L'EPSILON STEEPEST DESCENT ALGORITHME ASSOCIE A LA RECHERCHE LINEAIRE INEXACTE DE WOLFE

by

DEGAICHIA HAKIMA

Submitted to the de Mathématiques
in partial fulfillment of the requirements for the degree of

Doctorat

at the

UNIVERSITÉ BADJI MOKHTAR ANNABA

degreemonth undefined degreeyear undefined

Signature of Author .....
de Mathématiques
22 novembre 2014

Accepted by .....

Table des matières

.

Résumé

On considère le problème d'optimisation sans contraintes suivant :

$$(P) \quad \min \{f(x) : x \in R^n\}.$$

où $f : R^n \to R$ est continument différentiable.

On définit dans ce travail un nouveau algorithme qui accélère la convergence de la méthode du gradient. On étudie la convergence globale du nouveau algorithme qu'on a nommé Wolfe epsilon steepest descent algorithme, en utilisant la recherche linéaire inéxacte de Wolfe ([35],[36]). Dans [16] et [33], Benzine, Djeghaba et Rahali ont étudié le même problème en utilisant une recherche linéaire exacte ou une recherche linéaire inéxacte d'Armijo. On a aussi effectué 700 tests numériques et nous avons montré que le nouveau algorithme est plus pérformant que les deux autres déjà étudiés i.e. l'epsilon steepest algorithme avec des recherches linéaires exactes ou d'Armijo.

Abstract

We study the unconstrained minimization problem

$$(P) \quad \min \{f(x) : x \in R^n\}.$$

where $f : R^n \to R$ is a continously differentiable function. We introduce a new algorithm which accelerates the convergence of the steepest descent method. We study the global convergence of the new algorithm, named the Wolfe epsilon steepest descent algorithm, by using Wolfe inexact line search ([wolfe]). In [16], [33], Benzine, Djeghaba and Rahali studied the same problem by using exact and Armijo inexact line serachs. Numerical tests show that Wolfe Epsilon steepest descent Algorithm accelerates the convergence of the gradient method and is more successful than Armijo Epsilon Steepest descent, Exact Epsilon steepest descent algorithm and Steepest descent algorithm.

INTRODUCTION

Soit $f : R^n \to R$ et (P) le problème de minimisation non linéaire, sans contraintes, suivant :

$$\min \ \{f(x) : x \in R^n\} \qquad (P)$$

où $f : R^n \to R$ est continument différentiable. Notons

$$g_k = \nabla f(x_k)$$

Pour résoudre le problème (P), la majorité des méthodes génèrent une suite $\{x_k\}_{k\in\mathbb{N}}$ de la forme suivante :

$$x_{k+1} = x_k + \alpha_k d_k \tag{0.1}$$

où d_k est une direction de descente et α_k est le pas obtenu en effectuant une optimisation unidimensionnelle. Dans les méthodes du gradient conjugué les directions de descente sont de la forme :

$$d_k = -g_k + \beta_k d_{k-1} \tag{0.2}$$

où le scalaire β_k caractérise les différentes variantes du gradient conjugué. Si $\beta_k = 0$, alors on obtient la méthode du gradient. Un autre choix de directions est donné par

$$d_k = -B_k^{-1} g_k \tag{0.3}$$

où B_k est une matrice non singulière symetrique. Comme cas importants, citons :

$$\begin{aligned} B_k &= I \quad (M\acute{e}thode\ steepest\ descent\ ) \\ B_k &= \nabla^2 f(x_k)\ (Methode\ de\ Newton) \end{aligned}$$

Les méthodes quasi Newton sont aussi de la forme (0.3).

Toutes ces méthodes sont implémentées en prenant en considération que d_k est une direction de descente i.e.

$$d_k^T g_k < 0$$

Les propriètées de convergence des méthodes à directions de descente et recherches linéaires dépendent du bon choix de d_k et du pas α_k. L'angle que fait la direction d_k et la direction du gradient $-g_k$ est

fondamental. C'est pour celà qu'on définit

$$\cos(\theta_k) = \frac{-d_k^T g_k}{\|g_k\| \, \|d_k\|}$$

Nous choisirons α_k de sorte qu'on obtienne une décroissance suffisante de la fonction f, mais en même il faut que ce calcul ne soit pas coûteux en temps et en mémoire. Le choix optimal est obtenu en choisissant α comme solution optimale de la fonction d'une variable $\varphi(\alpha)$ definie par

$$\varphi(\alpha) = f(x_k + \alpha d_k)$$

Les recherches linéaires exactes consistent à calculer α_k comme solution du problème unidimensionnel suivant :

$$f(x_k + \alpha_k d_k) = \min \{f(x_k + \alpha d_k) : \alpha > 0\}$$

Malheureusement, les recherches linéaires exactes sont difficiles à réaliser pratiquement et sont coûteuses en temps et en mémoire. La stratégie que nous allons appliquer dans cette partie consiste à choisir α_k vérifiant les deux conditions suivantes :

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \sigma_1 \alpha_k g_k^T d_k \tag{0.4}$$

$$g(x_k + \alpha_k d_k)^T d_k \geq \sigma_2 g_k^T d_k \tag{0.5}$$

où $0 < \sigma_1 < \sigma_2 < 1$. La première relation (0.4) (*condition d'Armijo ([5]),* assure que la fonction décroit suffisamment. La seconde condition (0.5) prévient que le pas α_k devienne trés petit. Les deux conditions (0.4) et (0.5) s'appellent conditions de Wolfe.

On peut aussi choisir α_k vérifiant les conditions suivantes :

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \sigma \alpha_k g_k^T d_k \tag{0.6}$$

$$f(x_k + \alpha_k d_k) \geq f(x_k) + (1 - \sigma) \alpha_k g_k^T d_k \tag{0.7}$$

où $0 < \sigma < \frac{1}{2}$. (0.6) et (0.7) s'appellent conditions de Goldstein.

La méthode du gradient est l'une des plus simples et célèbres méthodes d'optimisation sans contraintes. Pour beaucoup de problèmes la méthode du gradient devient trés lente quans on s'approche d'un point stationnaire. IL existe beaucoup de méthodes qui y rémedient à ce problème. Au lieu de considerer $d_k = -\nabla f(x_k)$, on peut se déplacer le long de $d_k = -D_k \nabla f(x_k)$ ([8], [9], [10], [12], [13], [14], [15], [17], [21], [22], [23], [24], [25], [28], [31], [32], [33])), ou bien le long $d_k = -g_k + h_k$ ([18], [19], [20], [21], [27], [29], [30]),

où D_k est une matrice choisie convenablement et h_k est un vecteur approprié.

Dans [16] et dans [33], Benzine, Djeghaba et Rahali ont essayé de résoudre ce problème par une autre méthode, en accélérant la convergence de la méthode du gradient.

Pour arriver à ce but, ils ont élaboré un nouveau Algorithme qu'ils ont nommé l'epsilon steepest descent algorithm, dans lequel la formule de Florent Cordellier et celle de Wynn ([11], [37], [38]) jouent un role essentiel. Ils ont aussi prouvé la convergence globale en utilisant des recherches linéaires exactes et d'Armijo.

Dans ce travail on accelère la convergence de la méthode du gradient et on étudie la convergence globale en utilisant l'Epsilon Algorithme et les recherches linéaires inexactes de wolfe vérifiant (0.4) et (0.5). Nous avons appelé le nouveau algorithme : Wolfe epsilon steepest descent algorithm.

Avec 700 tests numériques nous avons montré que le nouveau algorithme est plus pérformant que les deux autres déjà étudiés i.e. l'epsilon steepest algorithme avec des recherches linéaires exactes ou d'Armijo ([16], [33]).

La thèse est divisée en cinq chapitres. Les algorithmes nouveaux et les résultats originaux de convergence se trouvent dans le chapitre 5.

Le problème traité dans cette thèse est l'accéleration de la convergence. Il est alors naturel de consacrer un chapitre entier à ce sujet. Ce sera le but du chapitre1. On commence par donner les motivations qui ont permis l'élaboration des méthodes d'accélération de la convergence. On présente ensuite des exemples de ces méthodes. On a deux classes de méthodes d'accélération de la convergence : linéaire, comme les procédés de sommation linéaire, et non linéaire, comme le procédé d'extrapolation de Richardson, le procédé Δ^2D'aitken, l'$\varepsilon-$algorithme, le $\rho-$algorithme et le $\theta-$algorithme. Pour qu'il y ait convergence des procédés de sommation linéaire, des conditions suffisantes sont données dans le théorème de Toepliz. On donnera également des résultats de convergence pour le procédé Δ^2D'aitken ainsi que pour l'$\varepsilon-$algorithme scalaire et l'$\varepsilon-$algorithme vectoriel normé. Ces deux derniers algorithmes nous permettront par la suite, la construction du nouvel algorithme proposé dans ce travail. On donnera des exemples dus à Brezinski de suites, dont la convergence est accélérée par l'$\varepsilon-$algorithme scalaire et par ses deux généralisations.

On donne dans le chapitre2 les définitions et les propriétés générales des problèmes d'optimisation sans contraintes. On définira la notion de direction de descente, notion trés importante qui nous servira par la suite à construire les algorithmes d'optimisation sans contraintes. Les conditions necessaires et ou suffisantes seront aussi détaillées dans ce chapitre.

La majorité des algorithmes d'optimisation non linéaire procèdent comme suit. Etant donné un point x_k, on cherche une certaine direction $d_k \in R^n$ et un pas $\lambda_k \in R$. Ceci nous permet de trouver le successeur x_{k+1} de x_k par la formule : $x_{k+1} = x_k + \lambda_k d_k$ et le procesuss est ainsi repeté. Le pas λ_k est une solution

optimale d'un problème de minimisation unidimentionnelle, c'est à dire la minimisation d'une fonction d'une variable réelle. Donc pour les problèmes de minimisation sans contraintes, on résout à chaque itération des sous problèmes de minimisation de fonctions d'une variable réelle. C'est pour celà qu'on consacre le chapitre3 à l'étude des minimums des fonctions d'une variable réelle ou recherche linéaire. On donne d'abord le principe général des algorithmes qui traitent ce problème. Ensuite on expose en détail les algorithmes et les propriétés de deux grandes classes de recherches linéaires. Il s'agit des recherches linéaires exactes et les recherches linéaires inexactes. On s'interesse specialement aux recherches linéaires d'Armijoo, de Goldstein et de Wolfe. On expose aussi le théorème de Zoutendijk.

Le chapitre 4 est consacré à la méthode du gradient ou steepest descent. On explique l'origine d'une telle appelation et on montre que son défaut principal est sa convergence trés lente au voisinage des points stationnaires. Des tests numérique effectués sur la fonction test de Rosembrok viennent confirmer ces faits.

Le chapitre 5 contient le résultat original de cette thèse. On accelère la convergence de la méthode du gradient et on étudie la convergence globale en utilisant l'Epsilon Algorithme et les recherches linéaires inexactes de wolfe vérifiant (0.4) et (0.5). Nous avons appelé le nouveau algorithme : Wolfe epsilon steepest descent algorithm.

Avec 700 tests numériques nous avons montré que le nouveau algorithme est plus pérformant que les deux autres déjà étudiés i.e. l'epsilon steepest algorithme avec des recherches linéaires exactes ou d'Armijo ([16], [33]).

Chapitre 1

METHODES D'ACCELERATION DE LA CONVERGENCE EN ANALYSE NUMERIQUE

1.1 Introduction

Considérons la suite $(S_n) = \{S_0, S_1, ..., S_n, ...\}$ qui converge vers S.

Les questions essentielles qui se posent sont les suivantes :

1- Comment estimer quantitativement la vitesse de convergencede cette suite ?.

2- Si cette suite converge lentement, est-il possible d'augmenter sa vitesse de convergence en la remplaçant par une autre suite (V_n) qui tendrait plus rapidement vers la même limite S .

Une réponse à la deuxième question est donnée par les méthodes d'accélération de la convergence. Cependant, il n'existe pas de méthode d'accélération de la convergence valable pour tous les cas (c'est selon la nature de la suite à accélérer).

les méthodes d'accélération de la convergence consistent toutes à transformer une suite (S_n) en une suite (V_n) de même nature, il faudra alors que la suite (V_n) converge vers la solution S du problème posé.

Dans le cas d'une suite réelle, on dit que (V_n) converge plus vite que (S_n) si :

$$\lim_{n\to\infty} \frac{V_n - S}{S_n - S} = 0$$

Si cette condition est vérifiée, on dit que l'on a accéléré la convergence de la suite (S_n).

Les possibilités d'accélération de la convergence d'une suite (S_n) dépendront de la vitesse avec laquelle cette suite converge.

Il existe plusieurs méthodes d'accélération de la convergence, nous donnons dans ce chapitre un aperçu sur quelques unes d'entre elles.

1.2 Les procédés de sommation linéaire

Les procédés de sommation linéaire ont été conçu par Hardy pour donner une limite aux séries divergentes par une sorte de prolongement analytique. Ces procédés peuvent aussi être appliqués aux séries lentement convergentes dans le but de les accélérer.

La classe la plus importante des procédés de sommation linéaire est basée sur l'application d'une matrice triangulaire $A = (a_{ij})$ qui transforme la suite originale (S_n) en une suite (V_n) définie par :

$$\begin{pmatrix} V_0 \\ V_1 \\ . \\ . \end{pmatrix} = A + \begin{pmatrix} S_0 \\ S_1 \\ . \\ . \end{pmatrix} \qquad \text{avec} \quad A = \begin{pmatrix} a_{00} & 0 & 0 & 0 & 0 \\ a_{10} & a_{11} & 0 & 0 & 0 \\ a_{20} & a_{21} & a_{22} & 0 & 0 \\ . & . & . & . & 0 \\ . & . & . & . & . \end{pmatrix} \tag{1.1}$$

On note par A ce procédé de sommation linéaire.

Soit S la limite de la suite (S_n), le procédé de sommation linéaire est dit régulier si la suite transformée (V_n) converge vers la même limite S.

Pour obtenir la régularité d'un procédé de sommation linéaire, Toepliz a donné dans le théorème suivant trois conditions suffisantes :

Théorème de Toepliz

Le procédé de sommation linéaire A est régulier si :

(i) $\sum_{k=0}^{\infty} |a_{nk}| < M, \forall n$

(ii) $\lim_{n\to\infty} a_{nk} = 0, \forall k$

(iii) $\lim_{n\to\infty} \sum_{k=0}^{\infty} a_{nk} = 1$

1.2.1 Quelques exemples de procédés réguliers

Le procédé de Hölder noté (H,1).

La matrice $A = (H, 1)$ est donnée par :

$$a_{ij} = \left\{ \begin{array}{c} \frac{1}{i+1} \text{ pour } j = 1, ...i \\ 0 \text{ pour } j > i \end{array} \right\}$$

Elle s'écrit alors :

$$A = \begin{pmatrix} \frac{1}{1} & 0 & 0 & 0 & . \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & . \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & . \\ . & . & . & . & 0 \\ . & . & . & . & . \end{pmatrix}$$

et on obtient la suite transformée suivante :

$$V_0 = S_0, V_1 = \frac{S_0 + S_1}{2}, V_2 = \frac{S_0 + S_1 + S_2}{3}, ...$$

Si avec la suite (V_n), la convergence n'est pas accélérée on généralise ce procédé en considérant la matrice A^K(K entier), on le note alors (H, K).

La matrice $(H, 2)$ est définie par : $(H, 2) = (H, 1) \cdot (H, 1)$, on obtient une nouvelle suite :

$$W_0 = V_0, W_1 = \frac{V_0 + V_1}{2}, W_2 = \frac{V_0 + V_1 + V_2}{3}, ...$$

On continue de cette manière jusqu'à obtention d'accélération de la convergence de la suite (S_n).

Le procédé de Cesaro d'ordre K noté (C, K)

La matrice d'ordre K est donnée par : $(C, K) = (H, 1) \cdot L^{K-1}$ où L est une matrice triangulaire inférieure gauche, avec tous les éléments égaux à 1.

Le procédé d'Euler

La matrice A de ce procédé est donnée par :

$$a_{ij} = \left\{ \begin{array}{c} \frac{q^{i-j}}{(q+1)^i} C_j^i \text{ pour } j = 0, 1, ...i \\ 0 \text{ pour } j > i \end{array} \right\} \qquad avec \sum_{j=0}^{\infty} a_{ij} = 1$$

où le paramètre q est choisi arbitrairement ($q = 1$).

Remarques 1.1

1- Les trois procédés ci-dessus sont réguliers.

2- Ces procédés vérifient : $\sum_{k=0}^{n} a_{nk} = 1, \forall n$ cette égalité garantit la troisième condition du théorème de Toepliz.

1.3 Le procédé d'extrapolation de Richardson

Le procédé d'extrapolation de Richardson n'emploie pas de matrice pour obtenir une nouvelle suite, mais fait intervenir une suite auxiliaire. Ces relations s'écrivent pour une suite (S_n) de la façon suivante :

$$\begin{aligned} V_{k+1}^{(n)} &= \frac{x_n V_k^{(n+1)} - x_{n+k+1} V_k^{(n)}}{x_n - x_{n+k+1}} \; n, k = 0, 1.... \\ V_0^{(n)} &= S_n \text{ pour } n = 0, 1, ... \end{aligned} \tag{1.2}$$

On place ces quantités dans un tableau à double entrée : l'indice inférieur représente une colonne et l'indice supérieur représente une diagonale descendante :

$$\begin{array}{llllllll} V_0^{(0)} = S_0 & & & & & & & \\ & V_1^{(0)} & & & & & & \\ V_0^{(1)} = S_1 & & V_2^{(0)} & & & & & \\ & V_1^{(1)} & & V_3^{(0)} & & & & \\ V_0^{(2)} = S_2 & & V_2^{(1)} & & V_4^{(0)} & & & \\ & V_1^{(2)} & & V_3^{(1)} & & . & & \\ V_0^{(3)} = S_3 & & V_2^{(2)} & & V_4^{(1)} & & . & \\ & V_1^{(3)} & & V_3^{(2)} & & . & & \\ V_0^{(4)} = S_4 & . & V_2^{(3)} & . & V_4^{(2)} & & . & \\ . & . & . & . & . & . & & \\ . & . & . & . & . & . & . & \end{array} \tag{1.3}$$

Si l'on connaît $V_k^{(n)}$ et $V_k^{(n+1)}$ on peut calculer $V_{k+1}^{(n)}$. Ceci est représenté par :

$$\begin{array}{ccc} V_k^{(n)} & \searrow & \\ & & V_{k+1}^{(n)} \\ V_k^{(n+1)} & \nearrow & \end{array} \tag{1.4}$$

On dit dans ce cas que le procédé d'extrapolation de Richardson est un algorithme de triangle, il transforme la suite (S_n) en un ensemble de suites $\left(V_k^{(n)}\right)$.

La convergence de cet algorithme est obtenue sous des conditions trés restrictives sur les suites $\left(V_k^{(n)}\right)$ et $\left(V_{k+1}^{(n)}\right)$ et sur la suite auxiliaire (x_n), elles ne sont vérifiées que dans des cas particuliers. les théorèmes suivants le montrent :

Théorème 1.1 [2]

Soit (x_n) une suite strictement décroissante de nombres réels positifs et tendant vers zéro losque n tend vers l'infini.

Une condition nécessaire et suffisante pour que :

$$\lim_{k\to\infty} V_k^{(n)} = \lim_{k\to\infty} S_k$$

pour tout $n = 0, 1, ...$ et pour toute suite convergente (S_k) est qu'il existe $a > 1$ tel que :

$$\frac{x_n}{x_{n+1}} \geq a > 1, n = 0, 1, ...$$

Cette condition est aussi une condition nécessaire et suffisante pour que :

$$\lim_{n\to\infty} V_k^{(n)} = \lim_{n\to\infty} S_n$$

pour tout $k = 0, 1, ...$et pour toute suite convergente (S_n).

Remarque 1.2

La condition du théorème ci-dessus entraine la convergence des colonnes et des diagonales du tableau représentant le procédé d'extrapolation de Richardson.

