien de la fonction ou de son inverse calculée à partir d'informations du premier ordre.

3.2.1.1 Méthodes du premier ordre

La première démarche consiste à approcher localement la fonction critère par le modèle d'ordre 1

$$f(x_k + d_k) = f(x_k) + \bigtriangledown f(x_k).d_k + o(\| d_k \|). \quad (1)$$

La diminution la plus grande de la fonction est obtenue pour la direction

$$d_k = - \bigtriangledown f(x_k) : \text{ c'est la méthode de plus profonde descente.}$$

On a la convergence vers un point stationnaire mais de manière très lente dès qu'on se trouve proche d'un minimum.

Une autre méthode est celle de Hooke & Jeeves. On effectue des minimisations par rapport à une seule variable à la fois. On peut employer pour chacune de ces optimisations un algorithme de plus profonde descente ou un algorithme de Newton. Cette méthode se révèle très mauvaise dans la pratique [4].

Les méthodes du premier ordre bien que plus simples que les méthodes du second ordre ont l'avantage de nécessiter peu d'espace mémoire (pas de stockage de matrices), ce qui les rend incontournables en très grande dimension. C'est pourquoi le Gradient Conjugué non Linéaire représente une alternative encore très répandue. Il s'agit d'une généralisation du Gradient Conjugué Linéaire dans laquelle on effectue une recherche linéaire à chaque itération.

$$\begin{cases} d_0 = - \bigtriangledown f(x_0), \\ d_{k+1} = - \bigtriangledown f(x_k) + \beta_k d_k, \\ x_{k+1} = x_k + \alpha_k d_k. \end{cases}$$

Dans la cas d'une fonction linéaire, la conjugaison des directions d_k et la recherche linéaire exacte induit un choix unique des paramètres β_k et α_k. Dans le cas d'une fonction non linéaire, cela n'est plus possible. On a alors recours à des formules spécifiques pour β_k et à une recherche linéaire pour le calcul de α_k.

On dénombre trois choix classiques de β_k. Nous les avons tous testés sur nos deux problèmes.

- méthode de la plus profonde descente,

$$\beta_k = 0,$$

- formule de Fletcher et Reeves,

$$\beta_k^{FR} = \frac{\| \bigtriangledown f(x_k) \|_2}{\| \bigtriangledown f(x_{k-1}) \|_2},$$

- formule de Polak et Ribière,

$$\beta_k^{PR} = \frac{\Big(\bigtriangledown f(x_k) - \bigtriangledown f(x_{k-1}) \Big)^T . \bigtriangledown f(x_k)}{\| \bigtriangledown f(x_{k-1}) \|_2},$$

Pour assurer la convergence globale de l'algorithme de Polak & Ribière associé à une recherche linéaire vérifiant les conditions de Wolfe, on a aussi testé $\beta_k = max(\beta_k^{PR}, 0)$ [36]. Par la suite, on appelera cet algorithme l'algorithme de Polak & Ribière stabilisé. On dispose aussi d'algorithmes de recherche linéaire particuliers pour obtenir la convergence globale de l'algorithme de Polak-Ribière [50].

3.2.1.2 Méthodes du second ordre et du pseudo second ordre

Une approche, plus puissante que les méthodes du premier ordre, due à Newton consiste à trouver un point qui annule le gradient. Pour cela, on considère un développement limité à l'ordre 1 du gradient

$$\bigtriangledown f(x_k + d_k) = \bigtriangledown f(x_k) + \bigtriangledown^2 f(x_k).d_k + o(\| d_k \|)$$

$$\bigtriangledown f(x_k + d_k) = 0 \Longrightarrow d_k = -\Big(\bigtriangledown^2 f(x_k) \Big)^{-1} . \bigtriangledown f(x_k)$$

Bien évidemment, le choix de cette direction revient à considérer pour f le modèle local quadratique défini par

$$f(x_k + d_k) = f(x_k) + \bigtriangledown f(x_k).d_k + d_k^T . \bigtriangledown^2 f(x_k).d_k, \qquad (2)$$

et à en rechercher un point critique.

Le calcul de la direction de descente dans les méthodes du second ordre et du pseudo second ordre est basé sur cette direction d_k, dite de Newton, qui entraîne un taux de convergence superlinéaire.

3.2.1.2.1 Décomposition du hessien

Les méthodes basiques consistent à choisir une factorisation pour résoudre exactement le système linéaire de Newton. Quand la matrice n'est pas définie positive, on utilise une factorisation de Cholesky modifiée (Gill & Murray [38], Schnabel & Eskow [99]) ou une factorisation de Bunch & Parlett [6]. Dans la pratique, ces méthodes sont très peu utilisées car soit le hessien est trop coûteux, soit la dimension est trop importante pour mettre en œuvre ces factorisations. De plus, le calcul de la direction de Newton de façon précise ralentit considérablement l'algorithme.

Comme les deux problèmes que nous traiterons mettent en jeu "peu" [4] de paramètres de contrôle, le coût de ces décompositions est très faible. On a simplement utilisé une décomposition LU quand on avait à inverser l'approximation du hessien.

3.2.1.2.2 Les méthodes de Quasi-Newton

Lorsque le calcul du hessien est très lourd, il est possible de l'approximer à l'aide de matrices dites de Quasi-Newton. On se donne une matrice initiale définie positive (généralement égale à l'identité) que l'on met à jour à chaque itération de l'algorithme afin d'approcher de mieux en mieux le hessien H ou l'inverse du hessien B.

4. dans le cas du problème 2, on a de l'ordre de 400 paramètres, mais cela reste faible par rapport au but industriel que nous nous sommes fixés.

La mise à jour de l'approximation du hessien H_k (ou de son inverse B_k) est généralement la solution du problème d'optimisation sous contraintes général

$$\min \omega(H_{k+1}, H_k, B_k, B_{k+1}),$$

$$sachant \begin{cases} H_{k+1} \quad \text{symétrique,} \\ H_{k+1}.(x_{k+1} - x_k) = \bigtriangledown f(x_{k+1}) - \bigtriangledown f(x_k). \end{cases}$$

La deuxième contrainte est appelée équation de la sécante vérifiée par la moyenne du hessien sur le morceau de droite compris entre les points x_k et x_{k+1}. La fonction ω désigne une mesure sur l'espace des matrices. Cette mesure est très souvent liée aux valeurs propres de la matrice H_{k+1}. Ainsi, on améliore le conditionnement de la matrice H_{k+1} qui doit être "inversée" pour la résolution du système de Newton. En optimisation sans contraintes, on a plutôt besoin de l'inverse du hessien. On utilise alors dans la majeure partie des cas les formules mettant à jour l'inverse du hessien B_k. On s'affranchit alors d'une résolution matricielle à chaque itération.

Par ailleurs, quand on travaille avec le hessien, on utilise le plus souvent des mises à jour qui préservent la définie positivité de la matrice de Quasi-Newton, afin de faciliter la résolution du système matriciel de Newton. C'est le cas de la plupart des mises à jour qui sont citées plus loin. Notamment, la mise à jour de BFGS vérifie cette propriété. En revanche, la mise à jour de rang 1 (SR1) n'assure pas le caractère défini positif des matrices itérées [85]. Cette mise à jour s'avère assez médiocre quand on utilise une recherche linéaire classique. Toutefois, elle semble pertinente dans un autre contexte qui est celui des régions de confiance dont nous décrivons le principe plus loin.

3.2.1.2.3 Différentes mises a jour

Plusieurs mesures ω ont été étudiées [109]. Ainsi, il a été défini quelques grandes classes de mise à jour dont la plus connue est la classe convexe de Broyden. On dispose des formules générales suivantes pour les matrices H_k et B_k approximant le hessien et son inverse :

$$H_{k+1} = H_k + \frac{y.y^T}{s^T y} - \frac{H_k s s^T H_k}{s^T H_k s} + \frac{\beta}{s^T H_k s}\Big(\frac{s^T H_k s}{s^T y} y - H_k s\Big)\Big(\frac{s^T H_k s}{s^T y} y - H_k s\Big)^T,$$

$$B_{k+1} = B_k + \frac{s.s^T}{s^T y} - \frac{B_k y y^T B_k}{y^T B_k y} + \frac{\eta}{y^T B_k y}\Big(\frac{y^T B_k y}{s^T y} s - B_k y\Big)\Big(\frac{y^T B_k y}{s^T y} s - B_k y\Big)^T,$$

où

$$\eta\beta\Big(\frac{y^T B_k y}{s^T y}.\frac{s^T B_k^{-1} s}{s^T y} - 1\Big) + \eta + \beta = 1.$$

Dans la suite, à une exception près, on utilisera les formules de type approximation de l'inverse du Hessien B_k, et non les formules d'approximation du

Hesssien H_k lui-même. La mise à jour de Davidon, Fletcher et Powell ou DFP, qui fut la première proposée, correspond à $\eta = 0$ (ou $\beta = 1$). Pour $\eta = 1$ (ou $\beta = 0$), on obtient la très répandue mise à jour de Broyden, Fletcher, Godfarb et Shanno ou BFGS. Cette mise à jour semble en moyenne la plus efficace. On peut citer les mises à jour classiques de Greenstatd, SR1 (de rang un) [25], et la mise à jour de Fletcher (η négatif) qui fait partie des mises à jour prometteuses car les grandes valeurs propres y sont mieux corrigées en comparaison de BFGS [27].

Ces algorithmes se révélent assez lents pour des problèmes mal conditionnés car la matrice initiale est très différente du hessien. C'est pourquoi, on peut introduire deux nouvelles formules générales pour les matrices H_k et B_k qui dépendent de deux paramètres supplémentaires ρ et γ:

$$H_{k+1} = \Big(\frac{1}{\gamma}\Big)\left(H_k + \frac{\gamma}{\rho}.\frac{y.y^T}{s^T y} - \frac{H_k s s^T H_k}{s^T H_k s} + \frac{\eta}{s^T H_k s}\Big(\frac{s^T H_k s}{s^T y} y - H_k s\Big)\Big(\frac{s^T H_k s}{s^T y} y - H_k s\Big)^T\right),$$

$$B_{k+1} = \gamma\left(B_k + \frac{\rho}{\gamma}.\frac{s.s^T}{s^T y} - \frac{B_k y y^T B_k}{y^T B_k y} + \frac{\eta}{y^T B_k y}\Big(\frac{y^T B_k y}{s^T y} s - B_k y\Big)\Big(\frac{y^T B_k y}{s^T y} s - B_k y\Big)^T\right).$$

Le paramètre ρ est un paramètre de stabilisation introduit par Biggs [3] et γ est un paramètre d'échelle (scaling) introduit par Oren [88]. Luksàn a répertorié les différentes heuristiques pour le choix des paramètres η, ρ, γ dans les articles [64, 66]. Il a procédé à une comparaison numérique exhaustive des différentes mises à jour [65]. On peut aussi se référer aux articles suivants pour d'autres mises à jour [19, 69, 109]. On peut mentionner les travaux de Ford & Moghrabi sur une méthode multi-pas [30, 31, 29].

Le gradient de la fonction critère est une fonction de $\mathbb{R}^n$ dans $\mathbb{R}^n$ si on note n le nombre de paramètres de contrôle du problème. Ainsi, le calcul du hessien demande n calculs de gradient soit n calculs adjoints si on utilise la différentiation automatique en mode direct ou en mode adjoint. On se rend donc compte que le nombre de calculs de gradient nécessaire à l'évaluation du hessien (à chaque itération) est trop pénalisant pour les problèmes industriels.

Par ailleurs, la différentiation du code adjoint construit demande un travail supplémentaire important pour les codes industriels. De plus elle nécessiterait dans le cas des problèmes d'évolution, la sauvegarde de données supplémentaires. Même si cela serait encore abordable pour les deux problèmes simplifiés étudiés, ceci ne le serait plus pour des problèmes industriels de grande taille. Enfin, on rencontre les mêmes problèmes de taille mémoire et de temps de calcul avec l'écriture des équations de l'adjoint du second ordre qui semble encore plus compliquée à mettre en œuvre [68].

En vue du traitement de problèmes industriels de grande taille, l'utilisation d'une matrice de Quasi-Newton semble incontournable pour la mise en œuvre d'une méthode de second ordre. On a donc implémenté toutes les mises à jour mentionnées précédemment avec et sans paramètres de stabilisation pour les deux problèmes étudiés. Les résultats numériques obtenus seront riches d'enseignement.

3.2.1.2.4 Mise a jour en jouant sur les colonnes Une autre stratégie pour la mise à jour de la matrice de Quasi-Newton consiste en la factorisation de l'approximation de l'inverse du hessien B_k selon

$$B_k = Z_k.Z_k^T.$$

En procédant à des rotations orthogonales sur Z_k, on essaye de réduire la possibilité d'avoir une sur-estimation du hessien. Différentes stratégies de "scaling" sur les colonnes de Z_k ont été proposées mais trop peu de tests semblent avoir été réalisés pour arriver à une conclusion [67, 100]. Nous n'avons pas nous-mêmes testé cette mise à jour.

3.2.1.2.5 La méthode de Quasi-Newton à mémoire limitée

Il arrive fréquemment qu'en grande dimension la mémoire disponible soit insuffisante pour stocker la matrice de Quasi-Newton. On remplace alors la formation et le stockage de cette matrice par la sauvegarde d'un nombre fixe m de paires de vecteurs $\{s_k, y_k\}$. Ces paires de vecteurs permettent par l'intermédiaire d'une relation de récurrence de calculer le produit de la matrice de Quasi-Newton avec n'importe quel autre vecteur [86]. Concernant le choix de la valeur de m, une valeur entre 3 et 10 donne les meilleurs résultats [85]. Les formules de calcul induites par cette méthode ont surtout été développées pour la mise à jour BFGS, mais on peut aussi les étendre à la mise à jour SR1. Il existe aussi, comme pour la méthode de Quasi-Newton, des variantes basées sur des stratégies de "scaling" et "sizing" [34] et des représentations de la matrice de Quasi-Newton compactes plus pertinentes pour l'optimisation sous contraintes [7].

Les matrices de Quasi-Newton à mémoire limitée se révèlent aussi efficaces pour former des préconditionneurs utiles lors de la résolution du système linéaire de Newton par une méthode itérative [76]. L'algorithme de Buckley & LeNir utilise ce principe pour préconditionner le Gradient Conjugué [5, 71].

Nos deux problèmes font intervenir un trop petit nombre de variables pour nécessiter expressément le recours à cette méthode. Cependant, en vue du traitement des problèmes de plus grande taille en nombre de variables (m paramètres par nœud du maillage), l'implémentation d'une méthode de Quasi-Newton à mémoire limitée ou l'utilisation de l'algorithme M1QN3 (développé par l'INRIA) fait partie des développements en cours.

3.2.1.2.6 Gauss-Newton

La méthode de Gauss-Newton s'applique au cas où la fonction à minimiser est de la forme

$$f(x) = \frac{1}{2}\sum_{i=1}^{m} r_i^2(x).$$

Au minimum, généralement nul, les fonctions $r_i(x)$ sont négligeables par rapport à leur gradient. On peut alors approximer le hessien de f par

$$\nabla^2 f(x) \simeq J.J^T,$$

avec J la matrice de terme général $J_{i,j} = \dfrac{\partial r_i}{\partial x_j}$.

Pour les fonctionnelles quadratiques faiblement non linéaires, cette méthode est meilleure que la méthode de Quasi-Newton. En contrepartie, on a besoin de calculer J, ce qui s'avère très coûteux quand m est grand. A ce propos, il est préferable que m soit plus grand que le nombre de paramètres afin de travailler avec une matrice de rang maximal. Pour le problème $\mathcal{P}_2$, concernant la partie thermique du critère, m serait le nombre de nœuds du domaine (il serait le nombre de nœuds de la surface de sortie pour le problème $\mathcal{P}_1$). Concernant la partie énergétique du critère, m est serait égal à 99.

De plus, l'application de l'algorithme de Gauss-Newton avec $m = 1$ implique une approximation du hessien de rang 1, ce qui n'est pas envisageable. La méthode de Gauss-Newton semble donc inadéquate pour le genre de problèmes que nous voulons aborder.

Soulignons aussi que la qualité de la méthode semble se dégrader lorsque l'approximation du hessien est singulière. Pour remédier à cela, on peut effectuer une régularisation de Tichonov [41] qui ressemble à l'introduction du cœfficient de Lagrange dans la méthode des régions de confiance.

3.2.1.2.7 Iterated Subspace Minimization Methods (ISM)

En grande dimension, le coût algébrique de la résolution de l'équation de Newton est prédominant par rapport aux évaluations de la fonction et du gradient lorsque celles-ci sont peu coûteuses. L'approche employée est alors de diminuer ce coût algébrique par des évaluations de fonctions supplémentaires. Dans le même esprit que des travaux de Saad [95], la résolution partielle ou non du système de Newton est suivie d'une minimisation de la fonction sur un espace de petite dimension (typiquement entre 10 et 20) bien choisi, pour laquelle on possède des méthodes robustes et rapides [14].

Dans notre contexte où nous cherchons justement à économiser le nombre d'évaluations de fonction et de gradient, cette approche ne semble pas pertinente.

3.2.1.2.8 Newton tronqué

En grande dimension, il résulte un fort coût algébrique de la résolution du système linéaire de Newton à chaque itération. Or, le calcul exact de la direction de Newton ne prend son intérêt que si on se trouve proche d'un minimum, là où le modèle quadratique constitue une excellente approximation de la fonction critère. On peut donc se contenter d'un calcul approché pendant une grande partie du processus d'optimisation. Pour cela, la résolution du système linéaire par

le Gradient Conjugué qu'on tronque dès qu'une précision suffisante est atteinte semble des plus naturelles : c'est la méthode de Newton tronqué [17].

Lorsque la matrice à inverser est définie positive, le Gradient Conjugué (préconditionné ou non) est stable. Un exemple de critère de troncature est

$$d_k \quad \text{vérifie} \quad \| \bigtriangledown^2 f(x_k).d_k + \bigtriangledown f(x_k) \| \leq \eta_k \| \bigtriangledown f(x_k) \|, \eta_k \in [0,1].$$

Quand la matrice à inverser n'est pas définie positive, le Gradient Conjugué peut devenir instable. On a alors le choix de tronquer dès qu'une direction de courbure négative est détectée ou, dès que l'itéré courant de l'algorithme itératif sort d'une sorte de région de confiance [4]. Une meilleure alternative consiste à continuer le processus itératif à l'aide de l'algorithme de Lanczos [89] dès qu'une direction de courbure négative est détectée, afin de calculer avec une meilleure précision la direction de descente [73]. Ces méthodes ne sont vraiment compétitives que si on préconditionne l'algorithme itératif considéré. Le choix du préconditionneur joue en règle générale un rôle crucial pour cet algorithme. Celui-ci peut être formé à partir d'une factorisation du hessien (ou de son approximation), ou de l'application d'un algorithme de Quasi-Newton à mémoire limtée en utilisant les directions intermédiaires générées par l'algorithme itératif au cours de la première résolution partielle du système de Newton.