Théorème 1.2 [2]

Supposons que la condition du théorème précédent soit satisfaite ; alors, une condition nécessaire et suffisante pour que la suite $\left(V_{k+1}^{(n)}\right)$ (k fixé) converge plus vite que la suite $\left(V_k^{(n)}\right)$ (k fixé) est que :

$$\lim_{n\to\infty}\frac{V_k^{(n+1)}-S}{V_k^{(n)}-S}=\lim_{n\to\infty}\frac{x_{n+k+1}}{x_n}$$

1.4 Le procédé Δ^2 D'aitken

Le procédé Δ^2 D'aitken consiste à transformer la suite (S_n) en une suite notée $(V(S_n))$définie par

$$V(S_n)=\frac{S_nS_{n+2}-S_{n+1}^2}{S_{n+2}-2S_{n+1}+S_n};n=0,1,....$$

Tous les termes de la suite sont définis si :

$$S_{n+2}-2S_{n+1}+S_n\neq 0;\forall n.$$

$V(S_n)$ peut encore s'écrire de la façon suivante :

$$V(S_n)=S_{n+1}-\frac{\Delta S_{n+1}}{\frac{\Delta S_{n+1}}{\Delta S_n}-1}$$

où la notation ΔS_m est définie par : $\Delta S_m=S_{m+1}-S_m$ pour tout $m\geq 0$.

Un résultat de convergence de ce procédé est donné par théorème suivant :

Théorème 1.3 [2]

Si on applique le procédé Δ^2 D'aitken à une suite (S_n) qui converge vers S et si :

$$\lim_{n\to\infty}\frac{S_{n+1}-S}{S_n-S}=\frac{\Delta S_{n+1}}{\Delta S_n}=a\neq 1$$

alors la suite $(V(S_n))$ converge vers S et cela plus vite que la suite (S_{n+1}).

1.5 L'ε-algorithme

1.5.1 L'ε-algorithme scalaire

L'ε-algorithme est dû à P. Wynn [5]. Il repose sur de solides bases théoriques et ses applications sont extrêmement importantes.

Dans L'ε-algorithme scalaire, on calcule également des quantités avec deux indices $\varepsilon_k^{(n)}$. On utilise

pour cela les relations

$$\begin{aligned} \varepsilon_{-1}^{(n)} &= 0 \qquad \varepsilon_0^{(n)} = S_n \qquad && n = 0, 1, ... \\ \varepsilon_{k+1}^{(n)} &= \varepsilon_{k-1}^{(n+1)} + \frac{1}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}} && n, k = 0, 1, ... \end{aligned} \tag{1.5}$$

On place ces quantités dans un tableau à double entrée : le tableau ε. L'indice inférieur reste constant dans une colonne tandis que l'indice superieur reste constant dans une diagonale descente.

$$\begin{array}{llllllll} \varepsilon_{-1}^{(0)} = 0 & & & & & & & \\ & \varepsilon_0^{(0)} = S_0 & & & & & & \\ \varepsilon_{-1}^{(1)} = 0 & & \varepsilon_1^{(0)} & & & & & \\ & \varepsilon_0^{(1)} = S_1 & & \varepsilon_2^{(0)} & & & & \\ \varepsilon_{-1}^{(2)} = 0 & & \varepsilon_1^{(1)} & & \varepsilon_3^{(0)} & & & \\ & \varepsilon_0^{(2)} = S_2 & & \varepsilon_2^{(1)} & & \varepsilon_4^{(0)} & & \\ \varepsilon_{-1}^{(3)} = 0 & & \varepsilon_1^{(2)} & & \varepsilon_3^{(1)} & & . & \\ & \varepsilon_0^{(3)} = S_3 & & \varepsilon_2^{(2)} & & \varepsilon_4^{(1)} & & . \\ \varepsilon_{-1}^{(4)} = 0 & & \varepsilon_1^{(3)} & & \varepsilon_3^{(2)} & & . & \\ . & \varepsilon_0^{(4)} = S_4 & . & \varepsilon_2^{(3)} & . & \varepsilon_4^{(2)} & & . \\ . & . & . & . & . & . & . & \\ . & . & . & . & . & . & . & . \end{array} \tag{1.6}$$

La relation (1.5) relie des quantités situées aux quatre sommets d'un losange :

$$\begin{array}{ccc} & \varepsilon_k^{(n)} & \\ \nearrow & & \searrow \\ \varepsilon_{k-1}^{(n+1)} & & \varepsilon_{k+1}^{(n)} \\ \searrow & & \nearrow \\ & \varepsilon_k^{(n+1)} & \end{array} \tag{1.7}$$

Si l'on connaît $\varepsilon_{k-1}^{(n+1)}$, $\varepsilon_k^{(n)}$ et $\varepsilon_k^{(n+1)}$, la relation (1.5) permet de calculer $\varepsilon_{k+1}^{(n)}$; c'est ce que signifient les flèches du tableau (4). On dit que l'ε-algorithme est un algorithme de losange.

La relation (1.5) permet donc de progresser de la gauche vers la droite et de haut en bas dans le tableau ε, à partir des conditions initiales $\varepsilon_{-1}^{(n)} = 0$ et $\varepsilon_0^{(n)} = S_n$ pour $n = 0, 1, ...;$. Si l'on connaît S_0 et S_1 on peut calculer $\varepsilon_1^{(0)}$; si l'on connaît S_1 et S_2 on peut calculer $\varepsilon_1^{(1)}$; à partir de $\varepsilon_1^{(0)}$ et de $\varepsilon_1^{(1)}$ on obtiendra ensuite $\varepsilon_2^{(0)}$ et ainsi de suite.

La relation (1.5) peut également s'écrire :

$$\varepsilon_2^{(n)} = \frac{S_n S_{n+2} - S_{n+1}^2}{S_{n+2} - 2S_{n+1} + S_n}; n = 0, 1, ..$$

Ceci est le procédé $\triangle^2$ d'Aitken. L'ε-algorithme scalaire est donc une généralisation du procédé $\triangle^2$ d'Aitken.

Généralisations

Première généralisation : Cette généralisation se fait en introduisant une suite auxiliaire (x_n) dans la relation (1.5), on obtient alors les régles suivantes :

$$\begin{aligned} \varepsilon_{-1}^{(n)} &= 0 \qquad \varepsilon_0^{(n)} = S_n \qquad n = 0, 1, ... \\ \varepsilon_{k+1}^{(n)} &= \varepsilon_{k-1}^{(n+1)} + \frac{\Delta x_n}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}} \qquad n, k = 0, 1, ... \end{aligned} \tag{1.8}$$

avec $\Delta x_n = x_{n+1} - x_n$ pour tout n.

Deuxiéme généralisation : Cette généralisation se fait aussi en introduisant une suite auxiliaire (x_n) dans la relation (1.5), on obtient alors les régles suivantes :

$$\begin{aligned} \varepsilon_{-1}^{(n)} &= 0 \qquad \varepsilon_0^{(n)} = S_n \qquad n = 0, 1, ... \\ \varepsilon_{k+1}^{(n)} &= \varepsilon_{k-1}^{(n+1)} + \frac{\Delta x_{n+k}}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}} \qquad n, k = 0, 1, ... \end{aligned} \tag{1.9}$$

avec $\Delta x_{n+k} = x_{n+k+1} - x_{n+k}$ pour tout n.

1.5.2 L'ε-algorithme vectoriel et l'ε-algorithme vectoriel normé

La forme vectorielle de l'ε-algorithme a également été étudiée par Wynn. Soit $\{S_n\}$ une suite de vecteurs de C^p. Les règles de l'ε-algorithme vectoriel sont les suivantes :

$$\begin{aligned} \varepsilon_{-1}^{(n)} &= 0 \in C^p \qquad \varepsilon_0^{(n)} = S_n \in C^p \qquad n = 0, 1, ... \\ \varepsilon_{k+1}^{(n)} &= \varepsilon_{k-1}^{(n+1)} + \left[\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}\right]^{-1} \qquad n, k = 0, 1, ... \end{aligned} \tag{1.10}$$

On voit que, pour pouvoir appliquer cet algorithme, il faut définir ce qu'on appelle l'inverse d'un vecteur. Etand donné $y \in C^p$, on définit son inverse $y^{-1} \in C^p$ par la relation :

$$y^{-1} = \frac{\bar{y}}{(y,y)}, \tag{1.11}$$

$\bar{y}$ est le vecteur dont les composantes sont les nombres complexes conjugués des composantes du vecteur y, (y,y) désigne le produit scalaire dans C^p, du vecteur par lui même. Si l'on désigne par $y_i (i = 1, ..., p)$, les composantes du vecteur y, alors

$$(y,y) = \sum_{i=1}^{p} y_i \bar{y}_i = \sum_{i=1}^{p} |y_i|^2 . \tag{1.12}$$

Les régles de l'ε-algorithme vectoriel normé sont les mêmes que celles de l'ε-algorithme vectoriel sauf pour l'inverse d'un vecteur.

L'inverse y^{-1} d'un vecteur y est défini par :

$$y^{-1} = \frac{\bar{y}}{\|y\|^2}$$

où $\|y\|$ désigne une norme vectorielle de y.

1.6 L'ρ-algorithme

Les régles de cet algorithme font intervenir une suite auxiliaire (x_n) de la manière suivante :

$$\begin{aligned} \rho_{-1}^{(n)} &= 0 \qquad \rho_0^{(n)} = S_n \qquad && n = 0, 1, ... \\ \rho_{k+1}^{(n)} &= \rho_{k-1}^{(n+1)} + \frac{x_{n+k+1} - x_n}{\rho_k^{(n+1)} - \rho_k^{(n)}} && n, k = 0, 1, ... \end{aligned} \tag{1.13}$$

Le calcul de $\rho_{2k}^{(n)}$ fait intervenir $S_n, S_{n+1}, ..., S_{n+2k}$ et $x_n, x_{n+1}, ..., x_{n+2k}$, on a alors le tableau suivant :

$$
\begin{array}{lllllll}
\theta_{-1}^{(0)}=0 & & & & & & \\
 & \theta_0^{(0)}=S_0 & & & & & \\
\theta_{-1}^{(1)}=0 & & \theta_1^{(0)} & & & & \\
 & \theta_0^{(1)}=S_1 & & \theta_2^{(0)} & & & \\
\theta_{-1}^{(2)}=0 & & \theta_1^{(1)} & & \theta_3^{(0)} & & \\
 & \theta_0^{(2)}=S_2 & & \theta_2^{(1)} & & \theta_4^{(0)} & \\
\theta_{-1}^{(3)}=0 & & \theta_1^{(2)} & & \theta_3^{(1)} & & . \\
 & \theta_0^{(3)}=S_3 & & \theta_2^{(2)} & & \theta_4^{(1)} & . \\
\theta_{-1}^{(4)}=0 & & \theta_1^{(3)} & & \theta_3^{(2)} & & . \\
. & \theta_0^{(4)}=S_4 & . & \theta_2^{(3)} & . & \theta_4^{(2)} & . \\
. & . & . & . & . & . & . \\
. & . & . & . & . & . & . \quad .
\end{array}
$$

1.7 Le θ-algorithme

Les régles de cet algorithme sont les suivantes :

$$\theta_{-1}^{(n)}=0 \text{ et } \theta_0^{(n)}=S_n \text{ pour } n=0,1,.... \tag{1.14}$$

$$\theta_{2k+1}^{(n)}=\theta_{2k-1}^{(n+1)}+D_{2k}^{(n)} \text{ pour } n,k=0,1,....$$

$$\theta_{2k+2}^{(n)}=\frac{D_{2k+1}^{(n+1)}\theta_{2k}^{(n+1)}-D_{2k+1}^{(n)}\theta_{2k}^{(n+2)}}{D_{2k+1}^{(n+1)}-D_{2k+1}^{(n)}} \text{ avec } D_k^{(n)}=\frac{1}{(\theta_k^{(n+1)}-\theta_k^{(n)}} \text{ pour } n,k=0,1,....$$

Les θ d'indice inférieur impair sont des intermédiaires de calcul, alors que les θ d'indice pair permettent d'étudier théoriquement la convergence de l'algorithme.

Le θ algorithme n'est pas un algorithme de losange par exemple pour le calcul de $\theta_2^{(n)}$, première étape de l'algorithme , la connaissance de quatre termes consécutifs est nécessaire : $S_n, S_{n+1}, S_{n+2}, S_{n+3}$, alors que les algorithmes ε et ρ n'avaient besoin que de trois termes successifs (algorithmes de losange). L'absence de forme déterminante pour $\theta_{2k}^{(n)}$ rend l'étude théorique de cet algorithme difficile sauf dans sa première étape.

1.8 Convergence de l'ε-algorithme scalaire

Avant d'établir la convergence de cet algorithme donnons le théorème suivant :

Théorème 1.4 [2]

Si on applique le procédé Δ^2d'Aitken à une suite (S_n) qui converge vers S et si :

$$\begin{aligned} \lim_{n\to\infty} \frac{S_{n+1}-S}{S_n-S} &= \lim_{n\to\infty} \frac{\Delta S_{n+1}}{\Delta S_n} = a \neq 1 \\ \text{où} \qquad \Delta S_{n+1} &= S_{n+2}-S_{n+1} \text{ et } \Delta S_n = S_{n+1}-S_n \end{aligned}$$

alors la suite

$$\varepsilon_2^{(n)} = \frac{S_n S_{n+2} - S_{n+1}^2}{S_{n+2} - 2S_{n+1} + S_n}; n = 0, 1, ...$$

converge vers S et cela plus vite que la suite (S_{n+1}).

Définition 1.1

On dit que la suite (S_n) est totalement monotone si :

$$\begin{aligned} (-1)^k \Delta^k S_n &\geq 0; n, k = 0, 1, ... \\ \text{où } \Delta^0 S_n &= S_n \\ \text{et pour tout } n,\ \Delta^{k+1} S_n &= \Delta^k S_{n+1} - \Delta^k S_n; n, k = 0, 1, ... \end{aligned}$$

Théorème 1.5 [2]

Si on applique l'ε-algorithme à une suite (S_n) qui converge vers S et s'il existe deux constantes $a \neq 0$ et b telles que la suite $(aS_n + b)$ soit totalement monotone, alors :

$$\begin{aligned} \lim_{n\to\infty} \varepsilon_{2k}^{(n)} &= S; k = 0, 1, ...\text{fixé} \\ \lim_{k\to\infty} \varepsilon_{2k}^{(n)} &= S; n = 0, 1, ...\text{fixé} \end{aligned}$$

Définition 1.2

On dit que la suite (S_n) est totalement oscillante si la suite $((-1)^n S_n)$ est totalement monotone.

Théorème 1.6 [2]

Si on applique l'ε-algorithme à une suite (S_n) qui converge vers S et s'il existe deux constantes $a \neq 0$ et b telles que la suite $(aS_n + b)$ soit totalement oscillante, alors :

$$\begin{aligned}\lim_{n\to\infty}\varepsilon_{2k}^{(n)} &= S; k = 0, 1, ...\text{fixé} \\ \lim_{k\to\infty}\varepsilon_{2k}^{(n)} &= S; n = 0, 1, ...\text{fixé}\end{aligned}$$

Remarque 1.3

Les théorèmes 1.5 et 1.6 n'accélérent pas la convergence, mais montrent que l'ε-algorithme converge. De plus on remarque que :

$$\begin{aligned}\lim_{n\to\infty}\frac{\varepsilon_{2k}^{(n)} - S}{S_{n+2k} - S} &= 0; k = 1, 2, ... \\ &\text{et} \\ \lim_{k\to\infty}\frac{\varepsilon_{2k}^{(n)} - S}{S_{n+2k} - S} &= 0; n = 1, 2, ...\end{aligned}$$

Ces résultats montrent que la condition du théorème 1.4 est satisfaite donc la suite $\varepsilon_{2k}^{(n)}$ converge plus vite que la suite (totalement monotone ou totalement oscillante) (S_{n+1}).

1.9 Convergence de l'ε-algorithme vectoriel normé (vectoriel).

L'ε-algorithme vectoriel est un cas particulier de l'ε-algorithme vectoriel normé, les théorèmes de convergence ci-dessous sont valables pour les deux algorithmes.

Hypothèses I :

1- (S_n) est une suite de vecteur de C^p

2- (a_n) est une suite de réels tels que $a_n \neq 0$ pour tout n et que (Δa_n) converge vers 0. On suppose de plus qu'il existe deux constantes α et β telles que :

$$\alpha < 1 < \beta \text{ et } a_n \notin [\alpha, \beta], \forall n$$

3- La suite (S_n) vérifie :

$$S_{n+1} - S = a_n (S_n - S); n = 1, 0, ...$$

avec $S \in C^p$.

Théorème 1.7 [2]

Si on applique l'ε-algorithme vectoriel normé (vectoriel) à une suite (S_n) qui vérifie les hypothèses

I précédentes et qui converge vers S alors :

$$\lim_{n\to\infty} \varepsilon_2^{(n)} = S$$

Si de plus la suite (a_n) admet une limite, alors :

$$\lim_{n\to\infty} \frac{\left\|\varepsilon_2^{(n)} - S\right\|}{\|S_{n+1} - S\|} = 0$$

Hypothèses II :

1- (S_n) est une suite de vecteur de C^p.

2- (e_n) est une suite de vecteur de C^p qui converge vers zéro lorsque n tend vers l'infini.

3- $y \in C^p$ et $y \neq 0$

4- a est un nombre réel tel que $0 < a < 1$

5- La suite (S_n) vérifie :

$$S_n - S = a^n (y + e_n) ; n = 1, 0, ...$$

avec $S \in C^p$.

Théorème 1.8 [2]

Si on applique l'ε-algorithme vectoriel normé (vectoriel) à une suite (S_n) qui vérifie les hypothèses II précédentes alors :

$$\lim_{n\to\infty} \varepsilon_2^{(n)} = S$$

et :

$$\lim_{n\to\infty} \frac{\left\|\varepsilon_2^{(n)} - S\right\|}{\|S_{n+k} - S\|} = 0; \forall k \geq 0 \text{ fixé}$$

Les théorèmes I.6 et I.7 montrent sous certaives conditions que l'ε-algorithme vectoriel normé (vectoriel) converge vers la limite de la suite originale et que cette convergence est accélérée..

1.10 Exemples de suites accélérées par l'ε-algorithme scalaire et ses deux généralisations

Exemple 1 :

Cet exemple est dû à Brezinski.

Soit (S_n) la suite définie par :

$$S_n = 1 + 3.e^{-1.4x_n} \text{ avec } x_n = 1.1^{n-1}$$

Cette suite converge lentement vers 1.

On a :

$S_0 = 1.73979....; S_1 = 1.64314....; .....; S_4 = 1.38631....$

En appliquant l'ε-algorithme scalaire, on obtient :

$\varepsilon_4^{(0)} = 1.73799....$

et en appliquant la première généralisation de l'ε-algorithme scalaire on obtient :

$\varepsilon_4^{(0)} = 1.00272....$

La deuxième généralisation de l'ε-algorithme scalaire donne :

$\varepsilon_4^{(0)} = 1.00224....$

Exemple 2 :

Cet exemple est également dû à Brezinski.

Soit (S_n) la suite définie par :

$$S_n = n \sin \frac{1}{n} \text{ avec } x_n = \log(n+1)$$

Cette suite converge vers 1.

On obtient : $S_{37} = 0.99987....$

et $\varepsilon_{36}^{(0)} = 0.999938....$en appliquant l'$\varepsilon$-algorithme scalaire.

La première généralisation de l'ε-algorithme scalaire donne $\varepsilon_{36}^{(0)} = 1.0000000027$, la deuxième généralisation de l'ε-algorithme scalaire donne $\varepsilon_{36}^{(0)} = 0.99999999976$..

Chapitre 2

OPTIMISATION SANS CONTRAINTES

2.1 Définitions

Définition 2.1 *Soit* $f : R^n \to R$, *on appelle problème de minimisation sans contraites le problème* (P) *suivant :*

$$(\ P\ ) \qquad \min\{f(x) : x \in R^n\}$$

1) $\hat{x} \in R^n$ *est un minimum global de* (P) *si et seulement si*

$$f(\hat{x}) \leq f(x) : \quad \forall x \in R^n$$

2) $\hat{x} \in R^n$ *est un minimum local de* (P) *si et seulement si il existe un voisinage*$V_\varepsilon(\hat{x})$ *tel que*

$$f(\hat{x}) \leq f(x) : \quad \forall x \in V_\varepsilon(\hat{x})$$

3) $\hat{x} \in R^n$ *est un minimum local strict de* (P) *si et seulement si il existe un voisinage*$V_\varepsilon(\hat{x})$ *tel que*

$$f(\hat{x}) < f(x) : \quad \forall x \in V_\varepsilon(\hat{x}),\ x \neq \hat{x}.$$

2.2 Direction de descente

Définition 2.2 *Soit* $f : R^n \to R$, $\hat{x} \in R^n$, $d \in R^n$ *est dite direction de descente au point* $\hat{x}$ *si et seulement si il existe* $\delta > 0$ *tel que*

$$f(\hat{x} + \lambda d) < f(\hat{x}) : \quad \forall \lambda \in]0, \delta[$$

Donnons maintenant une condition suffisante pour que d soit une direction de descente.

Théorème 2.1 *Soit* $f : R^n \to R$ *differentiable au point* $\hat{x} \in R^n$ *et* $d \in R^n$ *une direction vérifiant la condition suivante :*

$$f'(\hat{x}, d) = \nabla f(\hat{x})^t . d < 0.$$

Alors d *est une direction de descente au point* $\hat{x}$.

Preuve :

f est différentiable au point $\hat{x}$. Donc

$$f(\hat{x} + \lambda d) = f(\hat{x}) + \lambda \nabla f(\hat{x})^t . d + \lambda \left\| d \right\| \alpha(\hat{x}, \lambda d),$$

avec

$$\alpha(\hat{x}, \lambda d) \underset{\lambda \to 0}{\to} 0,$$

ceci implique que

$$f'(\hat{x}, d) = \lim_{\lambda \to 0} \frac{f(\hat{x} + \lambda d) - f(\hat{x})}{\lambda} = \nabla f(\hat{x})^t . d < 0.$$

La limite étant strictement négative, alors il existe un voisinage de zéro $V(0) =]-\delta, +\delta[$ tel que

$$\frac{f(\hat{x} + \lambda d) - f(\hat{x})}{\lambda} < 0, \quad \forall \lambda \in]-\delta, +\delta[\, . \tag{2.1}$$

La relation (2.1) est particulièrement vraie pour tout $\lambda \in]0, +\delta[$. On obtient le résultat cherché en multipliant la relation (2.1) par $\lambda > 0$.■

2.3 Schémas général des algorithmes d'optimisation sans contraintes

Supposons que d_k soit une direction de descente au point x_k.Ceci nous permet de considérer le point x_{k+1}, successeur de x_k, de la manière suivante :

$$x_{k+1} = x_k + \lambda_k d_k, \quad \lambda_k \in]0, +\delta[.$$

Vu la définition de direction de descente, on est assuré que

$$f\left(x_{k+1}\right) = f(x_k + \lambda_k d_k) < f(x_k).$$

Un bon choix de d_k et de λ_k permet ainsi de construire une multitude d'algorithmes d'optimisation.

2.3.1 Exemples de choix de directions de descente

Par exemple si on choisit $d_k = -\nabla f(x_k)$ et si $\nabla f(x_k) \neq 0$, on obtient la méthode du gradient. La méthode de Newton correspond à $d_k = -\left(H(x_k)\right)^{-1}.\nabla f(x_k)$. Bien sur $-\nabla f(x_k)$ est une direction de descente $(\nabla f(x_k)^t.d_k = -\nabla f(x_k)^t.\nabla f(x_k) = -\left\|\nabla f(x_k)\right\|^2 < 0)$. Pour la deuxième direction si la matrice hessienne $H(x_k)$ est définie positive alors $d_k = -\left(H(x_k)\right)^{-1}.\nabla f(x_k)$ est aussi une direction de descente.

2.3.2 Exemple de choix de pas λ_k

On choisit en général λ_k de façon optimale, c'est à dire que λ_k doit vérifier

$$f(x_k + \lambda_k d_k) \leq f(x_k + \lambda d_k) : \quad \forall \lambda \in [0, +\infty[.$$

En d'autres temes on est ramené à étudier à chaque itération un problème de minimisation d'une variable réelle. C'est ce qu'on appelle recherche linéaire.

2.4 Conditions nécessaires d'optimalité

2.4.1 Condition nécessaire d'optimalité du premier ordre

Théorème 2.2 *Soit* $f : R^n \to R$ *differentiable au point* $\hat{x} \in R^n$. *Si* $\hat{x}$ *est un minimum local de* (P) *alors* $\nabla f(\hat{x}) = 0$.

Preuve :

C'est une conséquence directe du théorème 2.1 et de la remarque 2.1. En effet, supposons que $\nabla f(\hat{x}) \neq 0$. Puisque la direction $d = -\nabla f(\hat{x})$ est une direction de descente, alors il existe $\delta > 0$ tel que :

$$f(\hat{x} + \lambda d) < f(\hat{x}) : \quad \forall \lambda \in]0, \delta[.$$

Ceci est contradiction avec le fait que $\hat{x}$ est une solution optimale locale de (P). ■

2.4.2 Condition nécessaire d'optimalité du second ordre

Théorème 2.3 *Soit $f : R^n \to R$ deux fois differentiable au point $\hat{x} \in R^n$. Si $\hat{x}$ est un minimum local de (P) alors $\nabla f(\hat{x}) = 0$ et la matrice hessienne de f au point $\hat{x}$, qu'on note $H(\hat{x})$, est semi définie positive.*

Preuve :

Soit $x \in R^n$ quelconque, f étant deux fois différentiable au point $\hat{x}$,on aura pour tout $\lambda \neq 0$

$$f(\hat{x} + \lambda x) = f(\hat{x}) + \frac{1}{2}\lambda^2 x^t H(\hat{x}) x + \lambda^2 \left\|x^2\right\| \alpha(\hat{x}, \lambda x), \qquad \alpha(\hat{x}, \lambda x) \underset{\lambda \to 0}{\to} 0.$$

Ceci implique

$$\frac{f(\hat{x} + \lambda x) - f(\hat{x})}{\lambda^2} = \frac{1}{2} x^t H(\hat{x}) x + \left\|x^2\right\| \alpha(\hat{x}, \lambda x). \tag{2.2}$$

$\hat{x}$ est un optimum local, il existe alors $\delta > 0$ tel que

$$\frac{f(\hat{x} + \lambda x) - f(\hat{x})}{\lambda^2} \geq 0, \quad \forall \lambda \in]-\delta, +\delta[.$$

Si on prend en considération (2.2) et on passe à la limite quand $\lambda \to 0,\ \lambda \neq 0$, on obtient

$$x^t H(\hat{x}) x \geq 0, \quad \forall x \in R^n. \blacksquare$$

2.5 Conditions suffisantes d'optimalité

Théorème 2.4 *Soit $f : R^n \to R$ deux fois differentiable au point $\hat{x} \in R^n$. Si $\nabla f(\hat{x}) = 0$ et $H(\hat{x})$ est définie positive alors $\hat{x}$ est un minimum local strict de (P).*

Preuve :

f étant deux fois différentiable au point $\hat{x}$,on aura pour tout $x \in R^n$

$$f(x) = f(\hat{x}) + \frac{1}{2}(x-\hat{x})^t H(\hat{x})(x-\hat{x}) + \|(x-\hat{x})\|^2 \alpha(\hat{x},(x-\hat{x})), \quad \alpha(\hat{x},(x-\hat{x})) \underset{x\to\hat{x}}{\to} 0, \quad (\nabla f(\hat{x}) = 0). \tag{2.3}$$

Supposons que $\hat{x}$ n'est pas un optimum local strict. Alors il existe une suite $\{x_k\}_{k\in\mathbb{N}^*}$ telle que $x_k \neq \hat{x} \ : \ \forall k$ et

$$x_k \neq \hat{x} \ : \ \forall k, \quad x_k \underset{k\to\infty}{\to} \hat{x} \quad et \quad f(x_k) \leq f(\hat{x}). \tag{2.4}$$

Dans (2.3) prenons $x = x_k$, divisons le tout par $\|(x_k - \hat{x})\|^2$ et notons $d_k = \dfrac{(x_k - \hat{x})}{\|(x_k - \hat{x})\|}$, on obtient

$$\frac{f(x_k) - f(\hat{x})}{\|(x_k - \hat{x})\|^2} = \frac{1}{2} d_k^t H(\hat{x}) d_k + \alpha(\hat{x},(x_k - \hat{x})), \quad \alpha(\hat{x},(x_k - \hat{x})) \underset{k\to\infty}{\to} 0. \tag{2.5}$$

(2.4) et (2.5) impliquent

$$\frac{1}{2} d_k^t H(\hat{x}) d_k + \alpha(\hat{x},(x_k - \hat{x})) \leq 0, \quad \forall k.$$

D'autre part la suite $\{d_k\}_{k\in\mathbb{NN}^*}$ est bornée ($\|d_k\| = 1, \ \forall n$). Donc il existe une sous suite $\{d_k\}_{k\in N_1\subset\mathbb{N}}$ telle que

$$d_k \underset{k\to\infty,\ k\in N_1}{\to} \widetilde{d} \ .$$

Finalement lorsque $k \to \infty, \ k \in N_1$, on obtient

$$\frac{1}{2} \widetilde{d} H(\hat{x}) \widetilde{d} \leq 0.$$

La dernière relation et le fait que $\widetilde{d} \neq 0$ ($\left\|\widetilde{d}\right\| = 1$) impliquent que la matrice hessienne $H(\hat{x})$ n'est pas définie positive. Ceci est en contradiction avec l'hypothèse. ■

2.5.1 Convergence des algorithmes et fonctions multivoques

Notion d'algorithme en optimisation

La plupart des méthodes de résolution des problèmes d'optimisation sont de nature itérative, c'est à dire qu'à partir d'un point initial x_0, elles engendrent une suite infinie $x_1, x_2, ..., x_k, ...$ dont on espère qu'elle converge vers la solution optimale.

Définition 2.3 Un algorithme de résolution est un procédé itératif qui permet à partir d'un point initial x_0 d'engendrer la suite $x_1, x_2, ..., x_k, ...$

Un algorihtme est parfaitement défini par la donnée de l'application a qui à x_k associe $x_{k+1} = a(x_k)$. L'étude de la convergence de l'algorithme se ramène à l'étude des propriétés de a.

Exemple 2.1

$a : x \to a(x) = x - \lambda \nabla f(x)$

a représente la méthode du gradient

Un modèle général des algorithmes : les applications multivoques

Soit $f : R^n \to R$. Considérons le problème de minimisation sans contraintes (P) suivant :

$$(P) \qquad min\,\{f(x) : x \in R^n\}$$

On sait qu'on peut générer plusieurs classes d'algorithmes qui ont pour but de résoudre le problème (P) ou d'atteindre des points vérifiant les conditions d'optimalité. Il serait alors logique au lieu d'étudier la convergence d'un algorithme, d'établir une théorie ou un modéle assez général qui étudie la convergence d'une classe d'algorithmes au lieu d'un seul. Soient $a_1, a_2, a_3, ..., a_p$ des applications qui génèrent p algorithmes , c'est à dire qu'à un point x_k on associe les p points différents $a_1(x_k), a_2(x_k), ..., a_p(x_k)$. Celà veut dire qu'on peut considérer l'application qui à x_k fait correspondre $A(x_k) = (a_1(x_k), a_2(x_k), ..., a_p(x_k))$. Ceci conduit naturellement à un modéle dans lequel les algorithmes sont représentés par des applications multivoques, c'est à dire des applications de R^ndans $P(R^n)$; qui à $x \in R^n$ fait correspondre une partie de R^n. D'une façon générale on notera :

$$A : R^n \overset{m}{\to} R^n$$

une application multivoque de R^n dans $P(R^n)$.

Convergence globale

Définition 2.4 Nous dirons qu'un algorithme décrit par une application multivoque A, est globalement convergent si quelque soit le point de départ x_0 choisi, la suite $\{x_k\}$ engendrée par $x_{k+1} \in A(x_k)$ (ou une sous-suite) converge vers un point satisfaisant les conditions nécessaires d'optimalité (ou solution optimale).

Notation : Tout le long de ce chapitre on notera par Ω l'ensemble suivant :

$\Omega = \{x : x$ est solution optimale locale ou globale où x satisfait une condition nécessaire d'optimalité$\}$.

Ω s'appelle ensemble des solutions.

Applications multivoques fermées

Cette notion généralise la notion de continuité pour les fonctions univoques. Elle joue un rôle essentiel dans l'étude des problèmes de convergence.

Définition 2.5 Soit $A : R^n \overset{m}{\to} R^n$ une application multivoque. On dit que A est fermée en x si et seulement si :

$$\left.\begin{array}{c} \text{pour toute suite } \{x_k\}_{k\in N} \text{ telleque } x_k \to x \\ \text{et pour toute suite } \{y_k\}_{k\in N} \text{ telleque } y_k \to y \\ \text{avec } y_k \in A(x_k) \end{array}\right\} \text{ alors } y \in A(x)$$

A est dite fermée dans $S \subset R^n$ si et seulement si A est fermée en tout point $x \in S$.

Remarque 2.1

Si A est univoque continue, alors A est fermée.

En effet, soit $A : R^n \to R^n$ continu en x et soit

$$\left.\begin{array}{c} x_k \to x \\ A(x_k) = y_k \to y \end{array}\right\} \Rightarrow A(x) = y \text{ d'après la continuité de } A.$$

Exemple 2.2 : Exemple d'application multivoque non fermée. $\Omega = \{1\}$

$A(x) = \begin{cases} \left[\frac{3}{2} + \frac{1}{4}x, 1 + \frac{1}{2}x\right] \text{ si } x \geq 2 \\ \frac{1}{2}(x+1) \text{ si } x \prec 2 \end{cases}$

Si $x_0 \geq 2$; l'algorithme génère une suite $\{x_k\}$ qui tend vers 2.

Si $x_0 < 2$; l'algorithme génère une suite $\{x_k\}$ qui tend vers 1.

Démontrons que A n'est pas fermée en $\tilde{x} = 2$.

En effet, considérons $\{x_k\}$ tel que $x_k = 2 - \frac{1}{k}$, $A(2) = 2$

$A(x_k) = \frac{1}{2}(2 - \frac{1}{k} + 1) = \frac{3}{2} - \frac{1}{2k} \to \frac{3}{2}$

$\frac{3}{2} \neq A(2)$. Donc A est non fermée au point $\tilde{x} = 2$.

Composition d'applications multivoques

Défnition 2.6 Soient $A : R^n \overset{m}{\to} R^n$ et $B : R^n \overset{m}{\to} R^n$. La composée $B \circ A$ est l'application multivoque $C : R^n \overset{m}{\to} R^n$ définie par : $C(x) = \underset{y\in A(x)}{\cup} B(y)$

Théorème 2.5 (composition des applications multivoques fermées).

Soit $A : R^n \overset{m}{\to} R^n$ et $B : R^n \overset{m}{\to} R^n$. On suppose que :

1- A est fermée en $x \in X$ et que B est fermée sur $A(x)$.

2- toute suite $\{y_k\}$ telle que $y_k \in A(x_k)$ avec $x_k \to x$, admet une sous-suite convergente.

Alors l'application multivoque $C = B \circ A$ est fermée en x.

Corollaire 2.1 Soit $A : R^n \overset{m}{\to} R^n$ une application univoque et $B : R^n \overset{m}{\to} R^n$ une application multivoque. Si A est continue en x et si B est fermée en $A(x)$, alors $B \circ A$ est fermée en x.

Théorème de Zangwill

Définition 2.7 On dit que $h : R^n \to R$ est une fonction de descente (relativement à une application multivoque A) si elle est continue et possède les propriétés suivantes :

1- Si $x \notin \Omega$ alors $h(y) < h(x)$ pour tout $y \in A(x)$.

2- Si $x \in \Omega$ alors $h(y) \leq h(x)$ pour tout $y \in A(x)$.

Théorème 2.6. (Zangwill 1969).

Soit (P) le problème d'optimisation sans contraintes suivant :

$$(P) \qquad \{min\ f(x) : x \in R^n\}$$

et Ω l'ensemble des solutions.

Soit $A : R^n \overset{m}{\to} R^n$ une application multivoque et considérons une suite $\{x_k\}$ engendrée par l'algorithme, c'est à dire vérifiant $x_{k+1} \in A(x_k)$. Supposons que les trois conditions suivantes soient vérifiées :

C1- Les points $\{x_k\}$ sont tous contenus dans un ensemble compact $K \subset R^n$.

C2- Il existe une fonction de descente h.

C3- L'application multivoque A est fermée dans $R^n \setminus \Omega$ et $\forall x \in R^n \setminus \Omega$, $A(x) \neq 0$

Alors :

pour tout x limite d'une sous-suite convergente de la suite $\{x_k\}$, on a

$$x \in \Omega \ \text{ et } \ h(x_k) \to h(x).$$

Exemple de condition C1 :

Supposons d'une part que l'ensemble :

$$X_{f(x_0)} = \{x : x \in R^n : f(x) \leq f(x_0)\} \ \text{ est borné}$$

,

et que la fonction objectif f décroit sur la suite $\{x_k\}$ (ceci est possible si $\nabla f(x_k)^t.d_k < 0$; pour tout k). Alors on aura

$$\{x_k\} \subset X_{f(x_0)}$$

qui est compact.

Exemple de condition C2 :

On peut prendre sous certaines conditions comme fonction de descente soit $h(x) = f(x)$ ou $h(x) = |\nabla f(x)|$.

Les modes de convergence

Etant donné que notre objectif est d'accélérer la convergence de suites, il est naturel de définir la notion de rapidité ou mode de convergence des suites.

Définition 2.8 Soit $\{x_k\}_{k\in N}$ une suite dans R^n qui converge vers $\hat{x}$.

1- On dit que $\{x_k\}_{k\in N}$ converge vers x linéairement avec le taux si

$$\lim_{k\to\infty} \sup \frac{\|x_{k+1} - \hat{x}\|}{\|x_k - \hat{x}\|} = \alpha \prec 1$$

Lorsque $\|x_{k+1} - \hat{x}\| \simeq \alpha \|x_k - \hat{x}\|$, la convergence est dite linéaire asymptotique.

2- La convergence est dite superlinéaire d'ordre γ si

$$\lim_{k\to\infty} \sup \frac{\|x_{k+1} - \hat{x}\|}{\|x_k - \hat{x}\|^{\gamma}} \prec +\infty,\ \gamma \succ 1$$

3- On dit que $\{x_k\}_{k\in N}$ tend vers x de façon superlinéaire si

$$\lim_{k\to\infty} \frac{\|x_{k+1} - \hat{x}\|}{\|x_k - \hat{x}\|} = 0$$

Lorsque $\|x_{k+1} - \hat{x}\| \simeq l \|x_k - \hat{x}\|^{\gamma}$, la convergence est dite superlinéaire asymptotique.

2.5.2 Cas convexe

Théorème 2.7 ([Bazara]) *Soit* $f : R^n \to R$ *convexe et différentiable et* (PC) *le problème d'optimisation sans contraintes et convexe suivant*

$$\{\min\ f(x) : x \in R^n\} \quad (PC),$$

et $\widehat{x} \in R^n$. *Alors* $\widehat{x}$ *est un minimum global de* (PC) *si et seulement si*

$$\nabla f(\hat{x}) = 0 \tag{2.6}$$

Chapitre 3

THEOREMES DE CONVERGENCE ASSOCIES AUX METHDES A DIRECTIONS DE DESCENTE ET AUX RECHERCHES LINEAIRES INEXACTES

3.1 Recherches linéaires

La recherche linéaire consiste à trouver λ_k de façon à diminuer la fonction f *Suffisamment* le long de cette direction.