La méthode de Newton tronqué est très efficace quand le problème traité est presque quadratique. Pour la rendre aussi performante dans le cas de problèmes fortement non linéaires, on peut la coupler avec une recherche non monotone et curvilinéaire, l'algorithme étant alors performant si le calcul des directions de courbure positive et négative est assez précis. Cependant, il faut mentionner une détérioration de l'algorithme lorsque seulement une approximation du hessien est considérée.

3.2.1.2.9 Méthode de Newton Discret

La méthode de Newton Discret constitue un intermédiaire entre les méthodes de Quasi-Newton et la méthode de Newton. Le hessien est approché par différences finies selon

$$\bigtriangledown^2 f(x).h \simeq \frac{\bigtriangledown f(x+\epsilon h) - \bigtriangledown f(x)}{\epsilon}.$$

On peut tronquer cette méthode (Newton Discret Tronqué) et prendre en compte des directions de courbure négative. On améliore ainsi la convergence et la robustesse de l'algorithme lors du traitement de problèmes mal conditionnés.

3.2.1.2.10 Méthode de Newton Discret Tronqué avec mémoire

La méthode de Newton Discret Tronqué, malgré sa robustesse pour les problèmes mal conditionnés, reste beaucoup plus coûteuse qu'une méthode de Quasi-Newton à mémoire limitée. A l'opposé, l'algorithme de BFGS à mémoire

limitée n'est pas aussi rapide et robuste lors du traitement de problèmes mal conditionnés que l'algorithme de Newton Discret Tronqué. Ces deux algorithmes complémentaires ont été associés pour former la méthode de Newton Discret Tronqué avec mémoire limitée afin de capter les avantages de chaque méthode [8]. L'algorithme consiste à effectuer périodiquement quelques itérations de Newton Discret Tronqué dans un algorithme de Quasi-Newton à mémoire limitée. Le Gradient Conjugué tronqué réalisé dans la phase discrète permet d'isoler des vecteurs qui servent à la mise à jour de la matrice de Quasi-Newton. Des résultats numériques sur deux algorithmes relevant de cette méthode, DINEMO et ALTERNATE, font ressortir la qualité des informations issues des phases de Gradient Conjugué.

Les trois méthodes précédentes sont d'un intérêt réel dès que le nombre de paramètres est grand. Malheureusement, elles semblent se dégrader assez fortement dès que le hessien est approché, comme on le préconise pour nos problèmes. Toutefois, une extension de notre travail pourrait être d'implémenter la méthode de Newton tronqué avec une matrice de Quasi-Newton non définie positive en utilisant un mise à jour SR1 par exemple.

3.2.1.2.11 Cas des fonctions partiellement séparables En grande dimension, les algorithmes les plus récents essayent d'imiter la structure du hessien. Lorsque la structure n'est pas connue, les algorithmes de Quasi-Newton avec mise à jour creuse apportent des alternatives intéressantes [28], mais buttent encore sur quelques problèmes qui les rendent moins compétitifs que les méthodes ci-dessus [86].

Lorsque la struture de f est connue, ce qui ne rentre pas dans les cas que nous souhaitons traiter, le cas des fonctions partiellement séparables donne lieu à un algorithme très efficace (voir aussi les propriétés particulières lors de la mise en oeuvre de la différentiation automatique dans [13]). Une fonction est dite partiellement séparable si et seulement si elle est la somme de fonctions élémentaires $f_i(x)$ pour lesquels on a un espace invariant non trivial [85] c'est-à-dire

$$f(x) = \sum_{i=1}^{n_e} f_i\Big(U_i(x)\Big), \qquad dim(U_i) = n_i * n, n_i \in [1,5].$$

Ainsi, si on se donne une approximation de chaque hessien élémentaire $H_i \simeq \nabla^2\phi_i$, on peut construire une approximation du hessien de f par

$$\nabla^2 f \simeq H = \sum_{i=1}^{n_e} U_i^T.H_i.U_i.$$

Comme on met à jour à chaque itération chaque hessien élémentaire (de petite taille), on obtient au bout de quelques itérations une bonne approximation du hessien de f [48]. Ceci est bien plus efficace que la méthode de Quasi-Newton à mémoire limitée qui apporte des corrections successives de rang 2 sur la matrice

totale. Ainsi on a besoin, en grande dimension, de réaliser un grand nombre d'itérations pour obtenir une approximation auusi bonne du hessien de f.

Cette méthode, bien que très efficace, requiert de connaître les espaces invariants de f au préalable. Or, leur détection automatique est assez difficile, ce qui rend la méthode peu applicable dans l'industrie. On peut toutefois mentionner les travaux de Gay sur l'élaboration de tels logiciels [32].

3.2.1.3 Méthodes de tenseur

Ces méthodes contrairement aux autres préconisent un modèle local d'ordre trois ou quatre défini par

Ordre 3

$$\psi(x_c + d) = \bigtriangledown f(x_c) + \bigtriangledown^2 f(x_c).d + \frac{1}{2} T_c.d^2,$$

Ordre 4

$$\psi(x_c + d) = f(x_c) + \bigtriangledown f(x_c).d + \frac{1}{2} \bigtriangledown^2 f(x_c).d^2 + \frac{1}{6} T_c.d^3 + \frac{1}{24} V_c.d^4.$$

On calcule les tenseurs par interpolation des informations gardées sur p itérés précédents. Des tests ont montré que $p = 1$ donne d'aussi bons résultats que $p \geq 1$ [98]. On est amené à employer une stratrégie particulière lorsque la matrice approximant le hessien est singulière, en particulier lorsque son noyau est de rang 1. Malheureusement on ne le sait pas *a priori*. Ces méthodes, très performantes lorsque le noyau est de rang 1, peuvent entraîner des gains au niveau des évaluations de fonctions et de gradient allant jusqu'à 50%.

Pour nos deux problèmes, on a implémenté le calcul des directions par les modèles de rang 3 et 4, avec un hessien approché et de rang maximal car on utilise une mise à jour de la matrice de Quasi-Newton qui assure la définie positivité de toutes les matrices itérées. Au vu des résultats moyens obtenus dont nous ne présenterons pas le détail, il semble que ces méthodes soient vraiment conseillées lorsqu'on possède le vrai hessien.

3.2.2 Calcul de la longueur de descente

Comme cela a été dit précédemment, la technique la plus efficace est celle de Newton. Seulement, cette méthode ne se révèle vraiment efficace que si le point de départ est assez proche de l'optimum. De plus, elle ne garantit pas la convergence vers un minimum, même local. C'est pourquoi, les algorithmes précédents sont couplés à une recherche linéaire (ou remplacés par une méthode des régions de confiance, que nous verrons plus loin). Ainsi, on retrouve dans l'algorithme la propriété de convergence globale vers un minimum local.

On suppose connaître la direction de descente d_k au point x_k. La recherche linéaire consiste à trouver α_k de façon à diminuer la fonctionnelle suffisamment le long de cette direction. Ce "suffisamment" sera quantifié ci-dessous dans la

description des conditions dites de Wolfe, Armijo et Goldstein & Price. Ensuite, on procèdera à la mise à jour

$$x_{k+1} = x_k + \alpha_k d_k .$$

Les méthodes les plus simples consistent à choisir α_k constant ou à effectuer une recherche itérative en subdivisant l'intervalle initial sur α_k par la méthode de la section dorée (appelée plus communément "golden search method") ou par une recherche dichotomique. Ces méthodes "simples" sont généralement très gourmandes en temps de calcul et donc peu utilisées quand l'évaluation de la fonction est coûteuse.

3.2.2.1 Conditions de Wolfe

Les recherches linéaires les plus évoluées sont celles qui utilisent le gradient de la fonction [27]. Une des plus puissantes consiste à satisfaire les conditions dites de Wolfe. On exige alors

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \beta_1 \alpha_k \bigtriangledown f(x_k)^T . d_k , \tag{3}$$
$$\bigtriangledown f(x_k + \alpha_k d_k)^T . d_k \geq \beta_2 \bigtriangledown f(x_k)^T . d_k , \tag{4}$$

pour un couple β_1, β_2 appartenant à $[0, 1]$ fixé à l'avance.

Il arrive souvent qu'on préfère les conditions de Wolfe dites fortes. La deuxième condition (4) est alors remplacée par

$$\mid \bigtriangledown f(x_k + \alpha_k d_k)^T . d_k \mid \leq \beta_2 \mid \bigtriangledown f(x_k)^T . d_k \mid .$$

La première condition (3) ou condition de diminution suffisante (ou condition d'Armijo) permet de ne pas prendre des pas α_k trop grands et ainsi de ne pas se déplacer vers un minimum local où la valeur de la fonction f serait supérieure à la valeur de la fonction à l'itéré courant x_k. La seconde condition (4) ou condition de courbure interdit le choix de pas trop petits pouvant entraîner une convergence lente. Le choix $\beta_1 = 0,9$ et $\beta_2 = 0,0001$ est un choix standard pour ces paramètres. On peut citer l'algorithme de Moré & Thuente construit pour le calcul d'une longueur de descente vérifiant les conditions de Wolfe [78].

3.2.2.2 Condition d'Armijo

La recherche d'Armijo se propose de trouver un pas qui satisfasse seulement la condition de diminution suffisante précédente (3). Elle est dangereuse car le pas initial pris le plus souvent égal au pas de Newton $\alpha_k = 1$ peut s'avérer trop faible [4]. En revanche, elle présente l'avantage de ne pas entraîner de surcoût lié à un calcul de gradient qui nécessite un temps de calcul non négligeable.

3.2.2.3 Goldstein & Price

La recherche de Goldstein & Price représente un compromis entre la recherche d'Armijo et la recherche de Wolfe (forte). Comme le gradient est souvent coûteux et que d'autre part il est important de satisfaire la condition de courbure (4), au moins en un sens faible, on approxime le gradient intervenant dans la condition de courbure par une approximation aux différences finies selon

$$\nabla f(x_k + \alpha_k d_k) \simeq \frac{f(x_k + \alpha_k d_k) - f(x_k)}{\alpha_k}.$$

Il n'est alors pas plus coûteux d'utiliser cette méthode que la recherche d'Armijo. En revanche, on rappelle qu'une recherche de Wolfe coûte plus cher car elle nécessite un calcul de gradient supplémentaire à chaque itération de la recherche linéaire. Les trois recherches linéaires précédentes ont été implémentées et testées sur les deux problèmes modèles. On les a toutes couplées à une interpolation cubique dont le principe est donné ci-après.

En ce qui concerne le choix de la longueur de descente initiale, on choisit durant les cinq premières itérations de l'algorithme d'optimisation de prendre une longueur plus grande que la valeur standard. En effet, on se situe loin de l'optimum, on a donc intérêt à prendre des pas grands pour converger plus vite. Pour les itérations suivantes, on adopte la valeur 1.

3.2.2.4 Interpolation cubique et parabolique

Ces heuristiques jouent un rôle important dans l'algorithme de recherche linéaire lors du calcul de l'itéré α_{k+1} car elles réduisent le nombre d'itérations du processus d'optimisation (soit le nombre d'évaluations de fonction et de gradient) par rapport à une recherche itérative par pure dichotomie par exemple. Avec les valeurs de la fonction et de son gradient en certains points, on peut calculer les paramètres d'une cubique (ou d'une parabole) qui approxime de façon plus ou moins satisfaisante la fonction. On se sert alors du minimum de la cubique, facile à calculer, pour passer de α_{k+1}^i à α_{k+1}^{i+1}.

3.2.2.5 Recherche non monotone

Lorsqu'on considère la fonction de Rosenbrock [51], exemple canonique en optimisation, on observe que la méthode de Newton (non monotone) converge en sept itérations alors qu'une méthode de Newton avec recherche linéaire monotone a besoin de beaucoup plus d'itérations pour localiser le minimum. Ceci fait apparaître que pour certaines fonctions (par exemple celles qui comportent des "narrow curved valleys") une recherche linéaire idéale pour la méthode de Newton doit rendre un accroissement de la valeur de la fonction possible au cours du processus d'optimisation, tout en retenant une convergence globale. Partant de cette observation, plusieurs méthodes ont vu le jour. Notamment,

on peut imposer à la valeur de la fonction à chaque itéré de satisfaire une condition d'Armijo en considérant la valeur maximale de la fonction sur un nombre fixé d'itérés précédents [51, 52, 104].

Pour nos problèmes, nous n'avons pas testé une telle recherche. Comme nous sommes limités par le nombre d'itérations, nous pensons en effet ne pas pouvoir réellement profiter de la non monotonie pour converger vers de meilleures valeurs.

3.2.2.6 Recherche avec courbure négative La capacité d'un algorithme à converger vers un point critique vérifiant les conditions d'optimalité du second ordre dépend de son aptitude à sortir des régions non convexes [72]. Contrairement à la méthode des régions de confiance (que nous verrons ci-dessous), les algorithmes de Newton avec recherche linéaire n'ont pas cette aptitude car le hessien est souvent perturbé de façon à ce qu'il soit défini positif, ou la matrice de Quasi-Newton est par construction définie positive. Cependant, la recherche linéaire peut recouvrer cette propriété en utilisant une direction négative du hessien, quand il y en a une. Les quelques algorithmes basés sur cette idée se sont révélés plus robustes et économiques que les algorithmes de Newton classiques et comparables avec l'algorithme des régions de confiance tronqué implémenté dans LANCELOT [79]. En effet, ils nous donnent la capacité d'exploiter efficacement la non convexité locale de la fonction objectif.

Ces algorithmes calculent une direction de descente de courbure positive s_k ($s_k^T H_k s_k \geq 0$) et une direction de courbure négative d_k ($d_k^T H_k d_k \leq 0$). Ces deux directions peuvent donner lieu à une recherche linéaire classique: on teste les deux directions et on choisit celle qui correspond à la diminution la plus grande. On peut aussi utiliser les deux directions simultanément dans la recherche, c'est la recherche curvilinéaire [72]. L'itéré suivant se déduit de l'itéré courant par

$$x_{k+1} = x_k + \alpha_k^2 s_k + \alpha_k d_k, \alpha_k > 0.$$

Bien que plus simple, la recherche linéaire avec une des deux directions semble donner de meilleurs résultats que la recherche curvilinéaire [44].

Concernant le calcul des deux directions s_k et d_k, il peut se faire simultanément en décomposant le hessien. Cela reste trop coûteux en grande dimension. Les algorithmes les plus efficaces les calculent séparément. Un algorithme de Gradient Conjugué tronqué fournit s_k, et des itérations supplémentaires de Lanczos associées à un calcul de plus petite valeur propre d'une matrice tridiagonale sont la base du calcul de d_k [26, 53, 83].

Comme nous l'avons dit précédemment lors de la description de la méthode de Newton tronqué, l'implémentation et le test de cette recherche dans notre contexte représente une extension possible de notre travail.

3.2.2.7 Recherche avec mémoire

Lorsqu'on travaille avec le hessien de la fonction, ce qui semble pour l'instant inaccessible pour nos problèmes, on prend seulement en compte le caractère

local de la fonction. Ceci peut donner de mauvais résultats quand la fonction est fortement non linéaire. La direction de Newton se révèle assez moyenne car le hessien varie beaucoup. Ainsi, cette méthode propose de considérer, à chaque itération, un modèle local combinaison linéaire du modèle quadratique classique et du modèle de l'itération précédente. L'algorithme plus global est alors moins sensible aux non linéarités [43]. Nous ne l'avons pas testé.

3.2.3 La méthode des régions de confiance

La méthode des régions de confiance est une méthode qui calcule "simultanément" la longueur et la direction de descente. Cette méthode, bien que plus lourde à mettre en œuvre que les méthodes classiques de Newton, peut être une alternative fort puissante dans le cas de problèmes fortement non linéaires. D'une part, cette méthode est "intermédiaire" entre celle de Newton et celle de plus profonde descente. Nous verrons ci-dessous pourquoi. En cela, on pourra faire le rapprochement de cette méthode avec la méthode de Newton tronqué. D'autre part, elle est très performante quand on possède le vrai hessien et que celui-ci n'est pas défini positif. Les directions de courbure négative sont utilisées de manière naturelle. On pourra rapprocher cette spécificité de celle des recherches linéaires qui prennent en compte les directions de courbure négative.

Comme les méthodes classiques, on considère le modèle quadratique ψ_k défini par l'équation (1). Mais, au lieu de travailler sur la direction de Newton qui peut se révéler assez pauvre, on travaille dans l'espace tout entier en prenant comme paramètre la norme maximale de la direction cherchée $\triangle_k$. Ainsi, on définit une région de confiance autour de x_k dans laquelle le modèle quadratique est minimisé.

Ensuite, l'algorithme calcule le déplacement d_k réalisant l'optimum du modèle quadratique dans la région de confiance (ou une fraction de celui-ci). On construit le nouveau point $x_{k+1} = x_k + d_k$, et on calcule le ratio entre la diminution réelle et la diminution prévue par le modèle.

$$pred_k(d_k) = \psi_k(x_k) - \psi_k(x_k + d_k),$$

$$ared_k(d_k) = f(x_k) - f(x_k + d_k),$$

$$\rho_k = \frac{ared_k(d_k)}{pred_k(d_k)}.$$

Si l'accord est bon ($\rho_k \geq 1$), respectivement modéré, respectivement mauvais ($\rho_k \leq 1$), le rayon Δ_k de la région de confiance est augmenté, respectivement inchangé, et on pose $x_{k+1} = x_k + d_k$, respectivement réduit et $x_{k+1} = x_k$. Ceci permet de mettre à jour la cohérence du modèle quadratique avec la fonction f réelle.

Ainsi, il ne reste plus qu'à résoudre le sous-problème classique des régions de confiance $\mathcal{P}_k$ à chaque itération défini par

$$(\mathcal{P}_k) \qquad \min \Big\{ \psi_k(d_k),\ \| D_k.d_k \| \leq \triangle_k \Big\}.$$

La matrice D_k est une matrice de poids qui permet éventuellement de prendre en compte la non isotropie de la fonction f dans $\mathbb{R}^n$.

Lorsqu'un déplacement solution d'un sous-problème $(\mathcal{P}_k)$ est rejeté, le rayon est diminué. En cas de quelques échecs successifs, le rayon de confiance est si faible que le déplacement calculé coïncide avec le déplacement selon la plus profonde descente. C'est pourquoi, on dit que la plus mauvaise direction de descente que peut considérer l'algorithme est la direction de plus profonde descente.