Ce *" suffisamment "* sera quantifié dans la suite dans la description des conditions dites d' Armijo, Wolfe, Goldstein&Price(recherches linéaires inexactes).

3.1.1 Principe des méthodes de descente

Le principe d'une méthode de descente consiste à faire les itérations suivantes

$$x_{k+1} = x_k + \lambda_k d_k, \lambda_k > 0 \tag{3.1}$$

tout en assurant la propriété

$$f(x_{k+1}) < f(x_k).$$

Le vecteur d_k est la direction de descente en x_k. Le scalaire λ_k est appelé le pas de la méthode à l'itération k.

On peut caractériser les directions de descente en x_k à l'aide du gradient :

Proposition 3.1 *Soit $d \in R^n$ vérifiant*

$$\nabla f\left(x\right)^{\top}.d < 0$$

alors d est une direction de descente en x.

Preuve. *on a pour $\lambda > 0$*

$$f(x + \lambda d) = f(x) + \lambda \nabla f(x)^{\top} d + \lambda \varepsilon(\lambda)$$

donc si on écrit

$$\frac{f(x + \lambda d) - f(x)}{\lambda} = \nabla f\left(x\right)^{\top} d + \varepsilon(\lambda)$$

on voit bien que pour λ suffisamment petit on aura

$$f(x + \lambda d) - f(x) < 0.$$

■

Ou encore que d fait avec l'opposé du gradient $-\nabla f\left(x\right)$ un angle θ strictement plus petit que $90°$:

$$\theta := \arccos \frac{-\nabla f\left(x\right)^{\top} d}{\left\|\nabla f\left(x\right)\right\| \left\|d\right\|} \in \left[0, \frac{\pi}{2}\right[.$$

L'ensemble des directions de descente de f en x,

$$\left\{d \in R^n : \nabla^{\top} f\left(x\right).d < 0\right\}$$

forme un demi-espace ouvert de R^n (illustration à la figure 3.1).

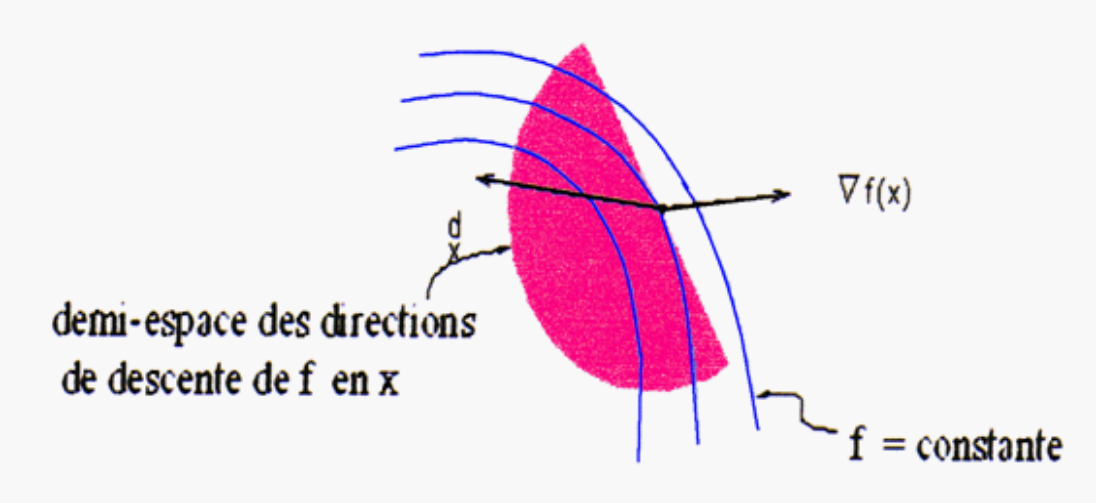


Fig.3.1 **Demi-espace (translaté) des direction de descente d de f en x**

De telles directions sont intéressantes en optimisation car, pour faire décroître f, il suffit de faire un déplacement le long de d.

Les méthodes à directions de descentes utilisent cette idée pour minimiser une fonction

Dans la méthode (3.1) le choix de λ_k est lié à la fonction

$$\varphi(\lambda) = f(x_k + \lambda d_k)$$

Comme dans la méthode de la direction de descente, la trajectoire de la solution suit un modèle de zigzag (voir figure [1.4].) Si λ est choisi tels que $f(x_k + \lambda d_k)$ soit le minimum dans chaque itération, alors les directions successives sont orthogonales.

En effet

si on note $g(x) = \nabla f(x)$

$$\frac{\mathbf{d}f(x_k + \lambda d_k)}{\mathbf{d}\lambda} = \sum_{i=1}^{n} \frac{\partial f(x_k + \lambda d_k)}{\partial x_{ki}} \frac{\mathbf{d}(x_{ki} + \lambda d_{ki})}{\mathbf{d}\lambda}$$

$$= \sum_{i=1}^{n} g_i(x_k + \lambda d_k) d_{ki}$$

$$= g\left(x_k + \lambda d_k\right)^\top d_k$$

où $g\left(x_k + \lambda d_k\right)$ est le gradient au point $x_k + \lambda d_k$.

En particulier, une façon de choisir λ_k peut être de résoudre le problème d'optimisation (à une seule variable)

$$\min_{\lambda>0} \varphi\left(\lambda\right) \tag{3.2}$$

Si le pas $\widetilde{\lambda}_k$ obtenu ainsi s'appelle le pas optimal alors nous pouvons écrire

$$\varphi'\left(\widetilde{\lambda}_k\right) = \nabla f(x_k + \widetilde{\lambda}_k d_k)^\top d_k = 0$$

c'est-à-dire

$$g\left(x_k + \widetilde{\lambda}_k d_k\right)^\top d_k = 0$$

ou bien

$$d_{k+1}^\top d_k = 0$$

où

$$d_{k+1} = -g\left(x_k + \widetilde{\lambda}_k d_k\right) = -g_{k+1}$$

est la direction de descente au point $x_k + \widetilde{\lambda}_k d_k$. Donc les directions successives d_k et d_{k+1} sont orthogonaux comme représenté dans la figure (3.2).

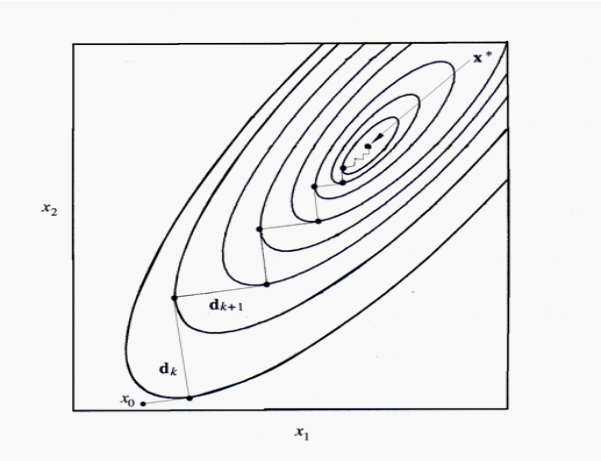


Fig.3.2 **Trajectoire d'une solution typique d'une méthode à direction de déscente**

Pour définir une direction de descente il faut donc spécifier deux choses :

♣ Dire comment la direction d_k est calculée. Ce choix influe directement dans la nomination de l'algorithme.

♣ Dire comment on détermine le pas λ_k ; c'est ce que l'on appelle : la recherche linéaire.

Algorithme1.1 (**méthode à directions de descente – une itération**)

Etape 0 : (initialisation)

On suppose qu'au début de l'itération k, on dispose d'un itéré

$x_k \in R^n$

Etape 1 :

Test d'arrêt : si $\|\nabla f(x_k)\| \simeq 0$, d'arrêt de l'algorithme ;

Etape 2 :

Choix d'une direction de descente $d_k \in R^n$;

Etape 3 :

Recherche linéaire : déterminer un pas $\lambda_k > 0$ le long de d_k de

manière à "*faire décroitre f suffisamment*" ;

Etape 4 :

Si la recherche linéaire est finie $x_{k+1} = x_k + \lambda_k d_k$;

remplacer k par $k+1$ et aller à l'étape 1. □

3.1.2 Recherche linéaire

Faire la recherche linéaire veut dire déterminer un pas λ_k le long d'une direction de descente d_k, autrement dit résoudre le problème unidimensionnel (3.2).

Notre intérêt pour la recherche linéaire ne vient pas seulement du fait que dans les application on rencontre, naturellement, des problèmes unidimensionnels, mais plutôt du fait que la recherche linéaire est un composant fondamental de toutes les méthodes traditionnelles d'optimisation multidimensionnelle

Objectifs à atteindre

il s'agit de réaliser deux objectifs

Le premier objectif

Consiste à faire décroître f suffisamment, cela se traduit le plus souvent par la réalisation d'une inégalité de la forme

$$f(x_k + \lambda_k d_k) \leq f(x_k) + \text{"un terme négatif"} \tag{3.3}$$

Le terme négatif, disons ν_k, joue un rôle-clé dans la convergence de l'algorithme utilisant cette recherche linéaire.

L'argument est le suivant.

Si $f(x_k)$ est minorée ($\exists c$ telleque $f(x_k) \geq c$ pour tout k), alors ν_k tend nécessairement vers zéro ($\nu_k \to 0$). c'est souvent à partir de la convergence vers zéro de cette suite que l'on parvient à montrer que le gradient lui-même doit tendre vers zéro. Le terme négatif devra prendre une forme bien particulière si on veut pouvoir en tirer de l'information.

En particulier, il ne suffit pas d'imposer $f(x_k + \lambda_k d_k) < f(x_k)$

Le second objectif

Consiste d'empêcher le pas $\lambda_k > 0$ d'être trop petit, trop proche de zéro.

Le premier objectif n'est en effet pas suffisant car l'inégalité (3.3) est en général satisfaite par des pas $\lambda_k > 0$ Arbitrairement petit.

Or ceci peut entraîner une "*fausse convergence*", c'est-à-dire la convergence des itérés vers un point non stationnaire.

Types de recherches linéaires

Il existe deux grandes classes de méthodes qui s'intéressent à l'optimisation unidimensionnelle :

a) Les recherches linéaires exactes.

b) Les recherches linéaires inexactes.

Les recherches linéaires exactes, malgré quelles n'aboutissent qu'à une solution optimale approchée, nécessitent beaucoup d'observations à chaque itération de l'algorithme principal.

Le mot exact prend sa signification dans le fait que si f est quadratique la solution de la recherche linéaire s'obtient d'une façon exacte est dans un nombre fini d'itérations.

Pour une fonction non linéaire (non quadratique) arbitraire,

- la détermination de ces pas demande en général beaucoup de temps de calcul et ne peut de toute façon pas être faite avec une précision infinie,

- l'efficacité supplémentaire éventuellement apportée à un algorithme par une recherche linéaire exacte ne permet pas, en général, de compenser le temps perdu à déterminer un tel pas,

- les résultats de convergence autorisent d'autres types de règles (recherche linéaire inexacte), moins gourmandes en temps de calcul.

On préfère imposer des conditions moins restrictives, plus facilement vérifiées, qui permettent toute fois de contribuer à la convergence des algorithmes. En particulier, il n'y aura plus un unique pas (ou quelques pas) vérifiant ces conditions mais tout un intervalle de pas (ou plusieurs intervalles), ce qui rendra d'ailleurs leur recherche plus aisée. C'est ce que l'on fait avec les règles d'Armijo, de Goldstein et de Wolfe décrites dans la prochaine section.

3.1.3 Recherches linéaires inexactes

On considère la situation qui est typique pour l'application de la technique de recherche linéaire à l'intérieur de la méthode principale multidimensionnelle.

Sur une itération k de la dernière méthode nous avons l'itération courante $x_k \in R^n$ et la direction de recherche $d_k \in R^n$ qui est direction de descente pour notre objectif : $f : R^n \to R$:

$$\nabla f\left(x_k\right)^{\top} . d_k < 0 \tag{3.4}$$

Le but est de réduire "de façon importante"la valeur de l'objectif par un pas $x_k \to x_{k+1} = x_k + \lambda_k d_k$ de x_k dans la direction d_k.

Pour cela de nombreux mathématiciens (Armijo, Goldstein, Wolfe, Albaali, Lemaréchal, Fletcher...) ont élaboré plusieurs règles.

l'objectif de cette section consiste à présenter les principales tests.

D'abord présentons le schéma d'une recherche linéaire inexacte.La règle d'Armijo, La règle de Goldstein, La règle de Wolfe)

3.1.4 La recherche linéaire inexacte d'Armijo

Soit $f : R^n \to R$, $x \in R^n$, $d \in R^n$ tel que $\nabla f^T(x_k)\, d_k < 0$. Définisons la fonction $\varphi : R \to R$ avec $\varphi(\lambda_k) = f(x_k + \lambda_k d_k) \qquad \lambda_k \geq 0$

Notons que :

$$\begin{aligned} \varphi'(\lambda) &= \nabla f^T(x_k + d_k)\, d_k, \\ \varphi'(0) &= \nabla f^T(x_k)\, d_k < 0, \\ \varphi(0) &= f(x_k) \end{aligned}$$

L'equation de la tangente au point $(0, \varphi(0))$ est la suivante

$$\begin{aligned} \{\lambda, y\} &: y = \varphi(0) + \varphi'(0)(\lambda - 0). \\ \varphi_1(\lambda_k) &= f(x_k) + \nabla f^T(x_k)\, d_k \lambda_k \end{aligned}$$

Posons

$$\varphi_1(\lambda) = \varphi(0) + \varphi'(0)\lambda.$$

L'équation de la tangente devient :

$$\varphi_1(\lambda) = f(x_k) + \nabla f^T(x_k)\, d_k \lambda$$

Définissons maintenant la fonction $\hat{\varphi}(\lambda)$ comme suit :

$$\hat{\varphi}(\lambda) = f(x_k) + \varepsilon\lambda\dot{\varphi}(0) = f(x_k) + \varepsilon\lambda\nabla f^T(x_k)\, d_k; \quad 0 < \varepsilon < 1 \tag{3.5}$$

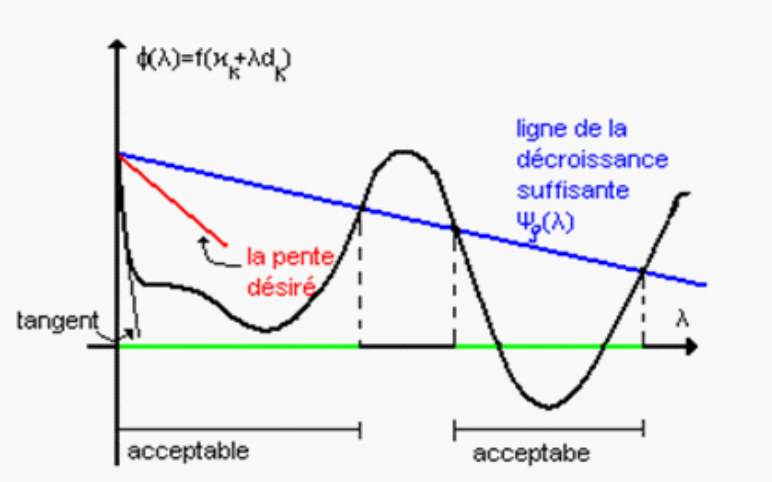


Fig3.1-Règle d'Amijo

On cherche $\bar{\lambda}_k$ tel que

$$\varphi\left(\bar{\lambda}_k\right) \leq \hat{\varphi}\left(\bar{\lambda}_k\right)$$

Remarques

1-La condition $\varphi\left(\bar{\lambda}_k\right) \leq \hat{\varphi}\left(\bar{\lambda}_k\right)$ implique la decroissance de la fonction f.

En effet

$$\begin{aligned} \varphi\left(\bar{\lambda}_k\right) &\leq \hat{\varphi}\left(\bar{\lambda}_k\right) \\ f\left(x_k+\bar{\lambda}_k d_k\right) &\leq f\left(x_k\right)+\varepsilon\bar{\lambda}_k \nabla f\left(x_k\right) d_k < f\left(x_k\right) \end{aligned}$$

car la direction d est une direction de descente.

2-Il ne faut pas prende $\bar{\lambda}_k$ très proche de zero car celà va alterer la convergence et la vitesse de convergence.

En effet

$$\begin{aligned} f\left(x_k+\bar{\lambda}_k d_k\right) &= f\left(x_k\right)+\bar{\lambda}_k \nabla f\left(x_k\right) d_k+\bar{\lambda}_k \alpha\left(x_k,\bar{\lambda}_k d_k\right) \\ f\left(x_{k+1}\right)-f\left(x_k\right) &= \bar{\lambda}_k\left[\nabla f\left(x_k\right) d_k+\alpha\left(x_k,\bar{\lambda}_k d_k\right)\right] \\ \text{si } \bar{\lambda}_k &\rightarrow 0 \qquad \alpha\left(x_k,\bar{\lambda}_k d_k\right) \underset{\bar{\lambda}_k \rightarrow 0}{\rightarrow} 0 \quad \text{donc} \quad f\left(x_{k+1}\right) \simeq f\left(x_k\right) \end{aligned}$$

Algorithme 3.1 (Règle d'Armijo)

Etape 0 : (initialisation)

$\alpha_{g,1} = \alpha_{d,1} = 0$, choisir $\alpha_1 > 0$, $\rho \in]0,1[$ poser $k = 1$ et aller à l'étape 1.

Etape 1 :

si $\varphi_k(\alpha_k) \leq \varphi_k(0) + \rho\varphi'_k(0)\alpha_k$: STOP ($\alpha^* = \alpha_k$).

si $\varphi_k(\alpha_k) > \varphi_k(0) + \rho\varphi'_k(0)\alpha_k$, alors

$\alpha_{d,k+1} = \alpha_d$,$\alpha_{g,k+1} = \alpha_k$ et aller à l'étape 2.

Etape 2 :

si $\alpha_{d,k+1} = 0$ déterminer $\alpha_{k+1} \in]\ \alpha_{g,k+1}\ ,\ +\infty\ [$

si $\alpha_{d,k+1} \neq 0$ déterminer $\alpha_{k+1} \in]\ \alpha_{g,k+1}\ ,\ \alpha_{d,k+1}\ [$

remplacer k par $k+1$ et aller à l'étape 1. □

Programme en fortran 77 d'Armijo

```
c*******************************************************************************
                    Programme en fortran 77 d' Armijo
c*******************************************************************************
      real x(2),xp(2),g(2),gp(2),f0,alph,f,p,s,a,b,pf,t,y,phopt
      integer i,k,test,n
      sig=0.3 ;ro=0.1 ;b=10000000 ;alph=10 ;test=0 ;k=0
      print*,'le dimontion n=' ;read*,n
      do i=1,n print*,'x0(',i,')=' ;read*, xp(i) end do10
     if (test==0.and.k.le.10) then call fonc(f0,gp,xp)
      do i=1,n x(i)=xp(i)-alph*gp(i) end do
      call fonc(f,g,x) ;p=0
      do i=1,n p=p+gp(i)*gp(i) end do
      s=f0-ro*alph*p
     if (f.gt.s) then b=alph else phopt=alph ;test=1 endif
      print*,'alpha(',k,')=', alph ;alph=b/2 ;k=k+1
      go to 10
      end if
      print*,'alpha optimal=', phopt
      end
c*******************************************************************************
      subroutine fonc(f,g,x)
```

```
      real f,g(2),x(2)
      f=(x(1))**2+(x(2))**4 ;g(1)=2*x(1) ;g(2)=4*(x(2))**3
      end
c*************************************************************************
c*************************************************************************
```

Remarque 3.1

Il est clair d'après la figure $Fig3.1-$Règle d'Amijo que l'inégalité d'Armijo est toujours vérifiée si $\alpha_k \succ 0$ est suffisamment petit. en effet, dans le cas contraire, on aurait une siute de pas strictement positifs $\{\alpha_{k,i}\}_{i\geq 1}$ convergeant vers 0 lorsque $i \to \infty$ et tels que $f(x_k + \alpha_k d_k) \leq f(x_k) + \rho\alpha_k \nabla^T f(x_k) d_k$ n'ait pas lieu pour $\alpha_k = \alpha_{k,i}$.