Si une erreur relative assez faible est effectuée lors du calcul du gradient, l'algorithme diminuera tout de même la fonction critère [9]. Dans le cas d'un algorithme de Newton non tronqué, la direction de descente considérée est la direction de Newton. Or, si l'erreur relative commise sur le gradient correspond à un vecteur propre associé à une valeur propre, de l'inverse de la matrice représentant le hessien, beaucoup plus grande que celle associée au vecteur proche le plus "proche" du gradient, la direction de Newton peut ne pas être une direction de descente [10]. Cette différence de comportement est à l'origine de la plus grande robustesse des algorithmes des régions de confiance lors du traitement de fonctions bruitées.

3.2.3.1 Algorithme de Newton

L'application de la méthode des multiplicateurs de Lagrange au problème sous contraintes $(\mathcal{P}_k)$ introduit un cœfficient de lagrange λ_k^* unique à l'optimum vérifiant

$$d \text{ solution de } (\mathcal{P}_k) \Longleftrightarrow \begin{cases} \| D_k.d \| \leq \triangle_k \quad \text{et} \quad \exists\, \lambda_k^* \geq 0, \\ \Big(\bigtriangledown^2 f(x_k) + \lambda_k^* D_k^T D_k \Big).d = - \bigtriangledown f(x_k), \\ \bigtriangledown^2 f(x_k) + \lambda_k^* D_k^T D_k \quad \text{semi-définie positive}, \\ \lambda_k^* \Big(\triangle_k - \| D_k.d \| \Big) = 0. \end{cases}$$

Les premiers algorithmes consistaient à trouver λ_k^* par un algorithme de Newton appliqué à une équation non linéaire [77].

3.2.3.2 Décomposition du hessien

Une autre façon de résoudre exactement le problème $(\mathcal{P}_k)$ consiste à décomposer le hessien par

- une factorisation de Cholesky modifiée [38, 99],
- une factorisation symétrique indéfinie de Bunch-Parlett [6, 55],
- une factorisation de Cholesky incomplète avec mémoire limitée [62, 70].

On définit à l'issue de cette décomposition une matrice de poids. Ensuite, par quelques changements de variables judicieux utilisant les produits de la décomposition, on se ramène à un problème de résolution élémentaire [42].

Le fort coût algébrique de ces factorisations, à part peut-être pour la factorisation symétrique indéfinie de Bunch-Parlett, rendent ce type de méthodes

impraticables en grande dimension. Afin d'être viable en grande dimension, plusieurs travaux se sont proposés de calculer l'optimum en n'utilisant seulement que des produits matrices-vecteurs, en considérant d'autres paramétrisations du problème [54, 96]. Ces algorithmes rencontrent des difficultés lors du traitement de problèmes réalistes, mais des résultats de recherches récentes les améliorent constamment

3.2.3.3 Dogleg algorithms

La résolution exacte du sous-problème $(\mathcal{P}_k)$ est trop coûteuse en grande dimension. Il est souvent plus avantageux de résoudre le sous-problème de manière approximative. La première idée est de restreindre la dimension de l'espace sur lequel le modèle est minimisé: c'est la démarche des algorithmes dits "dogleg". Plus exactement, on approxime par des morceaux de droites la courbe définie par $p(\delta), p$ solution de $\min \left\{ \psi(p),\ \| p \| \leq \delta \right\}, \delta \leq \triangle$ [77].

Ces algorithmes sont mal définis si le hessien est singulier. De plus, on doit résoudre un système linéaire coûteux en grande dimension.

3.2.3.4 Gradient Conjugué tronqué

Le modèle local étant quadratique, on peut lui appliquer un algorithme de Gradient Conjugué [101]. Comme dans la méthode de Newton tronqué, on effectue un Gradient Conjugué qu'on tronque dès qu'une direction de courbure négative est détectée, dès que l'itéré courant sort de la région de confiance ou qu'on a convergé suffisamment. Lorsqu'une direction de courbure négative est détectée, on arrête le Gradient Conjugué et on normalise cette direction afin de se situer sur la frontière de la région de confiance: c'est le point de Steihaug.

Comme dans l'algorithme de Newton tronqué, des itérations supplémentaires de Lanczos permettent de calculer une direction de courbure négative plus riche que la première. Un nombre entre 5 et 10 semble constituer un bon compromis entre l'apport au niveau de la convergence et le travail supplémentaire effectué [45].

3.2.3.5 Régions de confiance non monotone

Ces algorithmes sont issus de la même idée que les algorithmes de recherche linéaire non monotones. D'ailleurs, on observe une grande similarité entre les algorithmes. On peut souligner que les algorithmes de régions de confiance non monotones ont plutôt été élaborés pour les problèmes sous contraintes [18, 105].

3.2.3.6 Combinaison des régions de confiance et du backtracking

Dans l'algorithme des régions de confiance, si un accroissement de f est observé, on diminue le rayon $\triangle_k$ et on réamorce une résolution du sous-problème classique. En grande dimension, cela peut devenir très coûteux. Or, on observe

que réduire le rayon de confiance donne des directions qui sont proches de la direction qui échoua à la première itération. Nocedal & Yuan proposent alors de garder la direction de descente qui échoua en premier lieu et d'effectuer une recherche linéaire "backtracking" [87]. Ainsi, on économise un grand nombre de résolutions du sous-problème $(\mathcal{P}_k)$.

3.2.3.7 Propriétés

On peut déjà remarquer que la méthode des régions de confiance généralise la méthode de Newton avec recherche linéaire, qui est une méthode des régions de confiance à rayon Δ_k infini. On comprend ainsi que la méthode soit plus robuste dans les cas difficiles car le rayon s'ajuste à la fonction critère. De plus, elle définit une catégorie d'algorithmes qui possèdent des propriétés remarquables. Notamment, on possède des résultats de convergence plus forts que ceux obtenus pour une recherche linéaire classique [77]. Par ailleurs, un résultat de Carter montre que cette méthode est peu sensible au bruit [9]. On possède ainsi une alternative plus performante que la méthode de "downhill simplex" de Nelder & Mead pour les fonctions bruitées (quand on peut définir et calculer le gradient de la fonction critère, toutefois). En effet, on dispose de résultats forts de convergence globale même si on commet une erreur relative sur le calcul du gradient allant jusqu'à 50%.

En contre-partie, cette méthode possède quelques défauts qui peuvent la rendre parfois peu performante. En particulier, rien ne nous indique quelle valeur choisir pour le choix du rayon de confiance initial, contrairement à la longueur de descente dans les recherches linéaires. Or, un mauvais choix de ce rayon peut mener à de nombreuses itérations superflues. On peut mentionner ici l'existence d'une heuristique pour ce choix [97], même si elle entraîne un nombre d'évaluations de fonctions supplémentaires. Enfin, soulignons-le une nouvelle fois, cette méthode ne semble vraiment intéressante que lorsque l'on dispose du hessien et que celui-ci n'est pas défini positif (problèmes mal posés, fortement non linéaires). Or, dans beaucoup d'applications, une matrice de Quasi-Newton est construite. Cette matrice est le plus souvent définie positive (BFGS par exemple), ce qui ôte l'intérêt de travailler avec une méthode des régions de confiance. En effet, dans ce cas, l'algorithme se comporte comme un algorithme de Newton (au besoin tronqué) avec une recherche linéaire classique, excepté que la direction de descente est recalculée à chaque fois que le rayon Δ_k (ou par analogie α_k) échoue.

Les quelques défauts présentés par cette méthode vu les cas que nous voulons aborder, ne nous engagent pas à l'utiliser. En effet, le problème du choix du premier rayon entraîne des itérations supplémentaires que l'on ne peut pas se permettre. De plus, on ne possède pas le hessien, ce qui semble compromettre les résultats. Toutefois, concernant le deuxième point, la mise à jour SR1 semble apporter une alternative intéressante alliée à la méthode des régions de confiance.

Néanmoins, comme on l'a mentionné précédemment, la méthode des régions de confiance possède l'avantage d'être peu sensible au bruit. Cette propriété

très importante est peut-être le signe d'algorithmes économiques au sens où on n'a pas besoin d'un gradient exact à chaque itération mais plutôt d'un gradient qui converge petit à petit vers sa valeur exacte. Ceci ouvre la perspective, comme cela est déjà développé pour des problèmes de météorologie, de faire varier la résolution du maillage au cours de l'optimisation [16], diminuant ainsi le coût en temps de calcul et en mémoire requise. D'autre part, cette propriété semble montrer que l'utilisation d'un hessien approché peut donner des résultats satisfaisants, notamment si on considère une résolution du sous-problème caractéristique par un algorithme de Gradient Conjugué tronqué.

3.3 Méthodes mixtes

On a vu précédemment que les algorithmes stochastiques convergent vers le minimum global très lentement et que les algorithmes de gradient assurent une convergence vers un minimum local rapidement. Dans le cas de fonctions différentiables fortement non linéaires, des algorithmes hybrides sont développés afin de combiner les avantages des algorithmes précédents.

La combinaison la plus simple consiste à faire une optimisation locale sur la solution (recuit simulé) ou les solutions (algorithmes génétiques) données par l'algorithme stochastique. Cette démarche permet d'augmenter la précision des algorithmes stochastiques avec peu de travail supplémentaire.

La démarche que suivent la plupart de ces algorithmes est celle qui consiste à remplacer dans un algorithme stochastique la valeur de la fonction critère en un point par la valeur de la fonction critère obtenue après optimisation locale à partir de ce point. Ces algorithmes sont apppelés plus généralement algorithmes mémétiques [92]. La méthode SALO (Simulated Annealing and Local Optimization) utilise le recuit simulé [20], et la méthode GLS (Guided Local Search) utilise la méthode Tabou [108, 107]. Dans le même esprit, on note aussi le développement de deux algorithmes qui améliorent notablement les algorithmes génétiques. L'algorithme génétique parallèle et l'algorithme BGA ("Breeder Genetic Algorithm") sont des extensions des algorithmes mémétiques qui exploitent efficacement le parallélisme des machines [81, 82]. Ces algorithmes semblent donner de très bons résultats en optimisation combinatoire, l'optimisation locale s'effectuant bien sûr de différentes manières.

Pour tous les algorithmes précédents, l'algorithme principal est issu des méthodes stochastiques. A l'opposé, on peut citer deux autres méthodes qui résultent d'une adaptation des méthodes de gradient. La première est la méthode du gradient stochastique. Dans le calcul de la direction de descente, des bruits sont introduits afin d'éviter les minima locaux. Ceci se traduit par un terme supplémentaire dans le passage x_k à x_{k+1},

$$x_{k+1} = x_k - \tau_k \bigtriangledown f(x_k) + \sqrt{\frac{\tau_k}{\nu_k}}\epsilon_k.$$

où ϵ_i sont typiquement des variables identiquement distribuées suivant la loi normale $\mathcal{N}(0,1)$, et les suites $(\tau_k)_{k\in\mathrm{N}}$ et $(\nu_k)_{k\in\mathrm{N}}$ jouent le même rôle que la température guidant l'algorithme de recuit simulé. Il faut

$$\tau_k \longrightarrow 0,$$

$$\sum_k \tau_k \longrightarrow +\infty,$$

$$\nu_k \longrightarrow +\infty.$$

Par exemple, $\tau_k = \frac{1}{k}$. L'intérêt du gradient stochastique est qu'on ne peut pas, comme dans le gradient déterministe, converger numériquement vers un point selle. De plus, on est assuré en théorie de converger vers le minimum global.

Bien qu'il n'intervienne aucune notion probabiliste, on retrouve dans la seconde méthode quelques caractéristiques jusqu'à maintenant propres aux méthodes stochastiques. L'algorithme développé par N. Masmoudi [75] travaille sur une population de points $(x_1, ..., x_p)$ comme les algorithmes génétiques. Le passage entre deux populations s'effectue en définissant une direction $d \in \mathbb{R}^n$ puis en utilisant la différentiation automatique en mode direct pour calculer les dérivées partielles de f d'ordre élevé et ainsi en déduire les premiers cœfficients de Taylor des fonctions $f_i(t) = J(x_i + t.d)_{i\in[1,p]}$. Comme cela est déjà le cas pour le calcul du hessien, ceci n'est pas envisageable dans notre cas. On cherche ensuite les zéros des polynômes approchant les fonctions $f_i(t)$. Comme la population a une taille maximale, une sélection dépendant de la distance entre les points de la population et d'un paramètre α est effectuée. Ce paramètre α, qui décroît durant l'optimisation, joue sensiblement le même rôle que la température intervenant dans le recuit simulé.

Quand le temps de calcul de la fonction critère et du gradient est dérisoire, ces algorithmes améliorent de façon notable les algorithmes stochastiques. Par rapport aux algorithmes de gradient, ils sont beaucoup plus sûrs (convergence vers le minimum global) même si le temps de calcul est encore trop grand. Comme les algorithmes stochastiques, ces algorithmes sont difficilement utilisables lorsque la fonction critère et son gradient sont coûteux, ce qui est le cas pour les problèmes que nous abordons.

3.4 Méthodes déterministes sans utilisation du gradient

D'autres branches de l'optimisation déterministe s'intéressent aux problèmes mettant en jeu une fonction critère non différentiable, une fonction critère bruitée, ou une fonction critère présentant un grand nombre de minima locaux.

Concernant les fonctions non différentiables, les méthodes les plus classiques sont les méthodes de sous-gradient, de plans sécants ou de directions réalisables. On utilise les sous-gradients de f calculés pendant les itérations précédentes pour approcher le sous-différentiel de f [4].

Quand la fonction présente une multitude de minima locaux, une alternative aux méthodes stochastiques ou semi-stochastiques sont les méthodes régularisantes ("appelées smoothing methods") [84]. La fonction est régularisée

de manière à faire disparaître tous les minima locaux. Par exemple, la méthode de diffusion étale la fonction critère originelle en résolvant l'équation de diffusion dont la condition initiale est donnée par la fonction critère [91].

Lorsque la fonction est bruitée, on compte deux approches intéressantes. La première est la méthode de "downhill simplex" de Nelder & Mead [27] modifiée par Torczon pour assurer la convergence globale [106]. On part d'un simplexe dont les sommets sont définis par $N+1$ points (si le nombre de paramètres est N). Suivant les valeurs des fonctions au sommet, on effectue des tranformations sur ce simplexe (réflexion, expansion, contraction) jusqu'au moment où le simplexe est quasiment réduit à un point. Une deuxième approche utilise sur un modèle quadratique, déduit par interpolation de plusieurs points, l'algorithme des régions de confiance [21]. Quelques expériences numériques semblent conclure à la plus grande robustesse de cette méthode par rapport à la méthode de Nelder-Mead. De plus, le nombre d'évaluations de fonction pour atteindre l'optimum est divisé par deux.

Pour nos problèmes, on aurait pu prendre en compte le fait que la fonction risquait de ne pas être différentiable. Seulement ces méthodes sont généralement assez coûteuses en évaluations de fonction. Les méthodes régularisantes et les méthodes pour fonction bruitée présentent les mêmes caractéristiques. Bien que ceci soit discutable, on a donc préféré faire une hypothèse de régularité sur la fonction. Ainsi, on a mis en œuvre des algorithmes qui peuvent diminuer la fonction critère de manière notable en peu d'itérations.

4 Calcul du gradient

Autant que nous ayons pu en juger, l'optimisation dans un vrai contexte industriel[5] reste souvent confinée à des approches très rudimentaires. Les simulations numériques étant lourdes, il est fréquent de procéder en deux temps. D'abord on réalise une approximation du résultat de la simulation numérique par une fonction simplifiée (polynomiale) d'un petit nombre de paramètres. La détermination de cette fonction est par exemple issue d'un plan d'expérience numérique. Dans un second temps, ayant ainsi rendues les évaluations de la fonction "gratuites", un algorithme d'optimisation du type stochastique ou un simple algorithme de Gauss-Newton (avec un gradient approché par différences finies) est utilisé. Une alternative à cette approche en deux temps est de conserver la simulation numérique en tant que telle et de l'inclure dans un algorithme "d'ordre 0" du type simplexe.

Notre démarche part du constat que dans le monde de la recherche plus académique, à la fois des méthodes de calcul du gradient efficaces et des méthodes d'optimisation avancées utilisant ce gradient existent. Nous avons pour objectif de vérifier la portabilité de telles méthodes en milieu industriel "classique", rendant ainsi possible l'utilisation directe de la simulation numérique (et non

5. hormis dans quelques domaines de pointe.

d'une de ses approximations). Des démarches similaires ont déjà lieu dans des domaines industriels très en pointe du point de vue de la simulation numérique.

Dans cette optique, la partie précédente nous permet d'affirmer que seules les méthodes de gradient peuvent répondre à nos objectifs. Il reste à montrer que des méthodes de calcul de gradient adéquates existent, et, à choisir celle qui répond au mieux à nos exigences[6]. Si on met à part le cas du Calcul Symbolique, peu pertinent dans le contexte industriel, deux grands types de méthodes ressortent: les méthodes directes comprenant les différences finies et la différentiation automatique en mode direct, et les méthodes basées sur la résolution de l'adjoint comprenant notamment la différentiation automatique en mode adjoint.

4.1 Différences finies

La seconde consiste à approximer la dérivée selon une direction par différences finies. Elle est surtout utilisée lorsqu'on ne possède pas le code source mais juste l'exécutable jouant le rôle de boîte noire.

Lorsqu'on minimise par rapport à n variables, cette méthode requiert $n+1$ calculs de f, induisant, dans le cas d'une simulation longue, des temps de calcul trop conséquents. De plus, dans le cas de fonctions chahutées, un pas inadéquat peut mener à des dérivées totalement fausses. Il arrive aussi, au niveau numérique, que, par manque de consistance avec le schéma de calcul de f, des dérivées assez chaotiques soient obtenues.

Dans notre cas, les simulations et donc les évaluations de fonctions sont coûteuses. Ainsi, il semble inopportun d'utiliser les différences finies qui entraînent très vite un trop grand nombre de simulations lorsque le nombre de paramètres croît. Par ailleurs, la fonction qu'on veut optimiser n'est pas analytique par rapport à ses variables, le Calcul Symbolique ne nous est donc aussi d'aucun secours.

4.2 Introduction et résolution de l'état adjoint

Au problème original, on associe un problème adjoint qui consiste en la résolution d'un système d'équations aux dérivées partielles (rétrogrades si le système est un système d'évolution) dépendant de la solution des équations d'état. Jameson fut l'un des premiers à utiliser la théorie du Contrôle Optimal en Optimisation de forme [60, 61]. Cependant, il semble que le nombre d'itérations nécessaire à l'atteinte du minimum croisse plus que linéairement avec le nombre de paramètres de contrôle. Afin de diminuer le nombre d'itérations, des variantes appelées One-Shot Method [1, 2, 103] et Pseudo-Time Methods [59, 102] ont été développées.

4.2.1 Adjoint du problème

Au problème de contrôle standard

6. en fait, les deux aspects sont évidemment liés car la méthode d'optimisation choisie dépend de considérations faites sur la fonction et son gradient.

$$(\mathcal{P}) = \left\{ \begin{array}{l} min \mathcal{J}(u, \mathcal{X}), \\ \mathcal{R}(u, \mathcal{X}) = 0. \end{array} \right.$$

on associe le lagrangien de $(\mathcal{P})$ défini par

$$\mathcal{L}(u, \mathcal{X}, p) = \mathcal{J}(u, \mathcal{X}) + \Big(p, \mathcal{R}(u, \mathcal{X})\Big).$$

La stationnarité du lagrangien par rapport au triplet $(u, \mathcal{X}, p)$ induit une équation aux dérivées partielles sur l'adjoint p appelée équation adjointe de $(\mathcal{P})$

$$\Big(\frac{\partial \mathcal{R}}{\partial \mathcal{X}}\Big)^T . p = -\frac{\partial \mathcal{J}}{\partial \mathcal{X}}.$$

L'adjoint vérifiant cette équation vérifie la propriété

$$\frac{d\mathcal{J}}{du}(u, \mathcal{X}) = \frac{\partial \mathcal{J}}{\partial u}(u, \mathcal{X}) + \Big(p, \frac{\partial \mathcal{R}}{\partial u}\Big)(u, \mathcal{X}).$$

Ainsi, la résolution de l'équation adjointe et de l'équation d'état permet de calculer le gradient total de la fonctionnelle $\mathcal{J}$ par rapport aux variables de contrôle u. L'équation $\frac{d\mathcal{J}}{du}(u, \mathcal{X}) = 0$ est appelée "équation de design".

4.2.2 La méthode One-Shot

Le coût de la résolution de la totalité des systèmes linéaires induits par les équations adjointe et d'état est considérable dès que le maillage atteint une taille moyenne. La méthode One-Shot se propose de résoudre ces systèmes simultanément de manière approximative (quelques itérations de Jacobi). On peut définir un gradient approché qui est inséré dans une méthode de plus profonde descente. La longueur de descente dépend de la finesse du maillage et est assez difficile à calculer automatiquement.

Couplée avec des méthodes multi-grilles, on obtient des algorithmes très efficaces et rapides. L'influence de la taille du maillage semble limitée et le nombre d'itérations nécessaire pour atteindre un minimum local semble quasiment indépendant du nombre de paramètres de contrôle [2].

4.2.3 Pseudo-Time method

La méthode One-Shot demande l'utilisation d'algorithmes multi-grilles. Or, les solveurs industriels ne sont pour le moment que très rarement couplés à de tels algorithmes, ce qui rend la méthode peu utilisable dans la pratique.

Une autre démarche consiste à résoudre l'équation de design exactement à chaque itération en résolvant de mieux en mieux les équations adjointe et d'état. Pour cela, on cherche l'état stationnaire du système suivant

$$\begin{cases} \dfrac{\partial \mathcal{X}}{\partial t} + \mathcal{R}(u, \mathcal{X}) = 0, \\ \dfrac{\partial p}{\partial t} + \mathcal{A}(u, \mathcal{X}, p) = 0, \\ \dfrac{d\mathcal{J}}{du}(u, \mathcal{X}) = 0. \end{cases}$$

Ces méthodes diffèrent des méthodes précédentes basées sur le gradient. Il semble que ces méthodes convergent indépendemment du nombre de paramètres [102].

Les méthodes basées sur la résolution de l'adjoint exigent d'écrire l'adjoint du problème, c'est-à-dire l'adjoint des équations, ce qui n'est pas toujours trivial lorsque plusieurs types de conditions limites sont considérés. De plus, le comportement numérique de ces méthodes semble encore mal maîtrisé et fait toujours l'objet de nombreux travaux. Le problème posé par les conditions limites et l'existence de l'adjoint n'est pas toujours simple à résoudre.

C'est pourquoi, pour des raisons de simplicité mathématique et de mise en œuvre, notre choix s'est porté sur la différentiation automatique utilisée en mode inverse de la partie thermique du code PAMFLOW^{TM} à l'aide du logiciel ODYSSEE [23, 24]. En plus de sa relative simplicité, la différentiation automatique permet très aisément de connaître la contribution de chaque terme intervenant dans la dérivée. Nous allons la détailler maintenant.

4.3 La différentiation automatique

La différentiation automatique est le procédé qui, à partir d'un code informatique évaluant une fonction f, produit un code qui évalue les valeurs exactes (aux erreurs d'arrondi près) des dérivées partielles de f [37, 46]. En représentant chaque ligne du code par une fonction élémentaire, le code n'est ni plus ni moins une composition de n fonctions élémentaires (n étant le nombre de lignes du code). La différentiation automatique consiste à appliquer les formules de dérivation des fonctions composées à chacune de ces fonctions élémentaires soit à chaque ligne du code. On peut appliquer les formules de composition de deux façons différentes qu'on appelle mode de différentiation automatique :

- le mode linéaire direct ou linéaire tangent (différentiation directe),
- le mode linéaire inverse ou cotangent (différentiation adjointe).

4.3.1 Mode direct

La différentiation directe est le mode le plus simple et le plus intuitif. Le code est différentié classiquement ligne par ligne. A chaque variable du code v_i, il est associé une variable dérivée $\dot{v}_i$ qui contient la dérivée directionnelle de v_i dans une direction d donnée de $\mathbb{R}^n$. C'est le bon mode pour calculer la dérivée d'un grand nombre de variables de sortie par rapport à un petit nombre de variables d'entrée.

Les dérivées obtenues ont l'avantage d'être plus précises et moins chaotiques que celles obtenues par différences finies. Si n représente le nombre de paramètres de contrôle du problème, le coût de calcul du gradient par la méthode des différences finies est de n calculs de f; dans la méthode de différentiation directe, ce même calcul requiert $4n$ évaluations [80]. La différentiation directe est donc trop lourde lorsque le nombre de paramètres dépasse quelques dizaines. Notamment, pour notre cas, le coût de la fonction et le nombre de paramètres rendent illusoire l'application de ce mode.

4.3.2 Mode inverse

Le mode inverse de différentiation est moins intuitif et plus complexe. Ce mode s'effectue en dualisant ligne par ligne le code originel, mais dans l'ordre inverse à celui de l'exécution du code initial. Il s'agit du "bon" mode pour calculer le gradient d'une application de $\mathbb{R}^n$ dans $\mathbb{R}$, ou le gradient de $d^T f$ lorsque f va de $\mathbb{R}^n$ dans $\mathbb{R}^m$. A chaque variable v_i il est associé une variable duale $\bar{v_i}$ qui contient la variation de $d^T.f$, d donnée de $\mathbb{R}^m$, par rapport à une perturbation de v_i. On comprend ainsi que si v_i est une variable égale au paramètre d'entrée mais qui n'intervient pas dans le calcul de la variable de sortie, alors

$$\dot{v_i} = 1 \quad \text{et} \quad \bar{v_i} = 0$$

Ce mode possède les mêmes avantages (rapidité, précision), en plus d'une relative simplicité de mise en œuvre, que les méthodes standards basées sur la résolution de l'adjoint. Avec ce mode, le coût d'évaluation théorique de l'adjoint est de 5 fois le coût du calcul de la fonction [80], et ce *indépendamment* du nombre de paramètres par rapport auxquels on dérive.

Cependant, comme lors de la résolution de l'adjoint, on peut rencontrer des problèmes de taille mémoire.

En effet, le code tangent produit par ODYSSEE contient la trajectoire du code direct et ses expressions linéarisées. Ainsi, aucune sauvegarde n'est nécessaire. On utilise les données "en temps réel". En revanche, le code adjoint, généré par dérivation dans le sens inverse du code original, nécessite la sauvegarde et le stockage de *toute* la trajectoire ou du moins des parties de la trajectoire intervenant non linéairement dans les calculs. Par trajectoire, on entend non seulement la valeur des variables intervenant dans le calcul à tous les pas de temps, mais aussi les valeurs de passage des algorithmes itératifs internes éventuels à chaque pas de temps. Ainsi, pour des codes complexes de grande taille tels que les codes industriels, on se trouve assez vite confronté à des problèmes de taille mémoire [22].

4.3.3 Post-traitements du code adjoint construit par ODYSSEE

Même si ODYSSEE réalise une partie absolument considérable du travail en écrivant toutes les lignes de codes nécessaires au calcul de l'adjoint, un travail

non négligeable de compréhension et de traitement reste à effectuer sur le code issu de la différentiation.

D'abord, il faut procéder à l'initialisation des variables duales dans la routine de tête et éliminer celles correspondant à des variables supposées constantes au cours de la simulation. Ici, il s'agit des données géométriques et physiques que nous prenons comme fixées et par rapport auxquelles nous n'optimisons pas.

Ensuite, il est utile de programmer des formules plus simples et économiques pour la différentiation des résolutions matricielles implicites (Gradient Conjugué Préconditionné) [22]. Ainsi, on peut s'affranchir de la sauvegarde des valeurs de passage de ces algorithmes itératifs à chaque itération.

Enfin, la partie du post-traitement du code construit par ODYSSEE la plus longue mais la plus importante est l'élimination des sauvegardes superflues opérées par ODYSSEE qui rend parfois le code brut inutilisable; c'est le cas dans notre contexte. En utilisant l'invariance de certaines variables au cours de la simulation, la linéarité des calculs et les propriétés de la plupart des boucles, on a réduit considérablement ces sauvegardes.

4.3.4 Le cas stationnaire

Des études sur le comportement de la dérivée d'un état stationnaire par rapport à des paramètres ont montré sous une hypothèse de contractance que la dérivée peut être calculée à l'aide du seul état stationnaire [12, 15, 35].

Plus précisément, soient J le critère, P les paramètres, $(E_i)_{i\in[0,n]}$ les n états par lesquels passe le système durant son évolution. Le passage de l'état i à l'état $i+1$ est défini par $E_{i+1} = F(E_i)$. Soit J_{E_i} la transposée de la jacobienne de F évaluée en E_i. La dérivée du critère par rapport aux paramètres est donnée par

$$\Big(\frac{dJ}{dP}\Big)^T(P) = \Big(\frac{\partial J}{\partial E_n}\Big)^T(E_n).J_{E_{n-1}}.J_{E_{n-2}}....J_{E_0}.\Big(\frac{\partial E_0}{\partial P}\Big)^T(P).$$

Si l'état stationnaire est approché suffisamment précisément et si la matrice J_{E_n} est contractante ($\| J_{E_n} \| < 1$), une approximation raisonnable de la dérivée est fournie par

$$\Big(\frac{dJ}{dP}\Big)^T \simeq \Big(\frac{\partial J}{\partial E_n}\Big)^t.(J_{E_n})^n.\Big(\frac{\partial E_0}{\partial P}\Big)^T.$$

Cette propriété est très intéressante car elle permet de réduire la sauvegarde de tous les états intermédiaires à la sauvegarde d'un seul état. La taille mémoire requise est ainsi constituée principalement des variables duales.

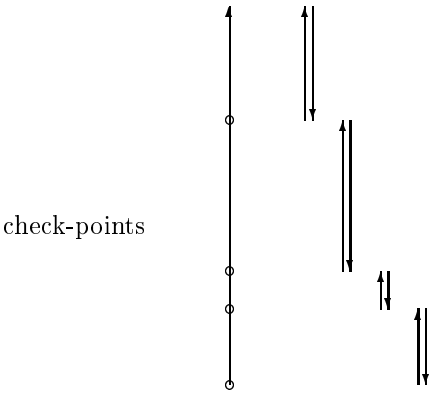
Ainsi, sur le problème 1 que nous avons traité, en sauvegardant une grande quantité des valeurs de la trajectoire intervenant avec la même valeur à chaque itération adjointe, nous sommes passés pour le temps de calcul du gradient par rapport au calcul direct de 3.6 (cas général) à 2.3 (cas stationnaire).

Evidemment, cette propriété est spécifique au cas stationnaire.

4.3.5 Le cas transitoire

En transitoire, l'état final dépend des états intermédiaires par lesquels passe le système, il est donc nécessaire de sauvegarder la trajectoire à chaque pas de temps. Or, la petitesse du pas de temps entraîne un nombre d'itérations important si on veut simuler un système pendant une durée physique pertinente (quelques dizaines de minutes). L'algorithme de sauvegarde de l'état à chaque pas de temps paraît donc inadéquat pour les problèmes d'évolution.

Pour palier le manque de mémoire vive disponible on a la possibilité d'effectuer des sauvegardes sur fichier. Mais, dans le cas de codes complexes, la sauvegarde sur fichier n'est pas à elle seule suffisante car il arrive fréquemment que la place mémoire (disque et vive) soit insuffisante pour conserver toute la trajectoire. L'adjoint ne peut être calculé directement. Basé sur l'idée classique "diviser pour régner", A. Griewank a proposé le concept de calcul des adjoints par morceaux [46]. De manière simplifiée, la méthode consiste à sauver l'état du modèle de temps en temps au cours d'une exécution du modèle direct. Ensuite, ces instants particuliers appelés "check-points", permettent de reprendre l'exécution du modèle direct qui construit un morceau de trajectoire utilisable pour évaluer l'adjoint. Le calcul de l'adjoint complet est ainsi réalisé sur un espace mémoire limité. Toute l'astuce consiste donc choisir intelligemment la répartition des check-points afin d'optimiser le temps de calcul et la mémoire utilisée.



1 fois l'évolution complète avec stockage des check-points	1 + 1 fois l'évolution "morcelée"

TAB. 1 – *Schéma général d'un calcul d'adjoint par morceaux*

On dénombre quatre principaux schémas de check-points. Le premier appelé

Treeverse [49], établi par Griewank, utilise une distribution binomiale pour la répartition de ces points afin d'atteindre une complexité spatiale logarithmique [47]. Le second consiste à placer ces instants périodiquement. Le schéma de période un est celui adopté par ODYSSEE. C'est celui que nous utiliserons dans toute la suite de notre étude, sachant que nous avons vérifié sur des cas tests qu'il est moins bon que le schéma Treeverse (ce que la théorie prévoit), mais que l'utilisation du schéma Treeverse n'affecterait pas significativement les conclusions que nous allons tirer sur la comparaison des divers algorithmes d'optimisation. Le troisième, basé sur l'idée qu'à la fin du calcul adjoint on a plus de mémoire disponible qu'au début du calcul adjoint, place les points de sauvegarde selon une régression arithmétique. Enfin, le quatrième utilise une méthode de bissection qui divise un problème de taille P en $log_2(P)$ sous problèmes de tailles égales (ou presque). Les complexités temporelle et spatiale [11] des différents schémas sont données dans le tableau 2, P étant le nombre d'itérations temporelles de la simulation.

Alg.	Schéma	Complexité temporelle	Complexité spatiale
1	Treeverse	$O(Plog(P))$	$O(log(P))$
2	sauv. per.	$O(P)$	$O(P)$
3	rég. arithm.	$O(P)$ (si possible)	$O(\sqrt{P})$
4	bissection	$O(Plog(P))$	$O(log(P))$

TAB. 2 – *complexités temporelle et spatiale des différents schémas*

A mémoire identique, il semble, d'après les études disponibles [11], que le schéma de Griewank soit meilleur que le schéma de bissection. On a vérifié ce comportement asymptotique dans notre cas (problème 2) en comparant ces deux schémas et un schéma hybride mêlant les deux approches.

Pour donner une idée de la taille mémoire et du temps de calcul, signalons que pour le problème P_2, avec un nombre de nœuds égal à 5.10^5 et un nombre d'itérations égal à 10^4, le calcul du gradient d'un calcul couplé (température + vitesse) exige environ 1 Giga de mémoire pour un calcul de durée six fois supérieure à celui de la simulation directe.

5 Résultats numériques

Cette partie est consacrée aux résultats numériques obtenus sur les deux problèmes $\mathcal{P}_1$ et $\mathcal{P}_2$. Afin de donner une idée du temps de calcul requis par chaque optimisation, le temps d'une simulation est pris pour unité de temps. On a estimé pour le problème $\mathcal{P}_1$, respectivement $\mathcal{P}_2$, que le calcul du gradient était équivalent à 2.5, respectivement 4, simulations . En effet, dans le cas stationnaire, le calcul du gradient équivaut effectivement à 2.5 évaluations de la fonction. Dans le cas transitoire, cela dépend de la mémoire disponible, et oscille entre 3.5 et 7.

5.1 Problème $\mathcal{P}_1$

Pour ce problème, on a d'abord testé plusieurs algorithmes d'optimisation avec un gradient calculé avec le code adjoint construit par différentiation automatique. Ensuite, on a comparé plusieurs stratégies de calcul du gradient couplées aux algorithmes d'optimisation qui ont donné les meilleurs résultats.

La recherche linéaire considérée dans tous les cas vérifie les conditions de Goldstein & Price. Le point de départ, identique pour toutes les optimisations effectuées, est défini par

$$(T_0, V_0, C_1, T_1) = (25, 10, 1, 16).$$

Chaque calcul de la fonction critère a nécessité 250 itérations afin d'assurer la convergence vers l'état stationnaire. Chaque comparaison est décrite par deux tableaux. Le premier tableau indique les résultats de l'optimisation effectuée: le nombre d'itérations, la valeur du critère final, la valeur de la norme du gradient au point final, le nombre total d'évaluations de la fonction (le temps de calcul du gradient étant défini ci-dessus) et le nombre de fois que le pas de Newton est accepté. Le second tableau indique la valeur minimale de la fonction critère calculée après 20 et 50 évaluations de fonction.

5.1.1 Plusieurs algorithmes de Gradient Conjugué non Linéaire : Tableaux 5, 6

On a comparé quatre stratégies pour le choix du paramètre β_k explicité en partie 3 ("Steepest Descent" (SD), Fletcher & Reeves (FR), Polak & Ribière (PR) et Polak & Ribière stabilisé). Dans tous ces tableaux et ceux du même type qui suivront, le nombre entre parenthèses indique le nombre d'itérations effectuées. En premier lieu, on vérifie l'observation attendue que les algorithmes de Gradient Conjugué non Linéaire sont beaucoup plus efficaces que la méthode de plus profonde descente. L'algorithme le plus performant est celui qui utilise la formule de Fletcher & Reeves. On remarque aussi que la stabilisation de l'algorithme de Polak & Ribière n'influe guère sur le résultat.

En second lieu, ces résultats font ressortir la lenteur de la convergence de ces algorithmes près de l'optimum.