En retranchant $f(x_k)$ dans les deux membres, en divisant par $\alpha_{k,i}$ et en passant à la limite quand $i \to \infty$, on trouverait

$$\nabla^T f(x_k) d_k \geq \rho \nabla^T f(x_k) d_k$$

ce qui contredirait le fait que d_k est une direction de descente ($\rho < 1$). $\square$

Dans le théorème suivant on va assurer l'existence du pas d'Armijo en posant quelques conditions sur la fonction φ_k.

Théorème 3.1 *Si $\varphi_k : R_+ \to R$;définie par $\varphi_k(\alpha) = f(x_k + \alpha d_k)$ est continue et bornée inférieurement, si d_k est une direction de descente en x_k ($\varphi'_k(0) < 0$) et si $\rho \in]0, 1[$, alors l'ensemble des pas vérifiant la règle d'Armijo est non vide.*

Preuve

On a

$$\begin{aligned} \varphi_k(\alpha) &= f(x_k + \alpha d_k) \\ \psi_\rho(\alpha) &= f(x_k) + \rho\alpha_k \nabla^T f(x_k) d_k \end{aligned}$$

Le développement de Taylor-Yong en $\alpha = 0$ de φ_k est
$\varphi_k(\alpha) = f(x_k + \alpha d_k) = f(x_k) + \rho\alpha_k \nabla^T f(x_k) d_k + \alpha\xi(\alpha)$ où $\xi(\alpha) \to 0$, $\alpha \to 0$:
et comme$\rho \in]0\,;\,1[$ et $\varphi'_k(0) = \nabla^T f(x_k) d_k < 0$ on déduit :
$f(x_k) + \alpha_k \nabla^T f(x_k) d_k < f(x_k) + \rho\alpha_k \nabla^T f(x_k) d_k$ pour $\alpha > 0$
On voit que pour $\alpha > 0$ assez petit on a :

$$\varphi_k(\alpha) < \psi_\rho(\alpha)$$

De ce qui précède et du fait que φ_k est bornée inférieurement, et $\psi_\rho(\alpha) \to -\infty$, $\alpha \to +\infty$; on déduit que la fonction $\psi_\rho(\alpha) - \varphi_k(\alpha)$ a la

propriété :

$$\begin{cases} \psi_\rho(\alpha) - \varphi_k(\alpha) \succ 0 \ \text{ pour } \alpha \text{ assez petit} \\ \psi_\rho(\alpha) - \varphi_k(\alpha) \prec 0 \ \text{ pour } \alpha \text{ assez grand} \end{cases}$$

donc s'annule au moins une fois pour $\alpha > 0$:

En choisissant le plus petit de ces zéros on voit qu'il existe $\bar{\alpha} > 0$ tel que

$\varphi_k(\bar{\alpha}) = \psi_\rho(\bar{\alpha})$ et $\varphi_k(\alpha) < \psi_\rho(\alpha)$ pour $0 < \alpha < \bar{\alpha}$

Ce qui achève la démonstration. $\square$

Décrivons maintenat la règle de Goldstein&Price.

3.1.5 La recherche linéaire inexacte de Goldstein

En ajoutant une deuxième inégalité $\varphi(\lambda) \geq \varphi(0) + \sigma_2 \dot{\varphi}(0)$ a la régle d'Armijo on obtient la règle de Goldstein, avec σ_1, σ_2 sont deux constantes vérifiant $0 < \sigma_1 < \frac{1}{2} < \sigma_2 < 1$. Les deux inégalités de la regle de Goldstein sont donc :

$$\begin{aligned} \varphi\left(\bar{\lambda}_k\right) &\leq \hat{\varphi}\left(\bar{\lambda}_k\right) \quad et \\ \varphi\left(\bar{\lambda}_k\right) &\geq \varphi(0) + \sigma_2 \dot{\varphi}(0). \end{aligned}$$

Soit en remplaçant

$$f\left(x_k + \bar{\lambda}_k d_k\right) \leq f(x_k) + \sigma_1 \bar{\lambda}_k \nabla f(x_k) d_k \tag{3.6}$$

$$et$$

$$f\left(x_k + \bar{\lambda}_k d_k\right) \geq f(x_k) + \sigma_2 \bar{\lambda}_k \nabla f(x_k) d_k \tag{3.7}$$

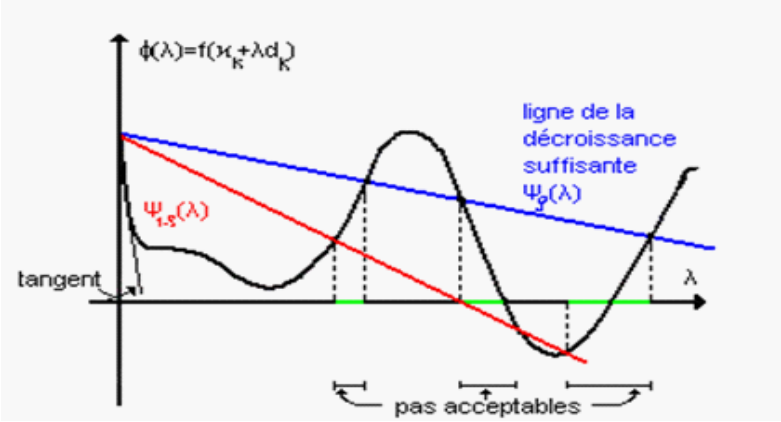


Fig3.2-Règle de Goldstein

La figure 3.2 montre, sur un exemple, l'ensembles des points satisfaisant les deux conditions Goldstein.

L'Algorithme de Goldstein

L' Algorithme essaye de trouver $\alpha_k \in]\beta_1, \beta_2[$ (voir fig3.1). On demarre avec un intervalle $[a_0, b_0]$ assez grand. On prend $\alpha_0 \in [a_0, b_0]$

Si α_0 vérifié Goldstein1 et Goldstein2

alors $\alpha_0 \in]\beta_1, \beta_2[$ et on s'arréte .

Si $\alpha_0 > \beta_2$

alors α_0 ne verifié pas Goldstein1, alors on prend $b_1 = \alpha_0$ et $a_1 = b_0$ et $\alpha_1 = \dfrac{a_1 + b_1}{2}$

et on recommence avec α_1

Si $\alpha_0 < \beta_1$

alors α_0 ne vérifié pas Goldstein2, on prend $a_1 = \alpha_0, b_1 = b_0$ et $\alpha_1 = \dfrac{a_1 + b_1}{2}$ et on teste de nouveau α_1

A l'itération k

Supposons qu'on ait $[a_k, b_k]$ et $\alpha_k = \dfrac{a_k + b_k}{2}$

Si α_k vérifie Goldstein1 et Goldstein2, $\alpha_k \in]\beta_1, \beta_2[$. Stop.

Si α_k ne vérifié pas Goldstein1 alors $\alpha_k > \beta_2$

On prend$b_{k+1} = \alpha_k$, $a_{k+1} = a_k$, $\alpha_{k+1} = \dfrac{a_{k+1} + b_{k+1}}{2}$

Si α_k ne vérifie pas Goldstein2 alors $\alpha_k < \beta_1$. On prend $a_{k+1} = \alpha_k$, $b_{k+1} = b_k$, $\alpha_{k+1} = \dfrac{a_{k+1} + b_{k+1}}{2}$

On obtient ainsi l'algorithme suivant :

Algorithme3.2 de Goldstein

ETAPE 1 (Initialisation)

Choisir $\alpha_0 \in [0, 10^{100}]$ et $\rho \in]0, 1[$. Poser $a_0 = 0$, $b_0 = 10^{100}$

Poser $k = 0$ et aller à ETAPE2

ETAPE2 (Test Goldstein1)

Iteration k on a $[a_k, b_k]$ et α_k, calculez $\varphi_k(\alpha_k)$

Si $\varphi_k(\alpha_k) \leq \varphi_k(0) + \rho\alpha_k\varphi'_k(0)$, allez à ETAPE 3

Sinon

Poser $b_{k+1} = \alpha_k$, $a_{k+1} = a_k$, et allez à ETAPE 4

ETAPE 3 (Test Gold 02)

Si $\varphi_k(\alpha_k) \geq \varphi_k(0) + (1 - \rho)\alpha_k\varphi'_k(0)$ stop. $\alpha^* = \alpha_k$

Sinon Poser $a_{k+1} = \alpha_k$, $b_{k+1} = b_k$ et allez à ETAPE 4

ETAPE 4

Poser $\alpha_{k+1} = \dfrac{a_{k+1} + b_{k+1}}{2}$

Poser $k = k + 1$ et allez à ETAPE 2 .

Programme en fortran 77 de Goldstein

```
c*************************************************************************
          Programme en fortran 77 de Goldschtien
c*************************************************************************
      real x(2),xp(2),g(2),gp(2),f0,alph,f,p,s,a,b,pf,t,y,phopt
      integer i,k,test,n
      ro=0.1 ;delt=0.3 ;b=10000000 ;alph=10 ;test=0 ;k=0
      print*,'le dimontion n=' ;read*,n
      do i=1,n print*,'x0(',i,')=' ;read*, xp(i) end do
10    if (test==0.and.k.le.1000) then call fonc(f0,gp,xp)
      do i=1,n x(i)=xp(i)-alph*gp(i) end do
      call fonc(f,g,x) ;p=0
      do i=1,n p=p+gp(i)*gp(i) end do
      s=f0-ro*alph*p ;y=f0-delt*alph*p
     if (f.gt.s) then b=alph else if (f.lt.y) then a=alph else
      phopt=alph ;test=1 endif
      endif
      print*,'alpha(',k,')=', alph ;alph=(a+b)/2 ;k=k+1
```

```
      go to 10
      end if
      print*,'alpha optimal=', phopt
      end
c*************************************************************************
      subroutine fonc(f,g,x)
      real f,g(2),x(2)
      f=(x(1))**2+(x(2))**4 ;g(1)=2*x(1) ;g(2)=4*(x(2))**3
      end
c*************************************************************************
c*************************************************************************
```

Dans le théorème suivant on va assurer l'existence du pas de Goldstein&price en posant quelques conditions sur la fonction φ_k.

Théorème3.2 *Si $\varphi_k : R_+ \to R$;définie par $\varphi_k(\alpha) = f(x_k + \alpha d_k)$ est continue et bornée inférieurement, si d_k est une direction de descente en x_k et si $\rho \in]0, 1[$, $\delta \in]\rho, 1[$, alors l'ensemble des pas vérifiant la règle de Goldstein&Price (3.6)-(3.7) est non vide.*

Preuve

On a

$$\begin{aligned} \varphi_k(\alpha) &= f(x_k + \alpha d_k) \\ \psi_\rho(\alpha) &= f(x_k) + \rho\alpha\nabla^T f(x_k)d_k \\ \psi_\delta(\alpha) &= f(x_k) + \delta\alpha\nabla^T f(x_k)d_k \end{aligned}$$

Le développement de Taylor-Yong en $\alpha = 0$ de φ_k est :

$$\varphi_k(\alpha) = f(x_k + \alpha d_k) = f(x_k) + \rho\alpha\nabla^T f(x_k)d_k + \alpha\xi(\alpha) \text{ où } \xi(\alpha) \to 0, \alpha \to 0.$$

et comme $\rho \in]0, 1[$ et $\varphi'_k(0) = \nabla^T f(x_k)d_k < 0$ on déduit :

$$f(x_k) + \alpha\nabla^T f(x_k)d_k < f(x_k) + \delta\alpha\nabla^T f(x_k)d_k < f(x_k) + \rho\alpha\nabla^T f(x_k)d_k \text{ pour } \alpha > 0$$

On voit que pour $\alpha > 0$ assez petit on a :

$$\varphi_k(\alpha) < \psi_\delta(\alpha) < \psi_\rho(\alpha)$$

De ce qui précède et du fait que φ_k est bornée inférieurement,

et $\psi_\rho(\alpha) \to -\infty; \alpha \to +\infty$, on déduit que la fonction $\psi_\rho(\alpha) - \varphi_k(\alpha)$ a la propriété :

$$\begin{cases} \psi_\rho(\alpha) - \varphi_k(\alpha) > 0 & \text{pour } \alpha \text{ assez petit} \\ \psi_\rho(\alpha) - \varphi_k(\alpha) < 0 & \text{pour } \alpha \text{ assez grand} \end{cases}$$

donc s'annule au moins une fois pour $\alpha > 0$.

En choisissant le plus petit de ces zéros on voit qu'il existe $\bar{\alpha} > 0$ tel que

$$\varphi_k(\bar{\alpha}) = \psi_\rho(\bar{\alpha}) \text{ et } \varphi_k(\alpha) < \psi_\rho(\alpha) \text{ pour } 0 < \alpha < \bar{\alpha}$$

De la meme manière, il existe $\tilde{\alpha} > 0$ tel que :

$$\varphi_k(\tilde{\alpha}) = \psi_\delta(\tilde{\alpha}) \text{ et } \varphi_k(\alpha) < \psi_\delta(\alpha) \text{ pour } 0 < \alpha < \tilde{\alpha}$$

et comme $\psi_\delta(\alpha) < \psi_\rho(\alpha)$ pour $\alpha > 0$, forcément $\tilde{\alpha} < \bar{\alpha}$ et $\alpha = \bar{\alpha}$ satisfait (3.6)-(3.7)

$$\left(\begin{array}{c} \psi_\delta(\tilde{\alpha}) = \ \varphi_k(\tilde{\alpha}) < \psi_\rho(\alpha) \text{ n'est autre que} \\ f(x_k) + \delta\tilde{\alpha}\nabla^T f(x_k) d_k = f(x_k + \tilde{\alpha} d_k) < f(x_k) + \rho\tilde{\alpha}\nabla^T f(x_k) d_k \end{array} \right)$$

Ce qu'il fallait démontrer. □

3.1.6 La recherche linéaire inexacte de Wolfe

La règle de Goldstein&Price peuvent exclure un minimum ce qui est peut être un inconvénient.

Les conditions de wolfe ([35, 1969]) n'ont pas cet inconvénient.

La règle de Wolfe fait appel au calcul de $\varphi'(\lambda)$, elle est donc en théorie plus couteuse que la règle de Goldstein. Cependant dans de nombreuses applications, le calcul du gradient $\nabla f(x)$ représente un faible cout additionnel en comparaison du coute d'évaluations de $f(x)$, c'est pourquoi cette règle est très utilisée.

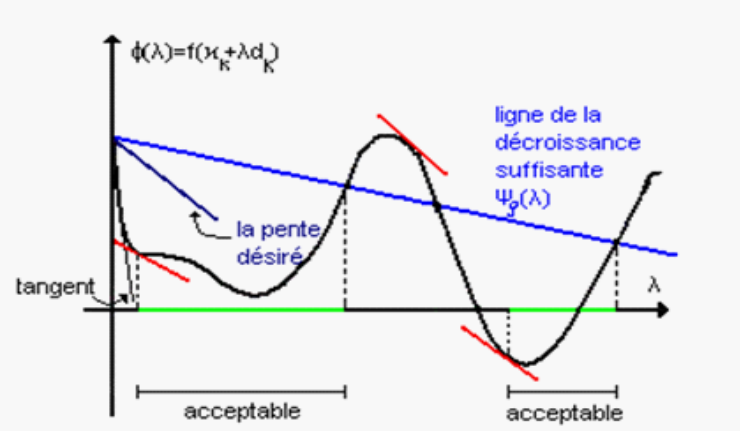


Fig3.3-Règle de Wolfe

Le figure 3.3 montre sur un exemple l'ensembles des points satisfaissant les conditions de Wolfe $\sigma_1 = 0,1$, $\sigma_2 = 0,7$ (*Lemaréchal 1980).*

Les deux conditions de la règle Wolfe sont donc les suivantes :

$$f\left(x_k + \alpha_k d_k\right) \leq f\left(x_k\right) + \sigma_1 \alpha_k \nabla f\left(x_k\right) d_k \tag{3.8}$$

et

$$\nabla f\left(x_k + \alpha_k d_k\right)^T d_k \geq \sigma_2 \nabla f\left(x_k\right)^T d_k \tag{3.9}$$

avec

$$0 < \sigma_1 < \sigma_2 < 1.$$

Remarque3.2

Soit $\alpha^* > 0$ tel que $\varphi'_k(\alpha^*) = 0$. Alors α^* vérifie (3.8) et (3.9).

En effet, α^*T.q $\varphi'(\alpha^*) = 0$ vérifié (3.8) car

$\delta\varphi'(0) < 0 = \varphi'(\alpha^*)$, donc $\delta\varphi'(0) < \varphi'(\alpha^*)$ donc α^* vérifié (3.9) .

$_*$ Si ρ est suffisament petit (on prend en général $\rho = 10^{-4}$) et si α^* exacte alors α^* vérifié (3.8) .

Par conséquent $\alpha^* \in]\beta_1, \beta_2[$

3.1.7 Rechreche linéaire inexacte de Wolfe forte

α_k vérifie les condition de Wolfe forte si les deux conditions suivantes sont vérifieés :

$$\varphi_k(\alpha_k) \leq \varphi_k(0) + \rho\alpha_k\varphi'_k(0), \quad 0 < \rho < \frac{1}{2} \tag{3.10}$$

$$|\varphi'_k(\alpha_k)| \leq -\sigma\varphi'_k(0), \quad 0 < \sigma < 1 \ , \ \ \sigma \cap \rho \tag{3.11}$$

Théorème3.3

Si α_k vérifie (3.10) et (3.11), alors α_k vérifie (3.8) et (3.9) .

Preuve

(3.10) et *(3.11)* sont les mêmes propositions. Reste à monter que

$$(3.11) \Longrightarrow (3.9)$$

En effet. Supposons que α_k vérifie (3.11). Alors on a

$$\sigma\varphi'_k(0) \leq \varphi'_k(\alpha_k) \leq -\sigma\varphi'_k(0)$$

Par conséquent on a

$$\varphi'_k(\alpha_k) \geq \sigma\varphi'_k(0)$$

Ceci implique que α_k vérifie (3.9) ■

Remarque 3.3

Supposons qu'on a une suite de paramètres $\{\sigma_j\}_{j\in\mathbb{N}}$ telle que

$$\sigma_j \to 0, \ j \to \infty$$

Fixons $k \in \mathbb{N}$. Supposons aussi que pour k fixé, (3.10) et (3.11). On obtient donc une suite $\{\alpha_{k,j}\}_{kfixe,j\in\mathbb{N}}$

$$\forall j \in \mathbb{N} : \sigma_j\varphi'_k(0) \leq \varphi'_k(\alpha_{kj}) \leq -\sigma_j\varphi'_k(0)$$

Ceci implique

$$\varphi'_k(\alpha_{kj}) \underset{j\to\infty, \ kfixé}{\to} 0$$

La relation précédente implique que

$$\varphi'_k(\alpha_k) \underset{k\to\infty}{\to} 0$$

Si φ'_k est continue et si $\alpha_j \to \bar{\alpha}$ alors $\bar{\alpha} = \alpha^*$.

Conclusion

Dans la rechreche linéaire inexacte de Wolfe forte, si on choisit σ trés petit, on a beaucoup de chances de tomber sur α_k proche de α^*. L'inconvénient est que ce choix (σ trés petit) entraine beaucoup d'itérations pour parvenir à obtenir α_k vérifiant (3.10) et (3.11). On pourrait choisir une suit $\{\sigma_j\}$ telle que $\sigma_j \to 0$ lentement. Ces choix influent sur la convergence et la vitesse de convergence .

Remarque3.4

Considérons la condition (3.9)

$$\varphi'_k(\alpha_k) \geq \sigma\varphi'_k(0)$$

Contrairement à (3.11), lorsque $\sigma \to 0$, $\varphi'(\alpha_k)$ peut ne pas tendre vers 0.