5.1.2 Plusieurs stratégies de "scaling" : Tableaux 7, 8, 9

On se propose dans cette partie de comparer les différentes stratégies de "scaling" répertoriées par Luksàn [65]. Dans nos applications numériques, les stratégies sont plus précisément

- stratégie 1 : $\rho = 1, \gamma = 1$,
- stratégie 2 : $\rho = \frac{1}{2} \frac{s_k^T y_k}{f(x_k) - f(x_{k+1}) + s_k^T \nabla f(x_{k+1})}$ (Spedicato), $\gamma = 1$,
- stratégie 3 : ρ donné par Spedicato, $\gamma = \frac{s_k^T H_k s_k}{y_k^T s_k} \rho$ (Luksàn),
- stratégie 4 : ρ donné par Spedicato, γ de Nocedal & Yuan,

– stratégie 5 : ρ donné par Spedicato, γ donné par Selective Scaling,
– stratégie 6 : ρ donné par Spedicato, γ donné par Interval Scaling,
– stratégie 7 : ρ donné par Spedicato, γ_1 donné par Luksàn puis $\gamma_k = 1$.

Toutes ces stratégies ont été couplées à une mise à jour BFGS de la matrice de Quasi-Newton.

On note tout d'abord que l'algorithme de BFGS standard (pourtant d'ordre 2) se comporte moins bien que les algorithmes de Gradient Conjugué non Linéaire. Pour un nombre d'évaluations de fonction presque égal, on perd un facteur 10^5 sur la valeur minimale de la fonction critère calculée par rapport à l'algorithme de Fletcher & Reeves.

Ensuite, on remarque que l'introduction du paramètre de Biggs, quelle que soit la stratégie de scaling adoptée, améliore remarquablement l'algorithme de BFGS. En effet, au bout de 50 évaluations de fonction, l'algorithme gagne un facteur 10^6 par rapport à l'algorithme de BFGS standard (et 10^5 par rapport à l'algorithme de Gradient Conjugué non Linéaire de Fletcher & Reeves). On remarque aussi qu'à cet instant du processus d'optimisation, deux à trois itérations (ou calcul de gradient) supplémentaires sont effectuées par rapport à l'algorithme FR. Ceci montre que, pour ce problème, la direction de Newton issue d'une mise à jour de BFGS et les directions de Gradient Conjugué sont mal "scalées", entraînant peu de succès de la longueur unité dans les recherches linéaires successives.

Enfin, on note que le scaling de Nocedal & Yuan et le scaling préliminaire de Luksàn sont les plus efficaces au niveau du scaling de la matrice de Quasi-Newton. L'algorithme le plus performant est de loin celui qui utilise le scaling de Luksàn (d'un facteur 100).

5.1.3 Plusieurs mises à jour : Tableaux 10, 11, 12, 13, 14

On se propose dans cette partie de comparer différentes mises à jour de la matrice de Quasi-Newton approchant l'inverse du hessien (sauf pour l'une d'entre elles, numérotée 6 ci-dessous), sans recours à une stratégie de stabilisation et de scaling, soit $\rho = \gamma = 1$. La dénomination des algorithmes est la suivante.

– Algorithme 1: $\eta = 1$ (BFGS),
– Algorithme 2: $\eta = 0$ (DFP),
– Algorithme 3: $\eta = \left(1 - \frac{y_k^T B_k y_k}{s_k^T y_k}\right)^{-1}$ (SR1),
– Algorithme 4: $\eta = \min\left(-1, \frac{1}{2}\left(1 - \frac{y_k^T B_k y_k}{s_k^T y_k}\frac{s_k^T B_k^{-1} s_k}{s_k^T y_k}\right)^{-1}\right)$ (Fletcher),
– Algorithme 5: η donné par Wolkowitz & Zhao [109],
– Algorithme 6: formule de mise à jour du hessien due à Liao [69],
– Algorithme 7: $\eta = 2$ proposé par Luksàn [65],
– Algorithme 8: η donné par Luksàn [65],
– Algorithme 9: η optimal pour Dennis & Wolkowicz [19],

- Algorithme 10: η optimal par rapport à la dimension (Dennis & Wolkowicz),
- Algorithme 11: β optimal par rapport à la dimension (Dennis & Wolkowicz).

Pour l'algorithme 6, il y a deux spécificités. D'une part, c'est le seul où l'on utilise la formule d'approximation du Hessien lui-même et donc il est nécessaire d'effectuer une résolution de système linéaire pour trouver la direction de descente (on utilise une méthode LU). D'autre part, c'est le seul pour lequel il n'y a pas de stratégie de stabilisation/scaling typiquement adaptée. Quand nécessaire (ce qui n'est pas le cas dans ce paragraphe), on en a développée une "empirique".

On peut voir que toutes les mises à jour testées améliorent l'algorithme de Quasi-Newton standard (avec BFGS). En effet, on observe d'une part une économie concernant les évaluations de fonction effectuées pendant les recherches linéaires. D'autre part, on constate une diminution du critère beaucoup plus importante après 50 évaluations de la fonction critère.

Si on se limite à 50 évaluations de la fonction critère, les mises à jour les plus efficaces sont les mises à jour 10 et 11 qui donnent des résultats presque identiques. On peut l'expliquer par les deux itérations supplémentaires réalisées par les algorithmes adoptant ces mises à jour, conséquence d'une meilleure économie d'évaluations de fonction dans la recherche linéaire. Il est aussi curieux de noter qu'en revanche, au bout de 20 évaluations de la fonction, ces mêmes mises à jour se révèlent être les plus mauvaises.

5.1.4 Plusieurs stratégies de calcul du gradient : Tableaux 15, 16, 17, 18

Dans cette section, on a comparé cinq stratégies de calcul du gradient en les couplant chacune à quatre des algorithmes d'optimisation testés précédemment. Les correspondances entre les tableaux de résultats et les algorithmes d'optimisation testés sont

- Tab. 15 : l'algorithme de plus profonde descente,
- Tab. 16 : l'algorithme de Polak & Ribière stabilisé,
- Tab. 17 : l'algorithme de BFGS avec paramètre de stabilisation de Spedicato et scaling de Luksàn
- Tab. 18 : un des deux algorithmes dus à Dennis et Wolkowicz, sans paramètres de stabilisation et de scaling.

Les cinq stratégies de calcul du gradient sont les suivantes:

- gradient donné par l'adjoint du code (GA),
- formule simplifiée du gradient automatique réservée aux problèmes stationnaires (GAS),
- gradient approché par différences finies avec un pas pris égal à 10% (DF1),
- gradient approché par différences finies avec un pas pris égal à 0.1% (DF2),
- gradient approché par différences finies avec un pas pris égal à 0.001% (DF3).

On note tout d'abord que la formule simplifiée du calcul du gradient par différentiation automatique donne exactement les mêmes résultats que le calcul avec sauvegarde de tous les états intermédiaires. Ceci montre que la formule simplifiée est très précise lorsque l'état final est très proche de l'état stationnaire du système.

Dans un second temps, on remarque le peu d'itérations effectuées lors des différents tests avec un gradient approché par différences finies. En effet, une direction qui ne vérifie pas la propriété de descente est rapidement créée. Ceci montre que pour les problèmes étudiés, l'approche des différences finies génère trop d'erreurs dans le calcul du gradient. Ces erreurs sont d'ailleurs beaucoup plus néfastes pour les deux algorithmes de Quasi-Newton. En effet, le calcul de la direction de descente résulte du produit de la matrice de Quasi-Newton (approchée avec le gradient) avec le gradient lui-même.

La qualité de l'algorithme s'accroît avec la petitesse du pas pris pour approcher les dérivées partielles de la fonction critère. Ceci nous indique une certaine régularité de la fonction (pas de bruit) mais aussi une certaine non linéarité. En outre, il est intéressant de voir qu'un pas égal à 0.001% donne des résultats beaucoup moins bons qu'un gradient calculé par différentiation automatique.

Ajouté au plus fort coût en évaluations de fonction, l'algorithme utilisant une approximation du gradient par différences finies se révèle beaucoup moins précis que celui qui utilise un gradient calculé par différentiation automatique suivant ou non la formule simplifiée.

5.2 Problème $\mathcal{P}_2$

Les valeurs des paramètres fixes intervenant dans ce problème sont

- $\left(T_0^{opt}, V_0^{opt}, C_1^{opt}, T_1^{opt}\right) = (25, 10, 3, 16)$,
- $\left(T_0(0), V_0(0), C_1(0), T_1(0)\right) = (17, 5, 1, 12)$.

On a comparé divers algorithmes d'optimisation avec α égal à 10^{-4} puis noté les différences de comportement de ces algorithmes en prenant α égal à 10^{-2}. L'évolution initiale choisie pour tous les tests est définie par

$$T_0^{init}\left(t_i\right) = T_0(0) + 0.01 * \left(T_0^{opt} - T_0(0)\right) * i, \quad i \in [1, 99].$$

On fait de même pour $\left(V_0^{init}\left(t_i\right), C_1^{init}\left(t_i\right), V_0^{init}\left(t_i\right)\right)$. Les valeurs de la fonction et de la norme du gradient en cette évolution sont respectivement 806.4 et 5.763.

5.2.1 Plusieurs algorithmes de Gradient Conjugué : Tableaux 19, 20

On a comparé les quatre stratégies pour le choix du paramètre β_k testées précédemment. A chaque optimisation, une recherche linéaire vérifiant les conditions de Goldstein & Price a été adoptée.

Le tableau Tab. 19 indique la valeur du critère obtenue au bout de 19 itérations, le nombre total d'évaluations de fonction requis et la norme du gradient à l'itéré final. Le tableau Tab. 20 indique la valeur du critère obtenu au bout d'environ 25, 50 et 100 évaluations de fonction. Le chiffre entre parenthèses indique le nombre d'itérés calculés (ou nombre d'évaluations de gradient).

Contrairement au problème précédent, l'algorithme de Polak & Ribière est bien meilleur que les deux autres algorithmes de Gradient Conjugué. La supériorité de cet algorithme est souvent observée en pratique, ce qui semble n'avoir jamais été vraiment compris d'un point de vue théorique. De même, la stabilisation de cet algorithme n'influe guère sur le résultat. En fait sur les 19 itérations, une seule valeur de β_k négative a été formée.

Par ailleurs, on note que la formule de Fletcher & Reeves est très peu compétitive. Elle est même moins efficace que la méthode de plus profonde descente. La formule de Fletcher & Reeves ne permet pas de diminuer la norme du gradient, contrairement à la formule de Polak & Ribière et à la méthode de plus profonde descente.

Ceci est souvent expliqué par l'adoption de petites longueurs de descente qui rendent la convergence lente. En effet, si une faible longueur de descente est générée au cours de l'algorithme de Fletcher & Reeves, les longueurs de descente suivantes risquent d'être faibles aussi [85]. Ceci peut s'expliquer naïvement de la manière suivante

$$
\begin{aligned}
\alpha_k \ll 1 \quad &\Rightarrow \quad x_{k+1} \simeq x_k \quad \text{et} \quad d_k \text{ mauvaise ,} \\
&\Rightarrow \quad \beta_{k+1} \simeq 1 \quad \text{par continuité du gradient,} \\
&\Rightarrow \quad d_{k+1} \simeq d_k \quad \text{si} \quad \| d_k \|_2 \gg \| g_{k+1} \|_2, \\
&\Rightarrow \quad \alpha_{k+1} \simeq \alpha_k \ll 1.
\end{aligned}
$$

Avec la formule de Polak & Ribière, on ne rencontre pas cette difficulté. Si une direction de descente mauvaise est générée alors la continuité du gradient induit $\beta_k \simeq 0$ et $d_{k+1} \simeq -g_{k+1}$. L'algorithme effectue ainsi un restart automatique à partir de la direction de plus profonde descente. Pour notre problème, les longueurs de descente calculées par les deux algorithmes sont du même ordre de grandeur. En revanche, avec la mise à jour de Fletcher & Reeves, les normes des directions de descente calculées sont entre cinq et dix fois plus grandes que celles des gradients intermédiaires. De plus, le coefficient β_k oscille entre 0.6 et 1.5. L'algorithme n'a pas pu rattraper le calcul d'une direction moyenne au cours des itérations suivantes. Il semble donc que l'efficacité relative des algorithmes de Gradient Conjugué non Linéaire dépende de la dimension du problème.

Par ailleurs, on vérifie que ces méthodes sont assez lentes dès qu'on se rapproche de l'optimum. Pour ces algorithmes, le travail étant quasiment nul comparé à la détermination et l'inversion du hessien dans un algorithme de Newton, une idée particulièrement bénéfique consiste à effectuer une recherche linéaire plus précise en diminuant le réel β_2 de la recherche de Wolfe [27]. Comme le signe du produit scalaire des gradients intermédiaires, calculés pendant la recherche linéaire, avec la direction de descente est important pour passer de α_k^i à α_k^{i+1},

seule une recherche de Wolfe est envisageable. On a donc regardé le comportement de ces algorithmes pour trois valeurs de β_2, à savoir $\beta_2 = 0.2$, $\beta_2 = 0.5$ et $\beta_2 = 0.9$. Les tableaux suivants sont similaires aux précédents, sachant que le nombre d'évaluations mentionné représente le nombre d'évaluations de gradient car chaque itération de la recherche linéaire requiert le calcul du gradient. La dernière ligne de chaque tableau donne la valeur minimale du critère calculée si on ne se donne droit qu'à 40 évaluations de gradient.

Les trois tableaux 21 montrent effectivement qu'une recherche linéaire plus précise améliore les résultats. Par ailleurs, il semble que les résultats n'évoluent plus quand β_2 est inférieur à 0.5. Ils sont d'ailleurs moins bons pour $\beta = 0.2$ car l'amélioration du résultat ne compense plus l'effort numérique supplémentaire.

Au vu des résultats obtenus avec une recherche de Goldstein & Price, une recherche de Wolfe paraît trop coûteuse à mettre en œuvre pour nos problèmes.

5.2.2 Plusieurs stratégies de "scaling" : Tableaux 22, 23, 24

On se propose dans cette partie de comparer les stratégies de "scaling" introduites lors de la présentation des résultats obtenus sur le problème $\mathcal{P}_1$. Les différentes stratégies sont ordonnées comme précédemment. On les a couplées à la mise à jour de BFGS et à une recherche linéaire vérifiant les conditions de Goldstein & Price.

Contrairement à ce que nous observions pour le problème $\mathcal{P}_1$, l'algorithme de BFGS standard est meilleur que tous les algorithmes de Gradient Conjugué non Linéaire (facteur 2 par rapport à Polak & Ribière).

De même, la seule prise en compte du paramètre de stabilisation ρ n'améliore plus mais dégrade les résultats. Cependant on note que la longueur unité est acceptée par la recherche linéaire un plus grand nombre de fois.

Ensuite, exceptées les stratégies 5 et 6, les autres stratégies de scaling n'apportent pas d'améliorations notables. La stratégie 6 donne des résultats très mauvais, alors que la stratégie 5 se révèle réellement efficace. De plus, cette stratégie est la seule qui accepte la longueur unité à toutes les itérations où celle-ci est testée en premier.

Dans ce cas aussi, un choix pertinent des paramètres de stabilisation et de scaling permet d'améliorer l'algorithme de Quasi-Newton standard.

5.2.3 Plusieurs mises à jour

On se propose dans cette partie de comparer les mises à jour de la matrice de Quasi-Newton, approchant l'inverse du hessien, introduites précédemment. Ces mises à jour sont couplées à une recherche linéaire de Goldstein & Price. Les premiers tableaux présentent les résultats obtenus sans recours à une stratégie de stabilisation et de scaling. Dans un second temps, on a branché sur ces algorithmes la stratégie de scaling sélectif allié à un paramètre de stabilisation donné par Spedicato.

5.2.3.1 Sans Scaling : Tableaux 25,26,27,28

Ces résultats montrent un comportement différent de celui observé pour le problème $\mathcal{P}_1$. D'abord, l'algorithme de BFGS est le meilleur, sauf à considérer l'algorithme 8. Les algorithmes 2 (DFP) et 5 (qui utilise la mise à jour de DFP) sont les plus mauvais. On note aussi une différence notable d'efficacité entre les algorithmes 10 et 11 qui donnaient des résultats similaires précédemment.

On peut remarquer aussi l'efficacité de la plupart des mises à jour testées autres que BFGS, dont notamment la mise à jour de DFP, dans l'acceptation de la longueur unité.

5.2.3.2 Avec Scaling : Tableaux 29,30,31,32

On peut noter que le scaling adopté dégrade le scaling de la direction de Newton pour la majorité des algorithmes : la longueur unité est moins souvent acceptée. Ainsi, l'efficacité de la stratégie de scaling en matière d'économie d'évaluations de fonction dans la recherche linéaire dépend de la mise à jour de la matrice de Quasi-Newton adoptée.

On peut faire la même conclusion concernant l'efficacité de la stratégie de scaling à nombre d'évaluations de fonction donné. Même si tous les algorithmes diminuent mieux la fonction que dans le cas où aucune stratégie de scaling n'est considérée, on note des comportements spécifiques à chaque mise à jour. En effet, les algorithmes 3 (SR1) et 8 ne sont que peu améliorés. Les algorithmes 2 et 5, peu recommandés sans stratégie de scaling, se révèlent à l'exception de l'algorithme 7 les plus efficaces. 50 évaluations de fonction suffisent pour obtenir une diminution de la fonction critère meilleure qu'après 100 évaluations de fonction sans stratégie de scaling. Les algorithmes 10 et 11 retrouvent des résultats similaires. Enfin, les algorithmes de BFGS et de Fletcher qui faisaient partie des méthodes les plus efficaces sans stratégie de scaling, sont dépassés par les autres algorithmes lorsque la stratégie de scaling 5 est considérée.

5.2.4 Plusieurs recherches linéaires : Tableaux 33,34,35

Si on garde la même heuristique pour l'interpolation cubique, les recherches d'Armijo et de Wolfe donnent exactement les mêmes résultats après 19 itérations d'optimisation. En revanche, ces résultats différent de ceux obtenus avec la recherche de Goldstein & Price. Les résultats suivants ont été obtenus. Les algorithmes numérotés correspondent aux algorithmes de Quasi-Newton testés dans la section précédente. Excepté pour les quatre premiers tests (Tab. 33), on a pris en compte le paramètre de Biggs de Spedicato et la stratégie de scaling sélectif.

Dans la plupart des cas, on obtient de meilleurs résultats avec la recherche de Goldstein & Price. Notamment, la mise à jour de Liao se dégrade fortement avec la recherche d'Armijo (on n'a donc pas testé cette mise à jour avec une stratégie de stabilisation/scaling à ce stade). De plus, on peut remarquer que la stratégie de scaling n'est pas adaptée pour la recherche d'Armijo, car la longueur unité n'est pas acceptée systématiquement. Les deux tests sans scaling et stabilisation (BFGS, Liao) sont meilleurs de ce point de vue, même si à nombre d'évaluations donné, le critère est moins diminué. Il reste à tester les autres stratégies de scaling et d'autres heuristiques pour le choix de ρ.