L'algoritheme de wolfe

ETAPE 1 Initialisation

Prendre $\alpha_0 \in [0, 10^{99}]$, calculez $\varphi(0)$, $\varphi'(0)$. Prendre $\rho = 0.1$ (ou $\rho = 0.1$ ou $\rho = 0.001$ ou $\rho = 10^{-4}$)
$\sigma = 0.9$ (ou σ plus petit encore)
Poser $a_0 = 0, b_0 = 10^{99}$, $k = 0$ et allez à ETAPE 2

ETAPE 2 (test de (Wolfe1))

Calculez $\varphi(\alpha_k)$. Si $\varphi(\alpha_k) \leq \varphi(0) + \rho\alpha_k\varphi'(0)$, aller à ETAPE 3. Sinon
Poser $a_{k+1} = a_k$, $b_{k+1} = \alpha_k$ et allez à ETAPE 4

ETAPE 3 (test (Wolfe2) ou (Wolfeforte2))

Calculez $\varphi'(\alpha_k)$. Si $\varphi'(\alpha_k) \geq \sigma\varphi'(0)$ ($|\varphi'_k(\alpha_k)| \leq -\sigma\varphi'_k(0)$). STOP
Prendre $\bar{\alpha} = \alpha_k$. Sinon Poser $a_{k+1} = \alpha_k$, $b_{k+1} = b_k$ et allez à ETAPE 4

ETAPE 4 (calcul de α_{k+1})

$\alpha_{k+1} = \dfrac{a_{k+1} + b_{k+1}}{2}$

Poser $k = k + 1$ et allez à ETAPE 2 .

Programme en fortran 77 de wolfe

c**********

Programme en fortran 77 de wolfe

```
c*************************************************************************
      real x(2),xp(2),g(2),gp(2),f0,alph,f,p,s,a,b,pf,t,y,phopt
      integer i,k,test,n
      sig=0.3 ;ro=0.1 ;b=10000000 ;alph=10 ;test=0 ;k=0 ;print*,'le dimontion n='
      read*,n
      do i=1,n print*,'x0(',i,')=' ; read*, xp(i) end do
10 if (test==0.and.k.le.10) then call fonc(f0,gp,xp)
      do i=1,n x(i)=xp(i)-alph*gp(i) end do
      call fonc(f,g,x) ;p=0
      do i=1,n p=p+gp(i)*gp(i) end do
      s=f0-ro*alph*p
     if (f.gt.s) then b=alph else pf=0
      do i=1,n pf=pf+g(i)*gp(i) end do
      t=-pf ;y=sig*(-p)
      if (t.lt.y) then a=alph else phopt=alph ;test=1 endif
      endif
      print*,'alpha(',k,')=', alph ;alph=(a+b)/2 ;k=k+1
      go to 10
      end if
      print*,'alpha optimal=', phopt ;x=xp
      end
c*************************************************************************
      subroutine fonc(f,g,x)
      real f,g(2),x(2)
      f=(x(1))**2+(x(2))**4 ;g(1)=2*x(1) ;g(2)=4*(x(2))**3
      end
c*************************************************************************
c*************************************************************************
```

Remarque3.5

La règle de Wolfe fait appel au calcul de φ'_k, elle est donc en théorie plus coûteuse que la règle de Goldstein&Price.

Cependant dans de nombreuses applications, le calcul du gradient $\nabla f(x)$ représente un faible cout additionnel en comparaison du cout d'évaluations de $f(x)$, c'est pourquoi cette règle est très utilisée. □

3.1.8 La règle de Wolfe relaxée

Proposée par Dai et Yuan, cette règle consiste à choisir le pas α_k satisfaisant aux conditions :

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \rho\alpha_k \nabla^T f(x_k) d_k \tag{3.12}$$

$$\sigma_1 \nabla^T f(x_k) d_k \leq \nabla f(x_k + \alpha_k d_k)^T d_k \leq -\sigma_2 \nabla^T f(x_k) d_k \tag{3.13}$$

autrement dit :

$$\varphi_k(\alpha_k) \leq \varphi_k(0) + \rho\varphi_k(0)\alpha_k \tag{3.12bis}$$

$$\sigma_1 \varphi'_k(0) \leq \varphi'_k(\alpha_k) \leq -\sigma_2 \varphi'_k(0) \tag{3.13bis}$$

où $0 < \rho < \sigma_1 < 1$ et $\sigma_2 > 0$.

Remarque3.6

On voit bien que les conditions de Wolfe relaxée impliquent les conditions de Wolfe fortes. Effectivement (3.10) est équivalente à (3.8), tandis que pour le cas particulier $\sigma_1 = \sigma_2 = \sigma$, (3.11) est équivalente à (3.9). En effet :

$$\begin{aligned} \sigma_1 \nabla^T f(x_k) d_k \quad &\leq \quad \nabla^T f(x_k + \alpha_k d_k) d_k \leq -\sigma_2 \nabla^T f(x_k) d_k \\ &\Rightarrow \quad \sigma \nabla^T f(x_k) d_k \leq \nabla^T f(x_k + \alpha_k d_k) d_k \leq -\sigma \nabla^T f(x_k) d_k \\ &\Rightarrow \quad \left|\nabla^T f(x_k + \alpha_k d_k) d_k\right| \leq -\sigma \nabla^T f(x_k) d_k \qquad \square \end{aligned}$$

Remarque3.7

. Les conditions de Wolfe relaxée impliquent les conditions de Wolfe faibles. Effectivement (3.12) est équivalente à (3.10), tandis que pour le cas particulier $\sigma_1 = \sigma$ et $\sigma_2 = +\infty$, (3.13) est équivalente à (3.10). En effet :

$$\begin{aligned} \sigma_1 \nabla^T f(x_k) d_k \quad &\leq \quad \nabla^T f(x_k + \alpha_k d_k) d_k \leq -\sigma_2 \nabla^T f(x_k) d_k \\ &\Rightarrow \quad \sigma \nabla^T f(x_k) d_k \leq \nabla^T f(x_k + \alpha_k d_k) d_k \qquad \square \end{aligned}$$

Dans le théorème suivant on va assurer l'existence du pas de Wolfe en posant quelques conditions sur la fonction φ_k.

Théorème3.4([28])

Si $\varphi_k : R_+ \to R$;définie par $\varphi_k(\alpha) = f(x_k + \alpha d_k)$ est dérivable et bornée inférieurement, si d_k est une direction de descente en x_k et si $\rho \in]0, 1[\,, \sigma \in]\rho, 1[\,$, alors l'ensemble des pas vérifiant la règle de

Wolfe (faible) (3.8)-(3.9) est non vide.

Preuve

On a

$$\begin{aligned} \varphi_k(\alpha) &= f(x_k + \alpha d_k) \\ \psi_\rho(\alpha) &= f(x_k) + \rho\alpha\nabla^T f(x_k) d_k \end{aligned}$$

Le développement de Taylor-Yong en $\alpha = 0$ de φ_k est :
$\varphi_k(\alpha) = f(x_k + \alpha d_k) = f(x_k) + \rho\alpha\nabla^T f(x_k) d_k + \alpha\xi(\alpha)$ où $\xi(\alpha) \to 0, \alpha \to 0$:
et comme $\rho \in]0,1[$ et $\varphi'_k(0) = \nabla^T f(x_k) d_k < 0$ on déduit :
$f(x_k) + \alpha\nabla^T f(x_k) d_k < f(x_k) + \rho\alpha\nabla^T f(x_k) d_k$ pour $\alpha > 0$
On voit que pour $\alpha > 0$ assez petit on a :

$$\varphi_k(\alpha) < \psi_\rho(\alpha)$$

De ce qui précède et du fait que φ_k est bornée inférieurement, et $\psi_\rho(\alpha) \to -\infty$, $\alpha \to +\infty$; on déduit que la fonction $\psi_\rho(\alpha) - \varphi_k(\alpha)$ a la propriété :

$$\begin{cases} \psi_\rho(\alpha) - \varphi_k(\alpha) > 0 & \text{pour } \alpha \text{ assez petit} \\ \psi_\rho(\alpha) - \varphi_k(\alpha) < 0 & \text{pour } \alpha \text{ assez grand} \end{cases}$$

donc s'annule au moins une fois pour $\alpha > 0$:
En choisissant le plus petit de ces zéros on voit qu'il existe $\bar{\alpha} > 0$ tel que

$$\varphi_k(\bar{\alpha}) = \psi_\rho(\bar{\alpha}) \text{ et } \varphi_k(\alpha) \prec \psi_\rho(\alpha) \text{ pour } 0 < \alpha < \bar{\alpha} \qquad ((*))$$

La formule des accroissements ... nis fournit alors un nombre $\hat{\alpha}$; $0 < \hat{\alpha} < \bar{\alpha}$
tel que

$$\begin{aligned} \varphi_k(\bar{\alpha}) - \varphi_k(0) &= \bar{\alpha}\varphi'_k(\hat{\alpha}) = \bar{\alpha}\nabla^T f(x_k + \hat{\alpha} d_k) d_k \\ &\Rightarrow \quad \rho\bar{\alpha}\nabla^T f(x_k) d_k = \bar{\alpha}\nabla^T f(x_k + \hat{\alpha} d_k) d_k \\ &\Rightarrow \quad \nabla^T f(x_k + \hat{\alpha} d_k) d_k = \rho\nabla^T f(x_k) d_k \geq \sigma\nabla^T f(x_k) d_k \end{aligned}$$

car $0 < \rho < \sigma < 1$ et $\nabla^T f(x_k) d_k < 0$.
Donc $\hat{\alpha}$ satisfait (3.9)

D'autre part, $\alpha = \hat{\alpha}$ satisfait (3.8), en effet, $\hat{\alpha}$ satisfait (*) $\varphi_k(\hat{\alpha}) < \psi_\rho(\hat{\alpha})$ n'est autre que :

$$f(x_k + \hat{\alpha} d_k) = f(x_k) + \rho \hat{\alpha} \nabla^T f(x_k) d_k$$

Ce qu'il fallait démontrer. □

En pratique, on utilise des algorithmes spécifiques pour trouver un pas de Wolfe.

3.1.9 Algorithme de Fletcher-Lemaréchal

On va présenter un alogorithme simple, appelé de Algorithme de Fletcher-Lemaréchal, dont on peut monter qu'il trouve un pas de Wolfe en un nombre fini d'étapes.

Algorithme (Règle de Fletcher-Lemaréchal)

Etape0 : (initialisation)

$\alpha_{g,1} = 0$, $\alpha_{d,1} = +\infty$, $\rho \in]0,1[$, $\sigma \in]\rho, 1[$, $\upsilon \in \left]0, \frac{1}{2}\right[$, $\tau > 1$,choisir $\alpha_1 > 0$, poser $k = 1$ et aller à l'étape 1.

Etape 1 :

1.1 si $h_k(\alpha_k) > h_k(0) + \rho h'_k(0)\alpha_k$, alors

$\alpha_{d,k+1} = \alpha_k$,$\alpha_{g,k+1} = \alpha_{g,k}$ et aller à l'étape 2.

1.3 Si non $(h_k(\alpha_k) \leq h_k(0) + \rho h'_k(0)\alpha_k)$, alors

1.3.1 si $h'_k(\alpha) \geq \sigma h'_k(0)$: STOP($\alpha^* = \alpha_k$).

1.3.2 si non $(h'_k(\alpha) < \sigma h'_k(0))$

$\alpha_{d,k+1} = \alpha_{d,k}$,$\alpha_{g,k+1} = \alpha_k$ et aller à l'étape 2.

Etape 2 :

si $\alpha_{d,k+1} = +\infty$ déterminer $\alpha_{k+1} \in]\tau\alpha_{g,k+1}, +\infty[$

si non déterminer $\alpha_{k+1} \in](1-\upsilon)\alpha_{g,k+1} + \upsilon\alpha_{d,k+1}, \upsilon\alpha_{g,k+1} + (1-\upsilon)\alpha_{d,k+1}[$

poser $k = k + 1$ et aller à l'étape 1. □

Théorème3.5 ([28])

Si $h_k : R_+ \to R$;définie par $h_k(\alpha) = f(x_k + \alpha d_k)$ est dérivable et bornée inférieurement, si d_k est une direction de descente en x_k et si $\rho \in]0,1[$, $\sigma \in]\rho, 1[$, alors l'alogorithme de Fletcher-Lemaréchal trouve le pas de Wolfe(3.8)-(3.9) en un nombre fini d'étapes.

Le théorème suivant est important car il nous explicite les conditions suffisantes pour l'existence des scalaires $\bar{\lambda}_k$.

Théorème 3.6 [40]

Si d_k est une direction de descente, f est continument différentiable et bornée, alors il existe toujours un scalaire $\bar{\lambda}_k$ satisfait les conditions (3.8) et (3.9).

3.2 Théorèmes de convergence associés aux recherches linéaires inexactes

3.2.1 Algorithme général des directions de descente

L'algorithme général associé aux recherches linéaires inexactes et aux directions de descente est le suivant :

Algorithme des directions de descente

Etape1.

Prendre $x_0 \in R^n$ quelconque, $\varepsilon = 10^{-6}$, $k = 0$.

Etape2

Si $\|\nabla f(x_k)\| < \varepsilon$, stop ; sinon, trouvez une direction de descente $d_k \in R^n$ vérifiant

$$\nabla f(x_k)^T d_k < 0.$$

Etape3

Calculez le pas α_k en utilisant l'une des règles a) ou b) ou c) ou d) suivantes :

a) (3.6) et (3.7)

b) (3.8) ou (3.9)

c) (3.10) ou (3.11)

d) (3.12) et (3.13)

Etape4

Posez

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

Posez $k = k + 1$ et allez à Etape2

Théorème de Zoudentijk Dans cette section, on va étudier la *contribution* de la recherche linéaire inexacte dans la convergence des algorithmes à directions de descente. Ce n'est qu'une contribution, parce que la recherche linéaire ne peut à elle seule assurer la convergence des itérés. On comprend bien

que le choix de la direction de descente joue aussi un rôle. Cela se traduit par une condition, dite de *Zoutendijk*, dont on peut tirer quelques informations qualitatives intéressantes.

On dit qu'une règle de recherche linéaire satisfait la *condition de Zoutendijk* s'il existe une constante $c > 0$ telle que pour tout indice $k \geq 1$ on ait

$$f(x_{k+1}) \leq f(x_k) - c \left\|\nabla f(x_k)\right\|^2 cos^2\theta_k \tag{3.14}$$

où θ_k est l'angle que fait d_k avec $-\nabla f(x_k)$, défini par

$$cos\theta_k = \frac{-\nabla^\top f(x_k) d_k}{\left\|\nabla f(x_k)\right\| \left\|d_k\right\|} \tag{3.15}$$

Voici comment on se sert de la condition de Zoutendijk.

Proposition 3.1 *Si la suite $\{x_k\}$ générée par un algorithme d'optimisation vérifie la condition de Zoutendijk* (3.14) *et si la suite $\{f(\varkappa_k)\}$ est minorée, alors*

$$\sum_{k\geq 1} \left\|\nabla f(x_k)\right\|^2 cos^2\theta_k < \infty \tag{3.16}$$

Preuve. ([18])

En sommant les inégalités (3.14), on a

$$\sum_{k=1}^{l} \left\|\nabla f(x_k)\right\|^2 cos^2\theta_k \leq \frac{1}{c}\left(f\left(x_1\right) - f\left(x_{l+1}\right)\right)$$

et puisque la suite $\{f(x_k)\}$ est minorée, c'est-à-dire $\exists c^{'} > 0$ telle que pour tout k, $f(x_k) \geq c^{'}$ alors

$$\sum_{k\geq 1} \left\|\nabla f(x_k)\right\|^2 cos^2\theta_k < \infty \text{ quand } l \to \infty.$$

■

Les deux propositions suivantes précisent les circonstances dans lesquelles la condition de Zoutendijk (3.14) est vérifiée avec les règles d'Armijo et de Wolfe.

Proposition 3.2

Soit $f : R^n \longrightarrow R$ une fonction $C^{1,1}$ (différentiable et sa dérivée vérifie pour une constante L et pour tout x et $y : \left\|\nabla f\left(y\right) - \nabla f\left(x\right)\right\| \leq L\left\|y - x\right\|$) dans un voisinage de $\Gamma = \{x \in R^n : f(x) \leq f(x_1)\}$.

On considère un algorithme à directions de descente d_k ; qui génère une suite $\{x_k\}$ en utilisant la recherche linéaire d'Armijo avec $\lambda_1 > 0$.

Alors il existe une constante $c > 0$ telle que, pour tout $k \geq 1$, l'une des conditions

$$f(x_{k+1}) \leq f(x_k) - c\nabla^\top f(x_k)d_k \tag{3.17}$$

ou

$$f(x_{k+1}) \leq f(x_k) - c\left\|\nabla f(x_k)\right\|^2 cos^2\theta_k \tag{3.14}$$

est vérifiée.

Preuve. ([18])

Si le pas $\lambda_k = \lambda_1$ est accepté, on a (3.17), en effet

d_k direction de descente implique que $f(x_{k+1}) < f(x_k)$ et car λ_1 est uniformément positif

alors

$$f(x_{k+1}) \leq f(x_k) - c\nabla^\top f(x_k)d_k.$$

Dans le cas contraire ($\lambda_k \neq \lambda_1$), la condition d'Armijo (3.8) n'est pas vérifiée avec un pas $\lambda_k^{'} \leq \frac{\lambda_k}{\tau}$ c'est-à-dire $f(x_k + \lambda_k^{'}d_k) > f(x_k) + \rho\lambda_k^{'}\nabla^\top f(x_k)d_k$

Comme f est $C^{1,1}$, on a pour tout $\lambda_k > 0$:

$$\begin{aligned} f(x_k + \lambda_k d_k) &= f(x_k) + \lambda_k\nabla^\top f(x_k)d_k \\ &\quad + \int_0^1 \left[\nabla f(x_k + t\lambda_k d_k) - \nabla f(x_k)\right]^\top \lambda_k d_k dt \end{aligned}$$

$$\Rightarrow f(x_k + \lambda_k d_k) \leq f(x_k) + \lambda_k\nabla^\top f(x_k)d_k + c\lambda_k^2\left\|d_k\right\|^2$$

où $c > 0$ est une constante. Avec l'inégalité précédente, et le fait que ,

on obtient :

$$\begin{cases} f(x_k + \lambda_k^{'}d_k) - f(x_k) > \rho\lambda_k^{'}\nabla^\top f(x_k)d_k \\ f(x_k + \lambda_k^{'}d_k) - f(x_k) \leq \lambda_k^{'}\nabla^\top f(x_k)d_k + c\lambda_k^{'2}\left\|d_k\right\|^2 \end{cases}$$

$$\Rightarrow \rho\lambda_k^{'}\nabla^\top f(x_k)d_k \leq \lambda_k^{'}\nabla^\top f(x_k)d_k + c\lambda_k^{'2}\left\|d_k\right\|^2$$

$$\Rightarrow -c\lambda_k^{'2}\left\|d_k\right\|^2 \leq (1-\rho)\lambda_k^{'}\nabla^\top f(x_k)d_k$$

or

$$\rho < 1 \Rightarrow 0 < 1-\rho < 1 \Rightarrow \frac{1}{1-\rho} > 1$$

d'où

$$\nabla^\top f(x_k)d_k \geq \frac{-c}{1-\rho}\lambda_k^{'} \|d_k\|^2 \Rightarrow -\nabla^\top f(x_k)d_k \leq \frac{c}{1-\rho}\lambda_k^{'} \|d_k\|^2$$

$$\Rightarrow \left|\nabla^\top f(x_k)d_k\right| = \left\|\nabla^\top f(x_k)\right\| \|d_k\| cos\theta_k \leq \frac{c}{1-\rho}\lambda_k^{'} \|d_k\|^2$$

ce qui permet de minorer $\lambda_k^{'} \|d_k\|$ *et donc aussi* $\lambda_k \|d_k\|$ *par une constante fois* $\left\|\nabla^\top f(x_k)\right\| cos\theta_k$. *Cette minoration et l'expression suivante de la condition d'Armijo (3.8)*

$$f(x_k + \lambda_k d_k) \leq f(x_k) - \frac{\rho}{1-\rho} c\lambda_k \left\|\nabla^\top f(x_k)\right\| \|d_k\| cos\theta_k$$

car $\frac{\rho}{1-\rho} < 1$ *on obtient*

$$f(x_k + \lambda_k d_k) \leq f(x_k) - c\left\|\nabla^\top f(x_k)\right\|^2 cos^2\theta_k.$$

■

Proposition 3.3

Soit $f : R^n \longrightarrow R$ *une fonction* $C^{1,1}$ (*différentiable et sa dérivée vérifie pour une constante* c *et pour tout* x *et* $y : \|\nabla f(y) - \nabla f(x)\| \leq c\|y - x\|$) *dans un voisinage de*

$\Gamma = \{x \in R^n : f(x) \leq f(x_1)\}$.

On considère un algorithme à directions de descente d_k ; *qui génère une suite* $\{x_k\}$ *en utilisant la recherche linéaire de Wolfe* (3.8)-(3.9) *avec*

$\lambda_1 > 0$.

Alors il existe une constante $c > 0$ *telle que, pour tout* $k \geq 1$, *la condition de Zoutendijk* (3.14) *est vérifiée.*

Preuve. *D'après (3.8)*

$$\nabla^\top f(x_k + \lambda_k d_k)d_k \geq \sigma\nabla^\top f(x_k)d_k$$

$$\Rightarrow (\nabla f(x_k + \lambda_k d_k) - \nabla f(x_k))^\top d_k \geq (\sigma - 1)\nabla^\top f(x_k)d_k$$

$$= -(1-\sigma)\nabla^\top f(x_k)d_k = (1-\sigma)\left|\nabla^\top f(x_k)d_k\right|$$

$$\Leftrightarrow (1-\sigma)\left|\nabla^\top f(x_k)d_k\right| \leq \left(\nabla f(x_k+\lambda_k d_k)-\nabla f(x_k)\right)^\top d_k$$

et du fait que f est $C^{1,1}$:

$$\begin{aligned}(1-\sigma)\left|\nabla^\top f(x_k)d_k\right| &= (1-\sigma)\left\|\nabla^\top f(x_k)\right\| \|d_k\| cos\theta_k \\ &\leq \|\nabla f(x_k+\lambda_k d_k)-\nabla f(x_k)\| \|d_k\| \\ &\Rightarrow (1-\sigma)\left\|\nabla^\top f(x_k)\right\| cos\theta_k \leq L\lambda_k \|d_k\| \\ &\Rightarrow \lambda_k \|d_k\| \leq \frac{(1-\sigma)}{L}\left\|\nabla^\top f(x_k)\right\| cos\theta_k\end{aligned}$$

en utilisant), on aura :

$f(x_k+\lambda_k d_k) \leq f(x_k)+\rho\lambda_k \nabla^\top f(x_k)d_k$

$$\Rightarrow f(x_k+\lambda d_k) \leq f(x_k)+\rho\lambda\nabla^\top f(x_k)d_k \leq f(x_k)+\left|\rho\lambda\nabla^\top f(x_k)d_k\right|$$

$$\Rightarrow f(x_k+\lambda d_k) \leq f(x_k)+\rho\lambda\left|\nabla^\top f(x_k)d_k\right| \leq f(x_k)-\rho\lambda\nabla^\top f(x_k)d_k$$

$$\Rightarrow f(x_k+\lambda d_k) \leq f(x_k)-\rho\lambda\left\|\nabla^\top f(x_k)\right\| \|d_k\| cos\theta_k$$

$$\Rightarrow f(x_k+\lambda d_k) \leq f(x_k)-\frac{\rho\,(1-\sigma)}{L}\left\|\nabla^\top f(x_k)\right\|^2 cos^2\theta_k$$

On en déduit (3.14). ■

L'inégalité (3.14), que nous appelons la condition de Zoutendijk, implique

$$\left\|\nabla^\top f(x_k)\right\|^2 cos^2\theta_k \underset{k\to\infty}{\to} 0 \tag{3.18}$$

Cette limite peut être utilisée alternativement pour extraire des résultats de convergence globale pour les algorithmes de la recherche linéaire.

En effet si on s'assure dans le choix de la direction d_k que l'angle θ_k défini par (3.15) est tel que :

$$\exists\delta > 0 \;\; tel\ que \;\; \cos\theta_k \geq \delta \;\; pour\ tout\ k$$

alors en utilisant (3.18), on aura :

$$\lim_{k\to\infty} \|\nabla f(x_k)\| = 0$$

En d'autres termes, nous pouvons être sûrs que les normes du gradient $\|\nabla f(x_k)\|$ convergent vers

zéro, si les directions de descente ne sont pas trop proches de l'orthogonalité avec le gradient.

En particulier, une méthode à direction de descente d_k , d_k faisant un angle nul avec le gradient génère une suite x_k vérifiant

$$\lim_{k\to\infty} \|\nabla f(x_k)\| = 0,$$

à condition bien sur d'utiliser une recherche linéaire de Wolfe ou de Goldstein.

Chapitre 4

METHODE DE LA PLUS FORTE PENTE

4.1 Introduction

Cette méthode fut découverte par Cauchy en 1847. Il est naturel de se demander l'origine ou la justification d'une telle appelation (méthode de la plus forte pente). Considérons un point $x_k \in R^n$, si $\nabla f(x_k) \neq 0$, alors la direction $d_k = -\nabla f(x_k)$ est une direction de descente (voir Théorème 3.1 et remarque 3.1). Le Théorème qui suit va nous montrer que c'est en fait la meilleure direction de descente. En d'autres termes la décroissance de la fonction sera la plus forte en suivant la direction : $-\nabla f(x_k)$.

Théorème 4 .1 *Supposons que $f : R^n \to R$soit différentiable au point x, et supposons que $\nabla f(x) \neq 0$. Considérons le problème optimal*

$$\underset{\|d\|\leq 1}{Minimiser}\ f'(x,d)$$

où $f'(x,d)$ *est la dérivée directionnelle de f au point x et dans la direction d. Alors La solution optimale de ce problème est donnée par*

$$\tilde{d} = -\frac{\nabla f(x)}{\|\nabla f(x)\|}$$

Preuve

Puisque

$$f'(x,d) = \lim_{\lambda \to 0_+} \frac{f(x+\lambda d) - f(x)}{\lambda} = \nabla f(x)^t d.$$

Notre problème revient donc à minimiser $\nabla f(x)^t d$ dans $\{d : \|d\| \leq 1\}$.
L'inégalité de shwartz donne

$$\left|\nabla f(x)^t d\right| \leq \|\nabla f(x)\| \, \|d\| . \tag{4.1}$$

Si

$$\nabla f(x)^t d \geq 0,$$

on a bien sur

$$\nabla f(x)^t d \geq - \|\nabla f(x)\| \, \|d\| .$$

Si

$$\nabla f(x)^t d \leq 0,$$

(4.1) implique que

$$-\nabla f(x)^t d \leq \|\nabla f(x)\| \, \|d\| ,$$

Par conséquent on a toujours

$$\nabla f(x)^t d \geq - \|\nabla f(x)\| \, \|d\| .$$

Pour $\|d\| \leq 1$, on a

$$\|\nabla f(x)\| \, \|d\| \leq \|\nabla f(x)\| \Rightarrow - \|\nabla f(x)\| \, \|d\| \geq - \|\nabla f(x)\| .$$

Donc : $\forall d : \|d\| \leq 1$ on a

$$\nabla f(x)^t d \geq - \|\nabla f(x)\| .$$

D'autre part on a : $\left\|\tilde{d}\right\| = 1$ et $\tilde{d}$ vérifie :

$$\nabla f(x)^t \tilde{d} = \nabla f(x)^t (-\frac{\nabla f(x)}{\|\nabla f(x)\|}) = - \|\nabla f(x)\| . \ \blacksquare$$

Interprétation du théorème 5.1 : Nous allons à partir du théorème 4.1 donner une idée intuitive sur l'appelation : méthode de la plus forte pente. En effet d'après le théorème 4.1 on a :

$$f'(x,d) \geq f'(x,\tilde{d}) : \quad \forall d, \quad \|d\| \leq 1.$$

Soit en utilisant la définition de la dérivée directionnelle.

$$\lim_{\lambda \to 0_+} \frac{f(x+\lambda d) - f(x)}{\lambda} \geq \lim_{\lambda \to 0_+} \frac{f(x+\lambda \tilde{d}) - f(x)}{\lambda}.$$

Cette dernière inégalité implique qu'il existe $\delta > 0$ tel que

$$[f(x+\lambda d) - f(x)] - \left[f(x+\lambda \tilde{d}) - f(x)\right] \geq 0, \quad \forall \lambda \in]-\delta, +\delta[\,,$$

ou encore

$$f(x+\lambda d) \geq f(x+\lambda \tilde{d}), \quad \forall \lambda \in]-\delta, +\delta[\quad \text{et} \quad \forall d, \quad \|d\| \leq 1.$$

4.2 Algorithme de la méthode de la plus forte pente

Cet algorithme est tres simple. Il suit le schémas suivant.

Algorithme de la méthode de la plus forte pente

Etape intiale : Choisir un $\varepsilon > 0$. Choisir un point initial x_1. Poser $k = 1$ et aller à l'étape principale.

Etape principale : Si $\|\nabla f(x)\| < \varepsilon$ stop. Sinon poser $d_k = -\nabla f(x_k)$ et soit λ_k la solution optimale de la recherche linéaire

$$Min \ \{f(x_k + \lambda d_k); \ \lambda \geq 0\}.$$

Poser $x_{k+1} = x_k + \lambda_k d_k$. Remplacer k par $k+1$ et répéter l'étape principale.

4.3 Inconvénients de la méthode de la plus forte pente

4.3.1 Lenteur de la méthode au voisinage des points stationnaires

Cette méthode travaille de façon performante dans les premières étapes de l'algorithme. Malheuresement, dès qu'on s'approche du point stationnaire, la methode devient trés lente. On peut expliquer intuitivement ce phénomène par les considérations suivantes

$$f(x_k + \lambda d) = f(x_k) + \lambda \nabla f(x_k)^t d + \lambda \|d\| \alpha(x_k; \lambda d),$$

où $\alpha(x_k; \lambda d) \to 0$ quand $\lambda d \to 0$.
Si $d = -\nabla f(x_k)$, on obtient : $x_{k+1} = x_k - \lambda \nabla f(x_k)$ et par conséquent

$$f(x_{k+1}) - f(x_k) = \lambda \left[- \|\nabla f(x_k)\|^2 + \|\nabla f(x_k)\| \, \alpha(x_k; \lambda \nabla f(x_k)) \right].$$

D'après l'expression précedente, on voit que lorsque x_k s'approche d'un point stationnaire, et si f est continuement différentiable, alors $\|\nabla f(x_k)\|$ est proche de zéro. Donc le terme à droite s'approche de zéro, indépendemment de λ, et par conséquent $f(x_{k+1})$ ne s'éloigne pas beaucoup de $f(x_k)$ quand on passe du point x_k au point x_{k+1}.

4.3.2 Le phénomène de Zigzaguing

Il n'est pas facile de vérifier que pour la méthode du gradient on a toujours

$$d_k^t . d_{k+1} = 0,$$

c'est à dire que la suite $\{x_k\}$ engendrée par l'algorithme de la méthode du gradient, zigzague. Ceci crée un phénomène de ralentissement dans l'acheminement des points x_k vers la solution optimale.

4.4 Quelques remèdes

4.4.1 Changement de direction

Au lieu de prendre comme direction de descente, la direction :

$$d_k = -\nabla f(x_k),$$

on prend des directions de la forme

$$d_k = -D . \nabla f(x_k),$$

où D est une matrice choisie convenablement (D pourrait être par exemple l'inverse de la matrice héssienne au point x_k c'est à dire $(H(x_k))^{-1}$).
Un autre choix pourrait s'operer de la façon suivante :

$$d_k = -\nabla f(x_k) + g_k,$$

où g_k est un vecteur approprié.

4.4.2 Accéleration de la convergence

On peut aussi accelerer la convergence de la méthode du gradient. Pour celà on transforme grace à un algorithme d'acceleration de la convergence la suite $\{x_k\}$ en une suite $\{y_k\}$ qui convergerait vers la même limite que la suite $\{x_k\}$, mais convergerait plus rapidement. Si on note par x^* cette limite commune on exprime cette rapidité par la limite suivante :

$$\lim_{k \longrightarrow \infty} \frac{y_k - x^*}{x_k - x^*} = 0$$

4.5 Programme en fortran 90 de la méthode de la plus forte pente et tests numériques

4.5.1 Programme en fortran 90

On présente dans cette partie le programme en Fortran 90 de l'algorithme steepest descent avec la recherche linéaire d'Armijo. On utilise le choix des paramètres suivants :

$\varepsilon = 10^{-5}, mu = 0,2,\ be = 0,5$

Programme en fortran 90

```
program steepest
implicit none
integer, parameter : :n=1000
double precision yc(n),pk(n)
double precision eps,s,be,f1,f0,mu,f2
integer k,l,m,i
k=0 ;m=0
print*,'enter the initial point'
do i=1,n
yc(i)=1.
enddo
print*,'enter eps'
read*,eps
be=0.5 ;mu=0.2
do
if(sqrt(dot_product(df(yc),df(yc)))<=eps.or.k>=500)exit
```

```
pk=-df(yc)
f0=f(yc)
!recherche d'Armijo
l=1 ;s=1
f1=f(yc+s*pk)
if(f1<=f0+mu*s*dot_product(df(yc),pk))then
s=s/be
do
f2=f(yc+s*pk)
if(f2>f0+mu*s*dot_product(df(yc),pk))exit
s=s/be
f1=f2 ;l=l+1
enddo
s=s*be
yc=yc+s*pk
k=k+1
print*,s
else
s=s*be
do
f1=f(yc+s*pk)
if(f1<=f0+mu*s*dot_product(df(yc),pk))then
yc=yc+s*pk
k=k+1
print*,s
exit
endif
s=s*be
l=l+1
enddo
endif
m=m+l+1
f0=f1
enddo
```

```
print*,'LA SOLUTION EST'
print '(2e10.2)',yc
print*,'LA VALEUR DE LA FONCTION EST'
print '(e10.2)',f(yc)
print*,'LA NORME DU GRADIENT EST'
print '(e10.2)',sqrt(dot_product(df(yc),df(yc)))
print*,'LE NOMBRE D IERATIONS EST'
print '(i10)',k
contains
function f(u)
double precision f,u(n)
integer i1
f=0
do i1=1,n
f=f+i1*u(i1)**2
enddo
f=f**2
endfunction
function df(u)
double precision df(n),u(n),sm
integer i1
sm=0
do i1=1,n
sm=sm+i1*u(i1)**2
enddo
do i1=1,n
df(i1)=4*i1*u(i1)*sm
enddo
endfunction
end
```

4.5.2 Tests numériques

Nous allons tester l'algorithme "steepest descent" avec différentes recherches linéaires, comme fonction test on prend : la vallée - banane - de Rosenbrock. Il sagit d'une fonction à deux variables du

quatrième degré qui se présente comme une vallée en U dont le fond assez plat est incurvé suivant une parabole.

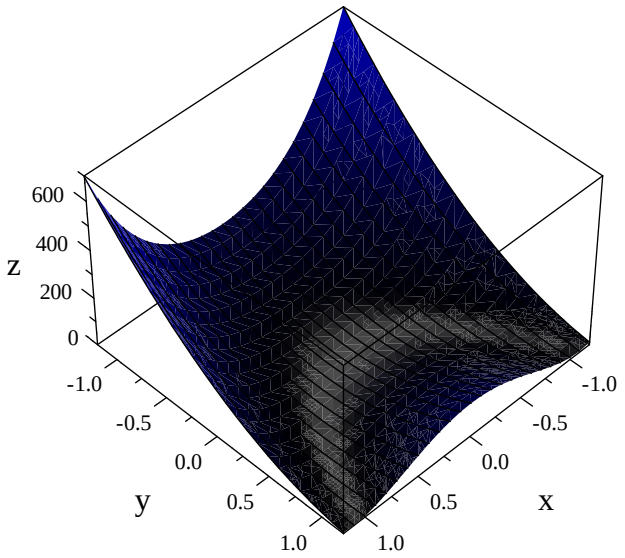


Figure 4.1 : La fonction de Rosenbrock ($f(x,y) = 100 \cdot (x^2 - y)^2 + (x-1)^2$)

Tableau 4.1 : Steepest descent avec recherche linéaire exacte (dichotomie)

k	t_k	x_k	$f(x_k)$	$\|g(x_k)\|$
1	–	$(-1.20, 1.000)$	24.200	232.867
10	4.777×10^{-03}	$(-1.009, 1.019)$	4.037	2.928
100	3.811×10^{-03}	$(-0.771, 0.596)$	3.138	3.092
200	8.999×10^{-03}	$(0.147, 0.018)$	0.727	1.642
500	1.634×10^{-03}	$(0.807, 0.650)$	3.726×10^{-02}	0.220
1000	1.243×10^{-03}	$(0.944, 0.891)$	3.095×10^{-03}	5.547×10^{-02}
1500	1.161×10^{-03}	$(0.981, 0.962)$	3.590×10^{-04}	1.826×10^{-02}
2000	1.136×10^{-03}	$(0.993, 0.986)$	4.538×10^{-05}	6.4229×10^{-03}

Tableau 4.2 : Steepest descent avec recherche linéaire d'Armijo

k	t_k	x_k	$f(x_k)$	$\|g(x_k)\|$
1	–	$(-1.200, 1.000)$	24.200	232.867
10	2.499×10^{-03}	$(-1.016, 1.035)$	4.067	3.239
100	4.999×10^{-03}	$(0.360, 0.123)$	0.413	1.399
200	2.499×10^{-03}	$(0.683, 0.466)$	0.100	0.409
500	2.499×10^{-03}	$(0.930, 0.865)$	4.841×10^{-03}	0.124
1000	1.250×10^{-03}	$(0.976, 0.953)$	5.501×10^{-04}	2.304×10^{-02}
1500	2.499×10^{-03}	$(0.988, 0.977)$	1.225×10^{-04}	1.390×10^{-02}
2000	4.999×10^{-03}	$(0.994, 0.989)$	2.838×10^{-05}	1.037×10^{-02}

Tableau 4.3 : Steepest descent avec recherche linéaire de Goldestein

k	t_k	x_k	$f(x_k)$	$\|g(x_k)\|$
1	–	$(-1.200, 1.000)$	24.200	232.867
10	9.024×10^{-03}	$(-0.939, 0.878)$	3.765	5.702
100	8.573×10^{-04}	$(-0.483, 0.248)$	2.221	2.956
200	8.573×10^{-04}	$(0.386, 0.149)$	0.376	1.221
500	8.573×10^{-04}	$(0.737, 0.543)$	6.894×10^{-02}	0.428
1000	9.024×10^{-03}	$(0.906, 0.821)$	8.749×10^{-03}	0.142
1500	8.573×10^{-04}	$(0.989, 0.978)$	1.117×10^{-04}	1.356×10^{-02}
2000	8.573×10^{-04}	$(0.998, 0.996)$	2.716×10^{-06}	1.478×10^{-03}

Tableau 4.4 : Steepest descent avec recherche linéaire de Wolfe

k	t_k	x_k	$f(x_k)$	$\|g(x_k)\|$
1	–	$(-1.200, 1.000)$	24.200	232.867
10	2.499×10^{-03}	$(-1.016, 1.03)$	4.067	3.239
100	4.999×10^{-03}	$(0.593, 0.347)$	0.166	0.762
200	2.499×10^{-03}	$(0.716, 0.511)$	8.059×10^{-02}	0.455
500	2.000×10^{-02}	$(0.927, 0.859)$	5.380×10^{-03}	0.332
1000	9.999×10^{-03}	$(0.975, 0.951)$	5.919×10^{-04}	8.178×10^{-02}
1500	9.999×10^{-03}	$(0.988, 0.977)$	1.279×10^{-04}	1.259×10^{-02}
2000	1.250×10^{-03}	$(0.994, 0.989)$	2.887×10^{-05}	6.108×10^{-03}

4.6 Convergence de la méthode de la plus forte pente

4.6.1 Cas des recherches linéaires exactes

avant de donner le théorème principal concernant la convergence de la méthode de la plus forte pente avec des recherches linéaires exactes et inexactes, démontrons d'abord ce résultat qui nous sera utile dans la section qui va suivre.