Enfin, même s'il reste à tester une interpolation cubique qui utilise les valeurs de gradient calculées durant la recherche linéaire, une recherche de Wolfe est trop lourde par rapport à une recherche de Goldstein & Price qui semble aussi efficace. Ceci est peut-être la conséquence d'une fort caractère quadratique des deux problèmes traités. Pour des cas plus complexes, d'une non linéarité plus prononcée, la recherche de Wolfe se comporterait certainement mieux. Malgré tout, la recherche de Goldstein & Price s'impose pour nos problèmes où les évaluations de gradient peu rentables sont à proscrire.

5.2.5 Le paramètre α vaut 10^{-2}

On a effectué des tests similaires dans le cas où α est égal à 10^{-2}. Pour la totalité de ces tests, la recherche linéaire adoptée vérifie les conditions de Goldstein & Price.

5.2.5.1 Plusieurs algorithmes de Gradient Conjugué non Linéaire Tableaux 36, 37

Les résultats sont similaires à ceux obtenus pour $\alpha = 10^{-4}$. L'algorithme de Polak & Ribière est l'algorithme le plus performant. Cependant, on note que l'algorithme de Fletcher & Reeves est le plus efficace si on effectue seulement 4 itérations d'optimisation. Puis, il se dégrade au fur et à mesure que l'optimisation avance. Après 19 itérations, les résultats sont moins bons que ceux obtenus avec la méthode de plus profonde descente. Comme on sait qu'il est fréquent que l'algorithme de Fletcher & Reeves se comporte bien pour les cas quadratiques, cela semble suggérer que pour le problème $\mathcal{P}_2$ avec ce coefficient de pénalisation les non linéarités sont beaucoup plus importantes.

5.2.5.2 Plusieurs stratégies de "scaling" : Tableaux 38, 39, 40

Comme précédemment, l'algorithme de BFGS standard est très compétitif. Il donne par ailleurs de meilleurs résultats que l'algorithme de Polak & Ribière. Les résultats sont similaires pour toutes les stratégies à l'exception de la stratégie 5 qui mène à des résultats assez mauvais, et de la stratégie 4 qui surpasse nettement les autres stratégies. De plus, alors que la longueur unité n'est jamais acceptée par les autres stratégies, celle-ci est toujours acceptée quand elle est testée en premier par la stratégie 4. Comme on en a fait la remarque à propos de l'algorithme de Fletcher & Reeves, la stratégie 4 semble adaptée au cas faiblement non quadratique alors que la stratégie 5 serait plutôt adaptée au cas fortement non linéaire.

5.2.5.3 Plusieurs mises à jour

Sans Scaling : Tableaux 41, 42, 43, 44

Les résultats sont équivalents à ceux obtenus pour α égal à 10^{-4}. Les mises à jour de BFGS et de Fletcher donnent de bons résultats. Les mises à jour 10

et 11 donnent des résultats toujours peu similaires sans prise en compte d'une stratégie de scaling. La mise à jour de DFP est la plus mauvaise bien que la longueur unité y soit la plus souvent acceptée. Enfin, après 100 évaluations de la fonction critère, on note que les deux meilleures mises à jour sont données par les mises à jour optimales de Luksàn (7) et de Wolkowics & Dennis (9). Toutefois, si on s'autorise quelques évaluations supplémentaires, on constate d'une part que la mise à jour de Wolkowics & Dennis (9) devient moins bonne que la mise à jour de BFGS, et, que la mise à jour (11) des mêmes auteurs est aussi compétitive que BFGS.

Avec Scaling : Tableaux 45, 46, 47, 48

On a considéré la stratégie de Nocedal & Yuan qui se révèle être la meilleure lorsqu'on la couple à un algorithme de BFGS.

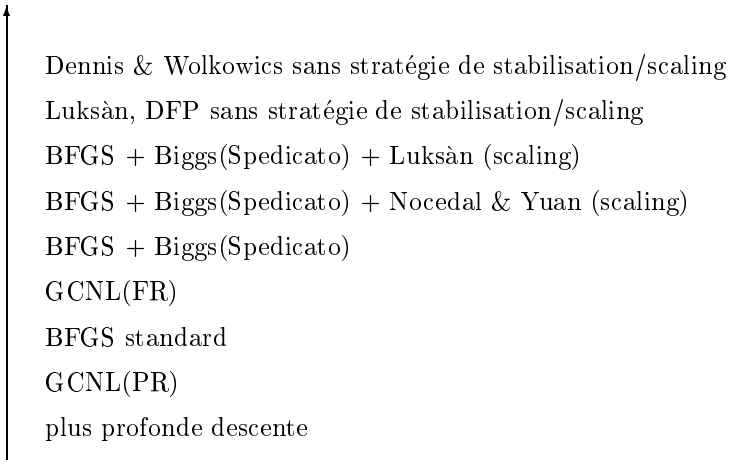
Contrairement au cas sans scaling, peut-être car on n'a pas considéré la même stratégie de scaling, les résultats sont assez différents de ceux obtenus pour α égal à 10^{-4}.

D'abord, pour la majorité des mises à jour, la direction de Newton est mieux conditionnée puisque la longueur unité est systématiquement acceptée. Ensuite, on constate que la prise en compte d'une stratégie de scaling n'implique pas toujours une meilleure performance de l'algorithme. On note enfin qu'après 100 évaluations de fonction, l'algorithme de BFGS fait partie des deux meilleurs algorithmes assez loin devant les autres algorithmes.

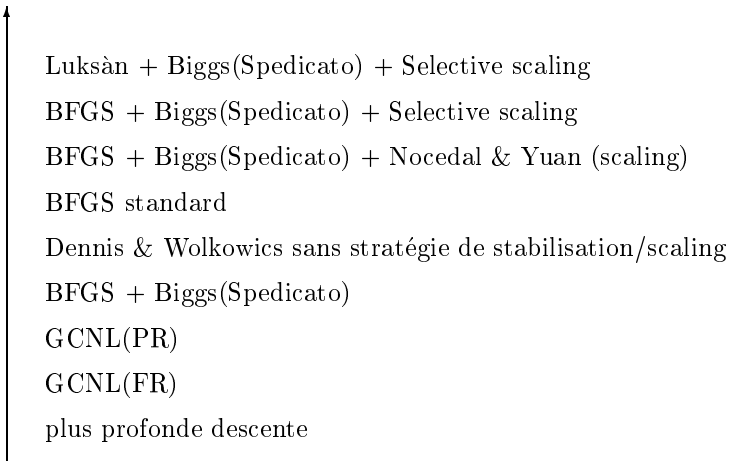
Ceci confirme que l'efficacité d'une mise à jour dépend de la stratégie de scaling adoptée et de la nature du problème. Quand le problème est plutôt quadratique, une mise à jour de BFGS couplée à des paramètres de stabilisation ρ, respectivement de scaling γ, donnés par Spedicato, respectivement Nocedal & Yuan, semble définir un algorithme efficace. En revanche, si le caractère quadratique est moins prédominant, une mise à jour de Luksàn et une stratégie de scaling sélectif paraissent plus satisfaisantes.

5.3 Tableaux récapitulatifs

Voici une classification des méthodes par ordre d'efficacité à nombre d'évaluations de la fonction critère fixé pour les deux problèmes étudiés (50 pour le problème $\mathcal{P}_1$ et 100 pour le problème $\mathcal{P}_2$, le paramètre α valant 10^{-4}).



TAB. 3 – *Test sur le problème $\mathcal{P}_1$: classification des méthodes par ordre d'efficacité à nombre d'évaluations de fonction fixé*



TAB. 4 – *Test sur le problème $\mathcal{P}_2$: classification des méthodes par ordre d'efficacité à nombre d'évaluations de fonction fixé*

6 Conclusions et Perspectives

6.1 Conclusions

Nous nous sommes proposés de présenter quelques idées simples concernant le traitement de problèmes de contrôle optimal en Mécanique des Fluides. Ayant pour objectif de traiter un tel problème en utilisant directement la simulation numérique (sans passer par l'intermédiaire d'une paramétrisation de ses résultats), et d'inclure les résultats de cette simulation dans un algorithme du premier ordre au moins, nous avons formulé quelques remarques générales concernant le processus d'optimisation à appliquer.

D'abord, du fait que les simulations numériques induisent un coût important de la fonction critère (de l'ordre de la dizaine d'heures), nous avons éliminé le recours aux méthodes autres que les méthodes de gradient, qui exigent souvent un trop grand nombre d'évaluations de la fonction. Ceci a aussi pour conséquence l'utilisation d'une méthode adjointe pour le calcul du gradient dont le temps de calcul est indépendant du nombre de paramètres. A cette fin, la différentiation automatique utilisée en mode inverse est un choix qui nous semble judicieux, car mêlant simplicité de mise en œuvre et efficacité. Le calcul du gradient obtenu est équivalent en moyenne à 5 évaluations de fonction, ce qui rend le calcul du hessien exact inenvisageable puisqu'il requiert n calculs du gradient, si n est le nombre de paramètres du problème traité. Notre seule alternative, si on veut appliquer une méthode du second ordre, est donc la construction d'une approximation du hessien à l'aide de matrices de Quasi-Newton, au besoin à mémoire limitée si le nombre de paramètres de contrôle dépasse le millier. Remarquons enfin que les algorithmes des régions de confiance et de Newton tronqué, malgré leur robustesse et efficacité lors du traitement les problèmes difficiles, semblent peu adaptés au cas d'un hessien seulement approché (quoique des avancées récentes dans ce domaine puissent amener à moduler cette opinion).

Pour tester les différentes approches, nous avons considéré deux problèmes simplifiés issus de la thermique. Le premier, stationnaire, a vu la comparaison de plusieurs algorithmes d'optimisation ainsi que de trois stratégies de calcul du gradient (différences finies, gradient automatique, gradient automatique simplifié). Le second, transitoire, a fait l'objet de quelques comparaisons d'algorithmes d'optimisation avec un gradient calculé à l'aide du code adjoint construit. Schématiquement, nos conclusions sont de deux types.

En premier lieu, pour le cas stationnaire, on a fait ressortir le gain en précision et efficacité du calcul du gradient par l'approche adjointe par rapport à l'approche par différences finies. On a pu aussi noter la précision de la formule simplifiée du gradient lorsqu'il est calculé par différentiation automatique quand on se situe proche de l'état stationnaire.

En second lieu, pour l'ensemble des cas stationnaire et transitoire, les tests numériques ont fait ressortir trois points importants

- la supériorité (attendue ...) de l'algorithme de Quasi-Newton par rapport aux algorithmes de Gradient Gonjugué non Linéaire - que cela soit par

rapport à l'algorithme de Fletcher & Reeves (meilleur pour le problème $\mathcal{P}_1$), ou par rapport à l'algorithme de Polak & Ribière (meilleur, lui, pour le problème $\mathcal{P}_2$) - et aux méthodes de tenseur,

- au sein des algorithmes de Quasi-Newton, la supériorité, à nombre d'évaluations de fonction fixé, de certaines mises à jour récentes (approximation de l'inverse du hessien), par rapport à la mise à jour souvent employée de BFGS. Notamment, on peut citer la mise à jour (11) de Dennis & Wolkowics qui s'est révélée dans les deux problèmes au moins aussi efficace que la mise à jour de BFGS. Signalons cependant qu'à notre connaissance, il n'existe pas de version "à mémoire limitée" pour la mise à jour de Dennis & Wolkowics, ce qui peut conduire à lui préférer BFGS ou SR1,
- l'apport d'une recherche de Goldstein & Price par rapport à une recherche d'Armijo pour l'efficacité de l'algorithme.

A l'issue de ce travail, le traitement de problèmes d'optimisation stationnaire par rapport à un grand nombre de variables semble tout à fait réalisable en utilisant la formulation simplifiée du gradient calculé par différentiation automatique et un algorithme de Quasi-Newton (à mémoire limitée, le cas échéant).

En revanche, et ce n'est pas une surprise, l'optimisation transitoire paraît d'un accès plus difficile même si l'algorithme de Griewank semble rendre envisageable le traitement des cas industriels de moyenne taille (300000 nœuds) sur des stations de travail puissantes (multi-processeurs).

6.2 Perspectives

Le présent travail a pour but de préparer une seconde étape consistant en la différentiation du code vitesse afin de prendre en compte le *couplage* thermo-hydraulique. Dans le même temps, des tests sur des géométries plus complexes pour des problèmes thermiques permettraient de tester la robustesse de l'approche décrite ci-dessus.

Comme extension possible, signalons que, reprenant une idée mise en application à Météo France [16], il paraît intéressant de faire varier la taille du maillage durant le processus d'optimisation. On peut par exemple travailler avec des maillages de plus en plus fins ou travailler avec un maillage assez grossier et réinitialiser l'optimisation avec le maillage fin périodiquement.

De plus, dans le cas où l'optimisation fait intervenir un nombre assez important de variables, la méthode de Quasi-Newton à mémoire limitée (M1QN3) implémentée dans MODULOPT semble la plus adéquate. Pour les problèmes de petite taille, une extension de notre travail consisterait à calculer le hessien par différentiation automatique afin d'utiliser les directions de courbure négative et de tester l'algorithme DINEMO [8]. De même, l'intégration du hessien (ou d'une de ses approximations) dans un algorithme des régions de confiance, du fait de sa faible sensibilité au bruit, peut se révéler efficace. En parallèle, pour ces problèmes de petite taille, une recherche non monotone apparaît comme une extension intéressante des algorithmes monotones.

Enfin, dans l'optique de problèmes à caractère industriel tel que celui décrit en introduction, rappelons qu'en plus d'atteindre un état cible donné on aimerait l'atteindre le plus vite possible (ou réaliser un compromis entre ces deux objectifs). Le problème mathématique est maintenant un problème de contrôle en temps minimal. Or, les stratégies d'attaque de ces problèmes consistent en la suite de problèmes d'optimisation tels que ceux étudiés pour nos résultats numériques [68]. Il nous reste à tester les algorithmes d'optimisation les plus performants dans de telles situations.

6.3 Remerciements

Nous remercions tout particulièrement le responsable du code PAMFLOW^{TM} à ESI France, Philippe Ravier, sans qui ce travail n'aurait pas pu se faire. Nous tenons aussi à remercier Isabelle Charpentier pour plusieurs discussions très enrichissantes sur la différentiation automatique et pour une relecture attentive de ce document, ainsi que Christèle Faure et toute l'équipe d'ODYSSEE (INRIA) pour avoir mis ce code à notre disposition.

Références

[1] Beux F. et Dervieux A. *A Hierarchical Approach for Shape Optimization.* Rapport de Recherche INRIA RR-1868, `http://www.inria.fr/RRRT/publications-fra.html`, Mars 1993.

[2] Beux F. et Marco N. *Multilevel Optimization : Application to Shape Optimum Design with a One-Shot Method.* Rapport de Recherche INRIA RR-2068, `http://www.inria.fr/RRRT/publications-fra.html`, 1993.

[3] Biggs M.C. *Minimization Algorithms Making Use of Nonquadratic Properties of the Objective Function.* J. Inst. Maths. Applics., 8, 315-327, 1971.

[4] Bonnans J.-F., Gilbert J.-C., Lemaréchal C. et Sagastizàbal C. **Optimisation Numérique**. Springer, 1997.

[5] Buckley A. and LeNir A. *A Variable Storage Algorithm for Function Minimization.* ACM Transactions on Mathematical Software 11, 2, pp. 103-119, 1985.

[6] Bunch J.R. and Parlett B.N. *Direct Methods for Solving Symmetric Indefinite Systems of Linear Equations.* SIAM Journal on Numerical Analysis, 8(4), 639-655, 1971.

[7] Byrd R.H., Nocedal J. and Schnabel R.B. *Representation of Quasi-Newton Matrices and their Use in Limited Memory Methods.* Mathematical Programming 63, 4, pp. 129-156, 1994.

[8] Byrd R.H., Nocedal J. and Zhu C. *Towards a Discrete Newton Method with Memory for Large Scale Optimization.* **Nonlinear Optimization and Applications**, G. Di Pillo and F. Giannessi, eds. Plenum., 1995.

[9] Carter R.G. *On the Global Convergence of Trust Region Algorithms using Inexact gradient Information.* SIAM Journal of Numerical Analysis, 28, pp. 251-265, 1991.

[10] Carter R.G. *A Worst-Case Example Using Linesearch Methods for Numerical Optimization with Inexact Gradient Evaluations.* Preprint MCS-P283-1291, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

[11] Charpentier I. *Génération de code adjoints : Traitement de la trajectoire du modèle direct.* Rapport de Recherche INRIA RR-3405, `http://www.inria.fr/RRRT/publications-fra.html`, avril 1998.

[12] Christianson B. *Reverse Accumulation and Attractive Fixed Points.* Optimization Methods and Software, 3:311-326, 1994.

[13] Thomas F. Coleman and Gudbjorn F. Jonsson. The efficient computation of structured gradients using automatic differentiation. *SIAM J. Sci. Comput.*, 20(4):1430–1437, 1999.

[14] Conn A.R., Gould N., Sartenaer A., and Toint Ph.L. *On Iterated-Subspace Minimization Methods for Nonlinear Optimization.* in **Linear and nonlinear conjugate gradient-related methods**, eds. L. Adams, JL. Nazareth, SIAM, 1996.

[15] Corliss G., Bishof C., Carle A., Griewank A., and Williamson K. *Derivative Convergence for Iterative Equation Solvers.* Optimization Methods and Software, 2:321-355, 1993.

[16] Courtier P., Thépaut J.-N., and Hollingsworth. *A Strategy for operational Implementation of 3d-VAR using an Incremental Approach.* Q. J. R. Meteorol.Soc.,120, pp. 1367-1387, 1994.

[17] Dembo R.S. and Steihaug T. *Truncated-Newton Algorithms for Large Scale Unconstrained Optimization.* Mathematical Programming, 26,190-212, 1983.

[18] Deng N.Y., Xiao Y., and Zhou F.J. *Nonmonotonic Trust Region Algorithm.* Journal of Optimization Theory and Application, 76:259-285, 1993.

[19] Dennis J.E., Jr. and Wolkowicz H. *Sizing and Least Change Secant Methods.* SIGNUM vol. 10, 1993.

[20] Desai R.and Patil R. *SALO: Combining Simulated Annealing and Local Optimization for Efficient Global Optimization.* In Proceedings of the 9th Florida Al Research Symposium (FLAIRS-'96), Key West, FL, pp. 233-237, June 1996.

[21] Elster C. and Neumaier A. *A trust Region Method for the optimization of Noisy Functions.* Computing 58, pp. 31-46, 1997.

[22] Eyssette F., Faure C., Gilbert J.C., Rostaing-Schmidt N. *Applicabilité de la différentiation automatique à un système d'équations aux dérivées partielles règissant les phénomènes thermohydrauliques dans un tube chauffant.* Rapport de Recherche INRIA RR-2735, `http://www.inria.fr/RRRT/publications-fra.html`, 1996.

[23] Faure C. *Documentation succinte d'Odyssée version 1.6.* INRIA, `http://www.inria.fr/RRRT/publications-fra.html`, décembre 1996.

[24] Faure C. et Papeguay Y. *Odyssée Version 1.6, The User's Reference Manual.* Rapport de Recherche INRIA RR-0211, `http://www.inria.fr/RRRT/publications-fra.html`, novembre 1997.

[25] Fayez Khalfan H., Byrd R.H. and Schnabel R.B. *A Theoretical and Experimental study of the Symmetric Rank One Update.* SIAM Journal on Optimization 3, pp. 1-24, 1993.

[26] Ferris M.C., Lucidi S., and Roma M. *Nonmonotone Curvilinear Linesearch Methods for Unconstrained Optimization.* Computational Optimization and Applications, 1995.

[27] Fletcher R. **Practical Methods of Optimization**. John Wiley & Sons, Chichester, 1987 (second edition).

[28] Fletcher R. *Computing Sparse Hessian and Jacobian Approximations with Optimal Hereditary Properties.* Springer. IMA Vol. Math. Appl. 93, 37-52, 1997.

[29] J.A. Ford and L.A. Moghrabi. Alternating multi-step quasi-Newton methods for unconstrained optimization. *J. Comput. Appl. Math.*, 82(1-2):105–116, 1997.

[30] Ford J.A. and Moghrabi I.A. *Multi-step Quasi-Newton Methods for Optimization*. Journal of Comp. Appli. Math., 50, pp. 305-323, 1994.

[31] Ford J.A. and Moghrabi I.A. *Multi-step Quasi-Newton Optimization Algorithms which utilize Curvature Information*. Technical Report CSM-251, Department of Computer Science, University of Essex, ftp://ftp.essex.ac.uk/pub/csc/technical-reports/CSM-251.ps.Z, 1995.

[32] Gay D.M. *More AD of Nonlinear AMPL Models: Computing Hessian Information and Exploiting Partial Separability*. Proceedings of the Second International Workshop on Computational Differentiation, SIAM. 173-184, 1996.

[33] Giannakoglou K.C. *A Design Method For Turbine Blades Using Genetic Algorithms On Parallel Computers*. John Wiley & Sons, Ltd, 1998.

[34] Gibson T., O'Leary D.P., and Nazareth L. *BFGS with Update Skipping and Varying Memory*. Technical Report CS-TR-3663, University of Maryland, `http://www/cs/umd.edu/ oleary/tr.html`, 1996.

[35] Gilbert J.C. *Automatic Differentiation and Iterative Processes*. Optimization Methods and Software, 1,13-21, 1992.

[36] Gilbert J.C. and Nocedal J. *Global Convergence Properties of Conjugate Gradient Methods for Optimization*. SIAM Journal on Optimization,2, 21-42, 1992.

[37] Gilbert J.C., Le Vey G. , Masse J. *La différentiation automatique de fonctions représentées par des programmes*. Rapport de Recherche INRIA RR-1557, `http://www.inria.fr/RRRT/publications-fra.html`, 1991.

[38] Gill P.E., Murray W., and Wright M.H. *Practical Optimization*. Academic Press, New York, 1981.

[39] Glover F., Taillard E., and de Werra D. *A User's Guide to Tabu Search*. Annals of Operations research 41, 3-28, 1993.

[40] Goldberg D.E. **Genetic Algorithms**. 1989.

[41] Golub G.H., Hansen P.C. and O'Leary D.P. *Tichonov Regularization and Total Least Squares*. Technical Report CS-TR-3829, University of Maryland, `http://www/cs/umd.edu/ oleary/tr.html`, 1997.

[42] Gould N.I.M. and Nocedal J. *The Modified Absolute Value Factorization Norm for Trust Region Minimization*. Technical Report RAL-TR-97-071, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, ftp://thales.math.fundp.ac.be/pub/reports/TR98-71.ps, 1997.

[43] Gould N.I.M., Lucidi S., Roma M. and Toint P. *A Linesearch Algorithm with Memory for Unconstrained Optimization*. Technical Report RAL-TR-98-03, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, ftp://thales.math.fundp.ac.be/pub/reports/TR98-03.ps, 1993.

[44] Gould N.I.M., Lucidi S., Roma M., and Toint Ph.L. *Exploiting Negative Curvature Directions in Linesearch Methods for*

Unconstrained Optimization. Technical Report RAL-TR-97-018, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, ftp://thales.math.fundp.ac.be/pub/reports/TR97-18.ps, 1997.

[45] Gould N.I.M., Lucidi S., Roma M., and Toint Ph.L. *Solving the Trust Region Subproblem using the Lanczos Method.* Technical Report RAL-TR-97-028, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, ftp://thales.math.fundp.ac.be/pub/reports/TR97-28.ps, 1997.

[46] Griewank A. **On Automatic Differentiation**. Mathematical Programming: Recent Developments and Applications (Kluwer Academic Publishers)4, 83-108, 1989.

[47] Griewank A. *Achieving Logarithmic Growth of Temporal and Spatial Complexity in Reverse Automatic Differentiation.* Optimization Methods and Software4, 1(1):35-54, 1992.

[48] Griewank A. and Toint Ph.L. *On the Unconstrained Optimization of Partially Separable Objective Functions.* in Nonlinear Optimization 1981 (M.J.D. Powell, ed.), Academic Press (London), pp 301-312, 1981.

[49] Griewank A. and Walther A. *Treeverse: An Implementation of the Checkpointing for the Reverse or Adjoint Mode of Differentiation.* Technical Report, 1992.

[50] Grippo L. and Lucidi S. *A Globally Convergent Version of the Polak-Ribière Conjugate Gradient Method.* Math. Program. 78A, No.3, 375-391, 1997.

[51] Grippo L., Lampariello F., and Lucidi S. *A Nonmonotone Line Search Technique for Newton's Method.* SIAM Journal on Numerical Analysis, 23(4), 707-716, 1986.

[52] Grippo L., Lampariello F., and Lucidi S. *A Truncated Newton Method with Nonmonotone Linesearch for Unconstrained Optimization.* Journal of optimization Theory and Applications, 60:401-419, 1989.

[53] Grippo L., Lampariello F., and Lucidi S. *A Class of Nonmonotone Stabilization Methods in Unconstrained Optimization.* Numerische Mathematik, 59:779-805, 1991.

[54] Rendel F. and Wolkowicz H. *A Semi-definite Framework for Trust region Subproblems with Applications to Large Scale Minimization.* Mathematical programming, 77, pp. 273-299, 1997.

[55] Higham N.J. *Stability of the Diagonal Pivoting Method with Partial Pivoting.* SIAM J. Matrix Anal. Appl. 18, No.1, 52-65, 1997.

[56] Ingber L. *Very Fast Simulated Re-Annealing.* Journal on Mathl. Comput. Modelling, 12, pp.967-973, 1989.

[57] Ingber L. *Adaptative Simulated Annealing (ASA).* Lester Ingber Research, McLean, VA, 1993.

[58] Ingber L. *Simulated Annealing: Practice versus Theory.* Journal on Math. Comput. Modelling, 8, pp.29-57, 1993.

[59] Angelo Iollo, Geojoe Kuruvila, and Shlomo Ta'asan. Pseudotime method for shape design of Euler flows. *AIAA J.*, 34(9):1807–1813, 1996.

[60] Jameson A. *Aerodynamic Design via Control Theory.* Lect. Notes Eng. 43, 377-401, 1989.

[61] Jameson A. *Optimum Aerodynamic Design Using Computational Fluid Dynamics and Control Theory.* AIAA Paper 95-1729, June 1995.

[62] Jones M.T. and Plassmann P.E. *An improved Incomplete Cholesky Factorization.* ACM Transactions on Mathematical Software, 21, pp. 5-17, 1995.

[63] Juedes J. *A Taxonomy of Automatic Differentiation Tools.* Proceedings of the Workshop on Automatic Differentiation of Algorithms: Theory, Implementation, and Application, 315-330, 1991.

[64] Luksàn L. *Computational Experience with Improved Variable Metric Methods for Unconstrained Minimization.* Kybernetika, Vol.26, pp. 415-431, 1990.

[65] Luksàn L. *Computational Experience with Known Variable Metric Updates.* J. Optimization Theory Appl. 83, No.1, 27-47, 1994.

[66] Luksàn L. *Simple Scaling for Variable Metric Updates.* Technical Report V-611, Institute of Computer Science, Academy of Sciences of the Czech Republic, `http://hypatia.dcs.qmw.ac.uk/sites/cz/uivt.cas.cz/papers/v611-92.ps.gz`, May 1995.

[67] Lalee M. and Nocedal J. *Automatic Column Scaling Strategies for Quasi-Newton Methods.* SIAM J. Optim. 3, No.3, 637-653, 1993.

[68] Le C.T. et Noailles J. *Contrôle en temps minimal et transfert orbital à très faibles poussées.* Équations aux dérivées partielles et applications. Articles dédiés à J.L. Lions, Elsevier Paris, 1996.

[69] Liao A. *Modifying BFGS Method.* Oper. Res. Lett. 20, No.4, 171-177, 1997.

[70] Lin C.-J. and Moré J.J. *Incomplete Cholesky Factorizations with Limited Memory.* Technical Report ANL/MCS-P682-0897, Argonne National Laboratory, Illinois, `http://www.library.anl.gov/`, 1997.

[71] Liu D.C. and Nocedal J. *On the Limited Memory BFGS Method for Large Scale Optimization.* Mathematical Programming, 45, 503-520, 1989.

[72] Lucidi S., Rochetich F., and Roma M. *Curvilinear Stabilization Techniques for Truncated Newton Methods in Large Scale Unconstrained Optimization.* SIAM Journal on Optimization, 8, 1998.

[73] Lucidi S., Roma M. *Numerical Experiences with New Trucated Newton Methods in Large Scale Unconstrained Optimization.* Kluwer Academic Publishers, 1995.

[74] Mantel B., Peigin S., Périaux J. and Timchenko S. *A Heat Flux Optimization Using Genetic Algorithms.* John Wiley & Sons, Ltd, 1998.

[75] Masmoudi N. and Massat C. *A new Global Optimization Algorithm.* CERFACS, ftp://ftp.cerfacs.fr/pub/shppt/exchanges/BIT/article.ps, 1996.

[76] Morales J.L. and Nocedal J.J. *Automatic Preconditioning by Limited Memory Quasi-Newton Updating.* Technical Report, Northwestern University, `http://www.ece.nwu.edu/ñocedal/recent.html`, 1998.

[77] Moré J.J. *Recent Developments in Algorithms and Software for Trust Region Methods.* In A. Bachem, M. Grötschel, B. Korte, éditeurs, Mathematical Programming, the State of the Art, pages 258-287. Springer-Verlag, Berlin, 1983.

[78] Moré J.J. and Thuente D.J. *Line Search Algorithms with Guaranted Sufficient Decrease.* ACM Transactions on Mathematical Software, 20(3): 286-307, 1994.

[79] Moré J.J. and Wright S.J. **Optimization software Guide**. Frontiers in Applied Mathematics, SIAM, 1993.

[80] Morgenstein, J. *How to compute fast a function and all its derivatives, a variation of the Theorem of Baur-Strassen, 16:60-62.* SIGACT News, 1985.

[81] Muhlenbein H., and Schlierkamp-Voosen D. *Predictive Models for the Breeder Genetic Algorithm: Continuous Parameter Optimization.* Evolutionary Computation, 1, 1993.

[82] Muhlenbein H., Sconish M., and Born J. *The Parallel Genetic Algorithm as Function Optimizer.* Parallel Computing, 17: 619-632, 1991.

[83] Nash S.G. *Newton-type Minimization via Lanczos Method.* SIAM Journal on Numerical Analysis, 21:770-788, 1984.

[84] Neumaier A. *Molecular Modeling of Proteins and Mathematical Prediction of Protein Structure.* SIAM Journal on Numerical Analysis, 21:770-788, 1984.

[85] Nocedal J. *Theory of Algorithms for Unconstrained Optimization.* Acta Numerica, 1992.

[86] Nocedal J. *Large Scale Unconstrained Optimization.* Inst. Math. Appl. Conf. Ser., New Ser. 63, 311-338, 1997.

[87] Nocedal J. and Yuan Y. *Combining Trust Region and Line search Techniques.* Advances in Nonlinear Programming, Kluwer, ed. Y. Yuan, pp. 153-175, 1998.

[88] Oren S.S. *On the Selection of Parameters in Self-Scaling Variable Metric Algorithms.* Mathematical Programming, 7, 351-367, 1974.

[89] Paige C.C. and Saunders M.A. *Solution of Sparse Indefinite Systems of Linear Equations.* SIAM Journal on Numerical Analysis, 12:617-629, 1975.

[90] Pardalos P.M., Pitsoulis L., Mavridou T., and Resende M.G.C. *Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRAS.* In A. Ferreira and J. Rolin, editors, **Parallel Algorithms for Irregularly Structured Problems**, Proceedings of the Second International Workshop-Irregular'95, volume 980 of Lectures Notes in Computer Science, pp. 317-331. Springer-Verlag, 1995.

[91] Piela L., Kostrowicki J., and Scheraga H.A. *The Multiple Minima Problem in the Conformational Analysis of Molecules. Deformation of the Protein*

Energy Hypersurface by the Diffusion Equation Method. J. Phys. Chem., 93, pp. 3339-3346, 1989.

[92] Radcliffe N.J. and Surry P.D. *Formal Memetic Algorithms.* Technical Report, Edimburgh Parallel Computing Center, ftp://ftp.epcc.ed.ac.uk/pub/tr/94/tr9415.ps.Z, 1994.

[93] Resende M.G.C. *Greedy Randomized Adaptative Search Procedures (GRASP).* AT & T Labs Research Technical Report: 98.41.1, `http://akpublic.research.att.com/library/trs/`, December 1998.

[94] Rochat Y. and Taillard E. *Probabilistic Diversification and Intensification in Local Search for Vehicle Routing.* Journal of heuristics 1, pp. 147-167, 1995.

[95] Saad Y. *Krylov Subspace Methods: Theory, Algorithms and Applications.* In Computational Methods in Applied Science and Engineering, INRIA 29 January - 2 February, 1990.

[96] Santos S.A. and Sorensen D.C. *A New Matrix-free Algorithm for the Large Scale Trust Region Subproblem.* Technical Report CRPC-TR95551, Rice University, Houston, Texas, `http://www.crpc.rice.edu/CRPC/softlib/TRs_online.html`, 1995.

[97] Sartenaer A. *Automatic Determination of an Initial Trust Region in Nonlinear Programming.* SIAM Journal on Scientific Computing, 18(6), 1788-1804, 1997.

[98] Schnabel R.B. and Chow T.-T. *Tensor Methods for Unconstrained Optimization using Second Derivatives.* SIAM Journal on Optimization, 1(3):293-315, 1991.

[99] Schnabel R.B.and Eskow E. *A new Modified Cholesky Factorization.* SIAM Journal on Scientific and Statistical Computing, 11:1136-1158, 1991.

[100] Seigel D. *Implementing and Modifying Broyden Class of Updates for Large Scale Optimization.* Report DAMPT NA12, University of Cambridge, 1992.

[101] Steihaug T. *The Conjugate Gradient Method and Trust Region in Large Scale Optimization.* SIAM Journal on Numerical Analysis, 20, 626-637, 1983.

[102] Ta'asan S. *Pseudo-Time Methods for Constrained Optimization Problems Governed by PDE.* Technical Report 95-32, ICASE, `http://www.icase.edu/library/ncstrl_icase.html`, 1995.

[103] Ta'asan S., Kuruvila and Salas M.D. *Aerodynamic Design and Optimization in One Shot.* In 30*th* Aerospace Sciences Meeting and Exibit, AIAA 92-005, Jan. 1992.

[104] Toint Ph.L. *An Assesment of Non-Monotone Linesearch Techniques for Unconstrained Optimization.* SIAM J. Sci. Comput. 17, No.3, 725-739, 1996.

[105] Toint Ph.L. *A Non-Monotone Trust-Region Algorithm for Nonlinear Optimization Subject to convex Constraints.* Math. Program. 77A, No.1, 69-94, 1997.

[106] Torczon V. *On the Convergence of the Multidimensional Search Algorithm.* SIAM Journal on Optimization 1, 123-145, 1991.

[107] Edward Tsang and Chris Voudouris. Fast local search and guided local search and their application to British Telecom's workforce scheduling problem. *Oper. Res. Lett.*, 20(3):119–127, 1997.

[108] Voudouris C. and Tsang E. *Function Optimization using Guided Local Search.* Technical Report CSM-249, Department of Computer Science, University of Essex, ftp://ftp.essex.ac.uk/pub/csc/technical-reports/CSM-249.ps.Z, 1995.

[109] Wolkowicz H. and Zhao Q. *An All Inclusive Efficient Trust Region of Updates for Least Change Secant Methods.* SIAM J. Optim. 5, No.1, 172-191, 1995.

Algorithme	SD	FR	PR	PR stabilisé
Nb itér.	19	19	16	19
Critère J	$4.38e^{-2}$	$2.59e10^{-14}$	$1.02e^{-8}$	$1.73e^{-11}$
Nb éval.	105	105	84	100
$\| \nabla J(x_f) \|$	1.02	$2.63e10^{-6}$	$1.09e^{-4}$	$1.86e^{-4}$

TAB. 5 – *Comparaison de plusieurs algorithmes de Gradient Conjugué non Linéaire (1)*

Nbre éval.	SD	FR	PR	PR stabilisé
20	36.39 (4)	1.08 (4)	0.66 (4)	0.66 (4)
50	1.26 (9)	$5.29e10^{-8}$ (9)	$1.09e^{-4}$ (9)	$1.09e^{-4}$ (9)

TAB. 6 – *Comparaison de plusieurs algorithmes de Gradient Conjugué non Linéaire (2)*

Algorithme	1	2	3	4	5	6	7
Nb itér.	19	12	13	12	12	12	14
Critère J	$4.47e^{-9}$	$1.48e^{-13}$	$9.24e^{-16}$	$2.7e^{-12}$	$1.99e^{-13}$	$1.48e^{-13}$	$1.41e^{-12}$
Nb éval.	102	61	59	51	59	56	60
$\| \nabla J(x_f) \|$	$7.66e^{-4}$	$3.53e^{-6}$	$3.26e^{-7}$	$2.3e^{-5}$	4.42^{-6}	3.53^{-6}	1.12^{-5}
$\alpha_k = 1$ accepté	2	3	5	7	4	3	7

TAB. 7 – *Comparaison de plusieurs stratégies de scaling couplées à une mise à jour BFGS (1)*

Nbre éval.	1	2	3	4
20	$1.08e^{-2}$ (4)	$6.02e^{-2}$ (4)	0.46 (5)	59.36 (5)
50	5.61^{-7} (10)	$2.45e^{-13}$ (11)	$1.64e^{-15}$ (12)	$2.70e^{-12}$ (12)

TAB. 8 – *Comparaison de plusieurs stratégies de scaling couplées à une mise à jour BFGS (2a)*

Nbre éval.	5	6	7
20	$6.02e^{-2}$ (4)	$6.02e^{-2}$ (4)	20.86 (5)
50	$5.36e^{-13}$ (11)	$2.45e^{-13}$ (11)	$3.42e^{-12}$ (12)

TAB. 9 – *Comparaison de plusieurs stratégies de scaling couplées à une mise à jour BFGS (2b)*

Algorithme	1	2	3	4	5	6
Nb itér.	19	19	12	19	14	19
Critère J	$4.46e^{-9}$	$4.39e^{-16}$	$1.16e^{-14}$	$1.65e^{-16}$	$2.46e^{-15}$	$5.61e^{-17}$
Nb éval.	102	98	61	97	70	100
$\| \nabla J(x_f) \|$	$7.48e^{-8}$	$1.53e^{-7}$	$1.03e^{-6}$	8.072^{-8}	$5.77e^{-7}$	$5.48e^{-8}$
$\alpha_k = 1$ accepté	2	4	3	4	4	4

TAB. 10 – *Comparaison de diverses mises à jour sans stratégie de scaling (1a)*

Algorithme	7	8	9	10	11
Nb itér.	19	19	19	12	15
Critère J	$1.66e^{-16}$	$5.86e^{-18}$	$2.44e^{-16}$	$5.87e^{-17}$	$1.83e^{-17}$
Nb éval.	98	104	98	52	70
$\| \nabla J(x_f) \|$	$8.11e^{-8}$	$7.29e^{-8}$	1.032^{-8}	$3.27e^{-7}$	$5.04e^{-9}$
$\alpha_k = 1$ accepté	4	4	4	5	5

TAB. 11 – *Comparaison de diverses mises à jour sans stratégie de scaling (1b)*

Nb éval.	1	2	3	4
20	$1.08e^{-2}$ (4)	$9.98e^{-3}$ (4)	$9.91e^{-3}$ (4)	$1.14e^{-2}$ (4)
50	$5.61e^{-7}$ (10)	$3.80e^{-14}$ (10)	$3.07e^{-14}$ (10)	$9.96e^{-15}$ (10)

TAB. 12 – *Comparaison de diverses mises à jour sans stratégie de scaling (2a)*

Nb éval.	5	6	7	8
20	$9.98e^{-3}$ (4)	$5.71e^{-3}$ (4)	$1.15e^{-2}$ (4)	$9.98e^{-3}$(4)
50	$2.31e^{-14}$ (10)	$5.31e^{-15}$ (10)	$1.01e^{-14}$ (10)	$2.39e^{-15}$ (10)

TAB. 13 – *Comparaison de diverses mises à jour sans stratégie de scaling (2b)*

Nb éval.	9	10	11
20	$1.01e^{-2}$ (4)	0.37 (5)	0.37 (5)
50	$1.97e^{-14}$ (10)	$5.87e^{-17}$ (12)	$5.87e^{-17}$ (12)

TAB. 14 – *Comparaison de diverses mises à jour sans stratégie de scaling (2c)*

Stratégie	GA	GAS	DF1	DF2	DF3
Nb itér.	19	19	3	19	19
Critère J	$1.38e^{-2}$	$1.38e^{-2}$	39.54	$6.21e^{-3}$	$1.15e^{-2}$
Nb éval.	105	105	23	165	152
$\| \nabla J(x_f) \|$	1.02	1.02	247.49	2.21	0.95
f après 50 éval.	1.26 (9)	1.26 (9)		8.28 (6)	6.58 (6)

TAB. 15 – *Comparaison des stratégies de calcul du gradient au sein de l'algorithme de plus profonde descente*

Stratégie	GA	GAS	DF1	DF2	DF3
Nb itér.	19	19	4	5	10
Critère J	$1.73e^{-11}$	$1.73e^{-11}$	5.701	1.92^{-3}	$6.89e^{-9}$
Nb éval.	100	100	31	40	83
$\| \nabla J(x_f) \|$	$1.86e^{-9}$	$1.86e^{-9}$	111.94	17.18	$1.012e^{-2}$
f après 50 éval.	$1.09e^{-2}$ (9)	$1.09e^{-2}$ (9)			$5.79e^{-4}$ (6)

TAB. 16 – *Comparaison des stratégies de calcul du gradient au sein de l'algorithme GCNL Polak & Ribière*

Stratégie	GA	GAS	DF1	DF2	DF3
Nb itér.	13	13	3	8	8
Critère J	$9.24e^{-16}$	$9.24e^{-16}$	1.55	$3.12e^{-3}$	$2.46e^{-07}$
Nb éval.	59	59	20	56	53
$\| \nabla J(x_f) \|$	$3.26e^{-7}$	$3.26e^{-7}$	230.14	0.58	$3.65e^{-2}$
$\alpha_k = 1$ accepté	5	5		3	4
f après 50 éval.	$1.64e^{-15}$ (12)	$1.64e^{-15}$(12)		$3.57e^{-3}$(8)	$8.55e^{-6}$(7)

TAB. 17 – *Comparaison des stratégies de calcul du gradient au sein de l'algorithme BFGS*

Stratégie	GA	GAS	DF1	DF2	DF3
Nb itér.	15	15	2	11	10
Critère J	$1.83e^{-17}$	$1.83e^{-17}$	6.72	$5.39e^{-5}$	$9.88e^{-8}$
Nb éval.	70	51	14	68	62
$\| \nabla J(x_f) \|$	$5.04e^{-9}$	$5.04e^{-9}$	133.26	2.00	$5.76e^{-3}$
$\alpha_k = 1$ accepté	5	5		5	6
f après 50 éval.	$5.87e^{-17}$ (12)	$5.87e^{-17}$ (12)		$9.21e^{-3}$(8)	$1.25e^{-2}$ (8)

TAB. 18 – *Comparaison des stratégies de calcul du gradient au sein de l'algorithme Dennis & Wolkowics (11)*

Algorithme	SD	FR	PR	PR stabilisé
Critère J	11.31	21.83	5.982	6.018
Nb éval.	111	113	114	110
$\| \nabla J(x_f) \|$	1.518	4.202	1.022	1.806

TAB. 19 – *Comparaison de plusieurs algorithmes de Gradient Conjugué non Linéaire (1)*

Nbre éval.	SD	FR	PR	PR stabilisé
25	29.74 (4)	59.33 (4)	31.32 (4)	31.62 (4)
50	17.89 (9)	40.47 (8)	14.65 (8)	12.37 (9)
100	11.98 (17)	27.24 (17)	6.932 (17)	6.503 (17)

TAB. 20 – *Comparaison de plusieurs algorithmes de Gradient Conjugué non Linéaire (2)*

$\underline{\beta_2 = 0.9}$

Algorithme	SD	FR	PR	PR stabilisé
Critère J	11.41	38.601	6.604	7.195
Nb éval.	40	42	37	41
Crit. 40 éval.	11.41	38.64	6.604	7.59

$\underline{\beta_2 = 0.5}$

Algorithme	SD	FR	PR	PR stabilisé
Critère J	10.72	16.36	5.460	5.411
Nb éval.	44	52	41	41
Crit. 40 éval.	11.26	18.36	5.64	5.81

$\underline{\beta_2 = 0.2}$

Algorithme	SD	FR	PR	PR stabilisé
Critère J	11.81	14.1	5.33	5.39
Nb éval.	48	70	51	51
Crit. 40 éval.	13.55	19.34	6.39	6.39

TAB. 21 – *Comparaison de plusieurs algorithmes GCNL pour des recherches linéaires de plus en plus précises*

Algorithme	1	2	3	4	5	6	7
Critère J	3.56	7.625	3.881	3.620	2.569	16.708	2.971
Nb éval.	106	101	102	101	108	102	104
$\| \nabla J(x_f) \|$	1.036	2.333	1.820	2.483	1.879	2.577	2.781
$\alpha_k = 1$ accepté	9	13	13	13	14	7	11

TAB. 22 – *Comparaison de plusieurs stratégies de scaling (1)*

Nbre éval.	1	2	3	4
25	27.23 (4)	28.51 (4)	25.00 (4)	31.74 (4)
50	11.11 (9)	13.38 (9)	10.58 (9)	9.478 (9)
100	3.891 (18)	7.625 (19)	3.881 (102)	3.620 (19)

TAB. 23 – *Comparaison de plusieurs stratégies de scaling (2a)*

Nbre éval.	5	6	7
25	29.22 (4)	40.35 (4)	24.43 (4)
50	8.406 (9)	31.73 (9)	9.383 (9)
100	3.045 (17)	17.453 (18)	3.466 (18)

TAB. 24 – *Comparaison de plusieurs stratégies de scaling (2b)*

Algorithme	1	2	3	4	5	6
Critère J	3.56	8.938	5.138	3.764	8.658	4.608
Nb éval.	106	103	107	106	102	101
$\| \nabla J(x_f) \|$	1.036	3.138	1.037	1.562	2.862	0.986
$\alpha_k = 1$ accepté	9	14	8	9	14	13

TAB. 25 – *Comparaison de diverses mises à jour sans stratégie de scaling (1a)*

Algorithme	7	8	9	10	11
Critère J	3.812	3.652	4.055	9.154	3.901
Nb éval.	104	100	101	102	103
$\| \nabla J(x_f) \|$	2.083	1.824	2.240	3.217	2.402
$\alpha_k = 1$ accepté	11	14	14	14	13

TAB. 26 – *Comparaison de diverses mises à jour sans stratégie de scaling (1b)*

Nbre éval.	1	2	3	4	5	6
25	27.23 (4)	32.51 (4)	25.20 (4)	28.38 (4)	32.51 (4)	30.05 (4)
50	11.11 (9)	17.16 (9)	11.99 (9)	11.65 (9)	17.15 (9)	11.20 (9)
100	3.891 (18)	9.479 (18)	5.177 (18)	4.072 (18)	9.725 (18)	4.61 (19)

TAB. 27 – *Comparaison de diverses mises à jour sans stratégie de scaling (2a)*

Nbre éval.	7	8	9	10	11
25	25.94 (4)	27.54 (4)	27.75 (4)	31.89 (4)	27.17 (4)
50	11.17 (9)	10.54 (9)	11.01 (9)	17.18 (9)	11.11 (9)
100	4.116 (18)	3.652 (19)	4.054 (19)	9.154 (18)	4.099 (18)

TAB. 28 – *Comparaison de diverses mises à jour sans stratégie de scaling (2b)*

Algorithme	1	2	3	4	5	6
Critère J	2.569	2.609	4.623	2.704	2.858	2.859
Nb éval.	108	101	110	111	104	104
$\| \nabla J(x_f) \|$	1.879	3.565	1.17	2.023	2.072	2.072
$\alpha_k = 1$ accepté	14	14	5	2	10	10

TAB. 29 – *Comparaison de diverses mises à jour avec selective scaling (1a)*

Algorithme	7	8	9	10	11
Critère J	2.292	2.749	2.45	2.820	2.547
Nb éval.	105	110	104	101	107
$\| \nabla J(x_f) \|$	2.243	1.004	2.712	3.969	1.827
$\alpha_k = 1$ accepté	9	3	9	14	7

TAB. 30 – *Comparaison de diverses mises à jour avec selective scaling (1b)*

Nbre éval.	1	2	3	4	5	6
25	29.22 (4)	23.58 (4)	28.20 (4)	28.02 (4)	23.58 (4)	26.65
50	8.406 (9)	8.536 (9)	10.88 (9)	9.417 (9)	8.536 (9)	8.388
100	3.045 (17)	2.609 (19)	4.696 (17)	3.167 (17)	2.511 (18)	2.884

TAB. 31 – *Comparaison de diverses mises à jour avec selective scaling (2a)*

Nbre éval.	7	8	9	10	11
25	28.72 (4)	25.90 (4)	29.61 (4)	23.58 (4)	29.23 (4)
50	7.956 (9)	10.10 (9)	6.38 (9)	8.529 (9)	8.431 (9)
100	2.501 (18)	3.209 (19)	2.729 (18)	2.820 (19)	2.847 (17)

TAB. 32 – *Comparaison de diverses mises à jour avec selective scaling (2b)*

Algorithme	FR	PR stab.	BFGS	Liao
Critère J	38.60	7.195	3.757	46.87
Nb éval.	116	114	101	102
$\| \nabla J(x_f) \|$	5.380	2.149	1.035	3.811
$\alpha_k = 1$ accepté	0	4	14	13

TAB. 33 – *Algorithmes testés avec recherche d'Armijo (1)*

Algorithme	1	2	3	4	5
Critère J	2.578	3.069	6.5	2.947	3.26
Nb éval.	114	108	111	116	110
$\| \nabla J(x_f) \|$	1.772	2.565	1.589	1.602	2.679
$\alpha_k = 1$ accepté	0	9	4	0	7

TAB. 34 – *Algorithmes testés avec recherche d'Armijo (2)*

Algorithme	7	8	9	10	11
Critère J	2.716	2.607	2.451	2.424	2.729
Nb éval.	115	104	107	107	113
$\| \nabla J(x_f) \|$	2.156	1.772	1.258	1.258	2.777
$\alpha_k = 1$ accepté	0	6	4	10	1

TAB. 35 – *Algorithmes testés avec recherche d'Armijo (3)*

Algorithme	SD	FR	PR	PR stabilisé
Critère J	58.00	46.75	26.45	25.65
Nb éval.	122	129	116	114
$\| \nabla J(x_f) \|$	9.557	10.58	4.553	4.552

TAB. 36 – *Comparaison d'algorithmes de Gradient Conjugué (1)*

Nbre éval.	SD	FR	PR	PR stabilisé
25	103.7 (4)	78.27 (4)	86.68 (4)	86.68 (4)
50	91.79 (8)	73.60 (8)	62.50 (8)	68.0 (8)
100	64.84 (16)	54.51 (15)	32.15 (16)	29.20 (17)

TAB. 37 – *Comparaison d'algorithmes de Gradient Conjugué (2)*

Algorithme	1	2	3	4	5	6	7
Critère J	21.57	21.58	22.47	22.28	27.55	21.58	22.19
Nb éval.	115	116	115	100	121	115	130
$\| \nabla J(x_f) \|$	4.803	4.846	4.926	4.625	10.42	4.847	4.627
$\alpha_k = 1$ accepté	0	0	0	14	0	0	0

TAB. 38 – *Comparaison de diverses stratégies de scaling (1)*

Nbre éval.	1	2	3	4
25	78.06 (4)	77.94 (4)	82.25 (4)	72.17 (4)
50	47.89 (8)	47.7 (8)	48.36 (8)	46.48 (9)
100	25.69 (16)	25.78 (16)	26.51 (16)	22.28 (19)

TAB. 39 – *Comparaison de diverses stratégies de scaling (2)*

Nbre éval.	5	6	7
25	83.90 (4)	77.94 (4)	77.82 (4)
50	51.57 (9)	47.7 (8)	57.04 (7)
100	31.79 (16)	25.7 (16)	27.97 (15)

TAB. 40 – *Comparaison de diverses stratégies de scaling (3)*

Algorithme	1	2	3	4	5	6
Critère J	21.57	31.80	26.6	22.24	23.13	23.21
Nb éval.	115	103	117	116	110	113
$\| \nabla J(x_f) \|$	4.803	7.011	5.308	6.115	5.263	4.172
$\alpha_k = 1$ accepté	0	13	2	0	5	0

TAB. 41 – *Comparaison de divers algorithmes sans stratégie de scaling (1a)*

Algorithme	7	8	9	10	11
Critère J	21.78	21.78	22.04	31.74	21.58
Nb éval.	110	116	112	103	115
$\| \nabla J(x_f) \|$	4.652	5.346	4.194	6.995	4.848
$\alpha_k = 1$ accepté	0	0	1	12	0

TAB. 42 – *Comparaison de divers algorithmes sans stratégie de scaling (1b)*

Nbre éval.	1	2	3	4	5	6
25	78.06 (4)	84.44 (4)	103.72(4)	80.05 (4)	84.44 (4)	71.39 (4)
50	47.89 (8)	53.10 (9)	54.03 (8)	50.11 (8)	49.63 (9)	44.52 (9)
100	25.69 (16)	32.72 (18)	31.96 (16)	25.76 (16)	25.23 (17)	25.75 (17)

TAB. 43 – *Comparaison de divers algorithmes sans stratégie de scaling (2a)*

Nbre éval.	7	8	9	10	11
25	78.28 (4)	77.91 (4)	78.63 (4)	84.41 (4)	78.01 (4)
50	43.07 (9)	49.53 (8)	44.93 (9)	53.07 (9)	48.00 (8)
100	24.26 (18)	26.04 (16)	24.32 (17)	32.69 (17)	25.72 (17)

TAB. 44 – *Comparaison de divers algorithmes sans stratégie de scaling (2b)*

Algorithme	1	2	3	4	5	6
Critère J	22.28	25.597	57.36	25.19	25.47	24.63
Nb éval.	100	100	105	101	102	99
$\| \nabla J(x_f) \|$	4.625	9.008	6.131	3.475	6.662	4.958
$\alpha_k = 1$ accepté	14	14	7	14	13	14

TAB. 45 – *Comparaison de divers algorithmes avec stratégie de scaling (1a)*

Nbre éval.	1	2	3	4	5	6
25	72.17 (4)	78.86 (4)	103.7 (4)	73.83 (4)	78.86 (4)	72.27 (4)
50	46.48 (9)	59.64 (9)	86.41 (8)	48.54 (9)	58.18 (9)	47.80 (9)
100	22.28 (19)	25.6 (19)	57.4 (18)	25.19 (19)	25.47(18)	24.63 (19)

TAB. 46 – *Comparaison de divers algorithmes avec stratégie de scaling (1b)*

Algorithme	7	8	9	10	11
Critère J	24.07	22.87	24.32	25.56	22.24
Nb éval.	100	99	102	100	99
$\| \nabla J(x_f) \|$	3.907	6.714	3.502	8.947	4.529
$\alpha_k = 1$ accepté	14	14	14	14	14

TAB. 47 – *Comparaison de divers algorithmes avec stratégie de scaling (2a)*

Nbre éval.	7	8	9	10	11
25	73.78 (4)	71.85 (4)	74.12 (4)	78.83 (4)	72.62 (4)
50	47.95 (9)	47.42 (9)	47.34 (9)	59.56 (9)	46.0 (9)
100	24.07 (19)	22.87 (19)	24.32 (19)	25.56 (19)	22.24 (19)

TAB. 48 – *Comparaison de divers algorithmes avec stratégie de scaling (2b)*