Théorème 4.2 *Soit $f : R^n \to R$ et $L \subset R$, un intervalle férmé et M l'application multivoque suivante :*

$$M : R^n \times R^n \longrightarrow R^n$$

$$(x, d) \to M(x, d) = \left\{ y : y = x + \overline{\lambda} d \right\}$$

$\overline{\lambda}$ vérifiant :

$$f(x + \overline{\lambda} d) = \min \ \{ f(x + \lambda d);\ \lambda \in L \}.$$

Si f est continue en x et si $d \neq 0$, alors M est férmée au point (x, d).

Preuve :

Soit $\{x_k, d_k\}_{k \in \mathbb{N}}$ et $\{y_k\}_{k \in \mathbb{N}}$ telles que

$$(x_k, d_k) \underset{k \to \infty}{\longrightarrow} (x, d),\ y_k \in M(x_k, d_k),\ y_k \underset{k \to \infty}{\longrightarrow} y. \tag{4.2}$$

Démontrons que

$$y \in M(x, d).$$

En effet $y_k \in M(x_k, d_k)$. Alors

$$y_k = x_k + \overline{\lambda_k}.d_k\ , \tag{4.3}$$

$\overline{\lambda_k}$ est solution optimale de la recherche linéaire

$$f(x_k + \overline{\lambda_k}.d_k\ ) = \min \ \{ f(x_k + \lambda.d_k\ );\ \lambda \in L \}, \tag{4.4}$$

Puisque

$$d_k \underset{k \to \infty}{\longrightarrow} d \neq 0,$$

alors

$$d_k \neq 0, \quad \forall k \geq k_0, \quad k_0 \in \mathbb{N}. \tag{4.5}$$

(4.3) et (4.5) impliquent

$$\overline{\lambda_k}. = \frac{\|y_k - x_k\|}{\|d_k\ \|}, \tag{4.6}$$

soit en remarquant que L est fermé, on a

$$\overline{\lambda_k} \underset{k\to\infty}{\longrightarrow} \overline{\lambda} = \frac{\|y - x\|}{\|d\ \|} \in L \tag{(4.7)}$$

(4.2), (4.3), (4.6) et (4.7) donnent

$$y = x + \overline{\lambda}.d,$$

(4.4) implique

$$f(x_k + \overline{\lambda_k}.d_k\ ) \leq f(x_k + \lambda.d_k\ ).$$

Faisons tendre k vers l'infini et prenons en considération le fait que f est continue au pount (x, d), on obtient :

$$f\left(x + \overline{\lambda}.d\right) \leq f\left(x + \lambda.d\right), \quad \forall \lambda \in L.$$

Ceci veut dire exactement que le point $y = x + \overline{\lambda}.d \in M(x, d)$. ■

Théorème 4.3 ([28]) *(convergence de la méthode de la plus forte pente avec des recherches linéaires exactes) Soit $f : R^n \to R$, telle que $f \in C^1(R^n)$ et (P) le problème de minimisation sans contraintes suivant :*

$$(\ P\ ) \qquad \min\left\{f(x) : x \in R^n\right\}.$$

On suppose que l'ensemble $\delta(x_0)$ suivant

$$\delta(x_0) = \left\{x \in R^n :\ f(x) \leq f(x_0)\right\}, \quad x_0 \in R^n,$$

est borné. Soit $\{x_k\}$ une suite générée par l'algorithme de la plus forte pente, c'est à dire

Initialisation : *$x_0 \in R^n$, point initial, poser $k = 0$ et aller à Etape principale*

Etape principale : *Si $\nabla f(x_k) = 0$ Stop.*

Sinon

Calculer λ_k vérifiant $f(x_k + \lambda_k d_k) = \min\ \{f(x_k + \lambda d_k);\ \lambda \geq 0\}$

Calculer $x_{k+1} = x_k - \lambda_k \nabla f(x_k)$

Poser $k = k + 1$ *et aller à Etape principale.*

Fin

Soit x^* *une limite quelconque d'une sous suite convergente de* $\{x_k\}$. *Alors* $\nabla f(x^*) = 0$.

4.6.2 Cas des recherches linéaires inexactes

Théorème 4.4([Bazara]) *(convergence de la méthode de la plus forte pente avec la recherche linéaire d'Armijo). Soit* $f : R^n \to R$, *telle que* $\nabla f(x)$ *est continuement Lipschitzien de constante* G *dans l'ensemble* $\delta(x_0) = \{x \in R^n : \ f(x) \leq f(x_0)\}$, $x_0 \in R^n$ *quelconque. Considérons l'algorithme de la méthode de la plus forte pente avec la recherche linéaire d'Armijo, c'est à dire l'algorithme défini comme suit :*

Initialisation

Fixons $\overline{\lambda} > 0$, $0 < \varepsilon < 1$ et $x_0 \in \mathbb{R}^n$ *point initial, poser* $k = 0$ *et aller à Etape principale.*

Etape principale

A l'iteration k *définissons la direction* $d_k = -\nabla f(x_k)$ *et considérons la fonction d'Armijo*

$$\widehat{\theta}(\lambda) = \theta(0) + \lambda\varepsilon\theta'(0) \tag{4.8}$$

où $\theta(\lambda)$ *est la fonction suivante :*

$$\theta(\lambda) = f(x_k + \lambda d_k) = f[x_k - \lambda\nabla f(x_k)] : \quad \lambda \geq 0. \tag{4.9}$$

Si $\nabla f(x_k)$ *=0 stop.*

Sinon

Trouver le plus petit entier $t \geq 0$ *tel que*

$$\theta\left(\frac{\overline{\lambda}}{2^t}\right) \leq \widehat{\theta}\left(\frac{\overline{\lambda}}{2^t}\right) \tag{4.10}$$

et définir le successeur x_{k+1} *de* x_k *comme suit*

$$x_{k+1} = x_k - \lambda_k\nabla f(x_k),$$

avec

$$\lambda_k = \frac{\overline{\lambda}}{2^t}$$

Poser $k = k + 1$ *et aller à Etape principale.*

Fin

Soit $\{x_k\}$ *une suite générée par cet algorithme. Alors ou bien il s'arrete après un nombre fini d'itérations eu n un point* x_{k_0} *tel que* $\nabla f(x_{k_0}) = 0$, *ou bien il genere une suite infinie* $\{x_k\}_{k\in\mathbb{N}}$ *telle que*

$$\nabla f(x_k) \underset{k\to\infty}{\longrightarrow} 0.$$

4.7 Acceleration de la convergence de la méthode de la plus forte pente

Nous avons exposé dans la section 4.3 que la méthode de la plus forte pente, malgé sa simplicité, présente l'inconvénient d'être très lente au voisinage des points stationnaires. Pour y remédier à cet inconvenient nous essayerons dans cette section, qui constitue l'une des parties originales de cette thèse, d'accélérer la convergence de la méthode du gradient. Pour celà nous présenterons deux algorithmes, l'un basé sur l'ε-algorithme, version Wynn et l'autre basé sur l'ε-algorithme, version F.Cordellier.

Nous montrerons que le second algorithme (version F. Cordellier) est plus avantageux du point de vue numérique que le premier (version P.Wynn).

Nous avons appelé nos deux algorithmes l'epsilon steepest descent version Wynn et l'epsilon steepest descent version Cordellier.

L'epsilon steepest descent version Wynn

Pour les notions concernant les méthodes d'accélération de la convergence, voir le chapitre 1. Etant donnée une suite $\{s_n\}_{n\in\mathbb{N}}$; Wynn ([37]) a montré, que les quantité $\varepsilon_2^{(n)}$ peuvent être calculées de la façon suivante :

$$\varepsilon_2^{(n)} = \frac{s_n s_{n+2} - (s_{n+1})^2}{s_{n+2} - 2s_{n+1} + s_n}; \quad n = 0, 1, 2, ... \tag{4.11}$$

On considère donc ici la deuxième colonne paire du tableau de l'$\varepsilon-$algorithme scalaire, associée a la suite $\{s_n\}_{n\in\mathbb{N}}$

Remarque

Pour calculer la quantité $\varepsilon_2^{(n)}$;il suffit d'avoir les éléments : s_n, s_{n+1} et de la suite $\{s_n\}_{n\in\mathbb{N}}$

L'algorithme epsilon steepest descent version Wynn

Initialisation : L'algorithme commence par un point initial $x_0 \in R^n$; les composantes de x_0 seront notées comme suit :

$$x_0 = \left(x_0^1, x_0^2, ..., x_0^i, ...x_0^n\right),$$

poser $k = 0$ et aller à l'étape principale

Etape principale : Supposons qu'à l'étape k on ait le point x_k ;

$$x_k = \left(x_k^1, x_k^2, ..., x_k^i, ...x_k^n\right),$$

si $\|\nabla f(x_k)\| = 0$ stop ; sinon poser

$$r_k = x_k$$

$$r_k = \left(r_k^1, r_k^2, ..., r_k^i, ...r_k^n\right),$$

Calculer par la méthode steepest descent les successeurs s_k et t_k de r_k ; c'est à dire

$$s_k = \left(s_k^1, s_k^2, ..., s_k^i, ...s_k^n\right),$$

$$t_k = \left(t_k^1, t_k^2, ..., t_k^i, ...t_k^n\right),$$

$$s_k = t_k r_k - \lambda_k \nabla f(r_k),$$

λ_k solution optimale de la recherche linéaire exacte

$$\underset{\lambda\geq 0}{Minimiser} f(r_k - \lambda \nabla f(r_k))$$

$$t_k = s_k - \beta_k \nabla f\left(s_k\right),$$

$\beta_k$solution optimale de la recherche linéaire exacte

$$\underset{\beta \geq 0}{Minimiser} f\left(s_k \ - \lambda \ \nabla \ f\left(s_k\right)\right)$$

Calculer

$$t_k^i - 2s_k^i + r_k^i; i \ = 1, ...n$$

Si

$$t_k^i - 2s_k^i + r_k^i \ \neq \ 0 \ ; i \ = 1, ...n$$

calculer

$$\varepsilon_2^k = \left(\varepsilon_2^{k,1}, \varepsilon_2^{k,2}, ..., \varepsilon_2^{k,i}, ...\varepsilon_2^{k,n}\right),$$

avec

$$\varepsilon_2^{k,i} \ = \ \frac{r_k^i \ t_k^i \ - \ \left(s_k^i\right)^2}{t_k^i \ - 2 \ s_k^i \ + r_k^i \ ,} \quad i = 1, 2, ...n.$$

Si $f(\varepsilon_2^k) < f(t_k)$ poser

$$x_k \ = \varepsilon_2^k$$

Remplacer k par $k+1$ et aller l'étape principale.
Si $f(\varepsilon_2^k) \geq f(t_k)$ ou si $t_k^{i_0} \ -2s_k^{i_0} \ +r_k^{i_0} = 0; i_0 \in \{1, ...n\}$,poser

$$x_k = t_k$$

Remplacer$k$ par $k+1$ et aller à l'étape principale.

L'epsilon steepest descent algorithme version F. Cordellier.

Etant donnée une suite$\{s_n\}_{n\in\mathbb{N}}$; Cordellier ([11]) propose une autre formule pour l'epsilon algorithme d'ordre 2. Les quantites $\varepsilon_2^{(n)}$ peuvent être calculées de la façon suivante :

$$\varepsilon_2^{(n)} = s_{n+1} + \left[\frac{1}{s_{n+2} - s_{n+1}} - \frac{1}{s_{n+1} - s_n}\right]^{-1} \tag{4.12}$$

Les calculs numériques ont montré que l'epsilon algorithme d'ordre 2 avec la formule (4.12) est stable alors qu'avec la formule (4.11) il ne l'est pas. Cela est dû au fait que la formule (4.11) est un rapport de quantités petites et mal calculées à cause des erreurs de cancellation.

L'epsilon steepest descent algorithme version F. Cordellier.

Initialisation : L'Algorithme commence par un point initial $x_0 \in R^n$; les composantes de x_0 seront notées comme suit :

$$x_0 = \left(x_0^1, x_0^2, ..., x_0^i, ...x_0^n\right),$$

poser $k = 0$ et aller à l'étape principale.

Etape principale : Supposons qu'à l'étape k on ait le point x_k ;

$$x_k = \left(x_k^1, x_k^2, ..., x_k^i, ...x_k^n\right),$$

si $\|\nabla\ f\ (x_k)\| = 0$ stop ; sinon poser

$$r_k\ =\ x_k,$$

$$r_k\ =\ \left(r_k^1, r_k^2, ..., r_k^i, ...r_k^n\right),$$

Calculer par la méthode de steepest descent les successeurs s_k et t_k de ; c'est à dire

$$s_k = t_k r_k - \ \lambda_k\ \nabla\ f\ (r_k),$$

$$t_k = \left(t_k^1, t_k^2, ..., t_k^i, ...t_k^n\right),$$

$$s_k = t_k r_k - \ \lambda_k \ \nabla \ f \ \ (r_k)\,,$$

λ_k solution optimale de la recherche linéaire exacte

$$\underset{\lambda \geq 0}{Minimiser} f\left(r_k \ - \lambda \ \nabla \ f\left(r_k\right)\right)$$

$$t_k = s_k - \beta_k \nabla f\left(s_k\right),$$

β_k solution optimale de la recherche linéaire exacte

$$\underset{\beta \geq 0}{Minimiser} f\left(s_k \ - \beta \nabla \ f\left(s_k\right)\right)$$

Calculer

$$s_k^i \ - r_k^i \ ; t_k^i - \ s_k^i \ ; \ \frac{1}{t_k^i \ - s_k^i} \ - \frac{1}{s_k^i \ - r_k^i} \ , i = 1, ..., n$$

Si

$$s_k^i - r_k^i \ \neq \ 0 \ ; t_k^i - s_k^i \ \neq 0 \text{ et } \frac{1}{t_k^i \ - s_k^i} - \ \frac{1}{s_k^i \ - r_k^i} \neq \ 0$$

Calculer

$$\varepsilon_2^{k,i} \ = s_k^i \ + \ \frac{1}{\frac{1}{t_k^i \ - s_k^i} \ - \frac{1}{s_k^i \ - r_k^i}} \ , \ i = 1, ...n.$$

Si $f(\varepsilon_2^k) < f(t_k)$

Poser $x_k = \varepsilon_2^k$,remplacer k par $k+1$ et aller à l'étape 1.

Si $f(\varepsilon_2^k) \geq f(t_k)$ ou $s_k^{i0} \ - r_k^{i0} = 0$ ou $t_k^{i0} \ - s_k^{i0} = 0$ ou $\frac{1}{t_k^{i0} \ - s_k^{i0}} \ - \frac{1}{s_k^{i0} \ - r_k^{i0}} = 0$,$i_0 \in \{1, ..., n\}$

Poser $x_k = t_k$;remplacer k par $k+1$ et aller à l'étape 1.

Convergence de l'epsilon steepest descent algorithme :

cas des recherches linéaires exactes

Maintenant nous sommes en mesure de donner et de démontrer le théorème de convergence de l'epsilonsteepest descent algorithme. La démonstration de ce théorème est la même pour les deux versions de l'epsilon steepest descent algorithme.

Théorème 4.5

Soit $f : R^n \longrightarrow R$ telle que $f \in C^1(R^n)$ et (p) le problème d'optimisation sans contraintes suivant :

$$(p) \qquad min\{f\ (x) : x \in R^n\}$$

.

On suppose que l'ensemble $\delta(x_0) = \{x \in R^n : f\ (x) \leq f\ (x_0), x_0 \in R^n\}$ est borné. Soit $\{x_n\}_{n\in\mathbb{N}}$ la suite générée par l'epsilon steepest descent algorithme. Soit x une limite d'une sous-suite convergente de $\{x_n\}_{n\in\mathbb{N}}$ alors $\nabla f\ (x^*) = 0$

Démonstration :

Supposons que l'algorithme génère une suite infinie$\{x_n\}_{n\in\mathbb{N}}$; car dans le cas contraire, il s'arrêterait dans un nombre fini d'itérations en un point x tel que$\nabla f\ (x) = 0$; et le problème serait résolu. Soit $\{x_n\}_{n\in\mathbb{N}_1}, N_1 \subset N$, une sous suite convergente de $\{x_n\}_{n\in\mathbb{N}}$ x^*. de limite x^*. Montrons que $x^* \in \Omega$, avec

$\Omega = \{x \in R^n : \nabla f\ (x) = 0\}$

Supposons le contraire, c'est à dire que $x^* \notin \Omega$, La suite $\{x_n\}_{n\in\mathbb{N}}$ pourrait être definie comme suit, x_0 étant un point initial, si x_n est connu à l'itération n, x_{n+1} sera défini comme suit :

$$x_{n+1} \in A(x_n)$$

A étant une fonction multivoque définie de la façon suivante :

$$A = C \circ B = C \circ B_2 \circ B_1$$

avec

$$B_1 \quad : R^n \longrightarrow R^n \times R^n$$

$$x \longrightarrow B_1(x) = (x_1\ x - \ \lambda_x\ \nabla\ f(x))\,;$$

$$\lambda_x \text{verifiant} \quad : \ f(x - \lambda_x\ \nabla\ f\ \ (x)) \leq f(x - \lambda_x\ \nabla\ f\ \ (x)) : \lambda \geq 0$$

$$B_2 \quad : R^n \times R^n \longrightarrow R^n \times R^n \times R^n$$

$$(x, y)\ \longrightarrow B_2(x, y) = (x, y, y - \ \lambda_y\ \nabla f\ \ (y))$$

$$\lambda_y \text{verifiant} : \quad f(y - \lambda_y \nabla\ f(y)) \leq \quad f(y\ -\ \lambda \nabla\ f(y)) : \lambda \geq 0$$

$$C \quad : R^n \times R^n \times R^n \longrightarrow R^n$$

$$(x, y, z)\ \longrightarrow C(x, y, z) = \omega = \left(\omega^1, ..., \omega^n\right)$$

avec

$$\omega^i \ = \begin{cases} \frac{x^i\ z^i - \left(y^i\right)^2}{z^i - 2y^i + x^i}, & \text{si} \ \ z^i - 2y^i\ + x^i \neq 0 \\ z^i, & \text{sinon} \end{cases}$$

$$x = \left(x^1, ..., x^n\right),\ y = \left(y^1, ..., y^n\right), z = \left(z^1, ..., z^n\right)$$

dans le cas de la formule de Wynn et :

$$\omega^i \ = \begin{cases} y^i\ + \frac{1}{\frac{1}{z^i - y^i} - \frac{1}{y^i - x^i}}, & \text{si} \ \ z^i - y^i \neq 0\,; y^i - x^i \neq 0 \text{ et } \frac{1}{z^i - y^i} - \frac{1}{y^i - x^i} \neq 0 \\ z^i, & \text{sinon} \end{cases}$$

$$x = \left(x^1, ..., x^n\right),\ y = \left(y^1, ..., y^n\right), z = \left(z^1, ..., z^n\right)$$

dans le cas de la formule de Cordellier.

L'application multivoque B est fermée en tout point $x \notin \Omega$. D'autre part f étant continue, alors

$$\lim_{n\to\infty, n\in\mathbb{N}_1} f(x_n) = f(x^*)$$

Montrons que

$$\lim_{n\to\infty.} f(x_n) = f(x^*)$$

En effet, $\varepsilon > 0$ donné, il existe $n_0 \in N_1$ tel que

$$|f(x_n) \ - f(x^*)| = f(x_n) - f(x^*) \prec \varepsilon, pour\ n \geq n_0,\ n \in N_1.$$

En particulier pour $n = n_0$

$$f(x_{n0}) - f(x^*) \prec \varepsilon$$

Soit maintenant $n > n_0$; $n \in N$. Il est facile de voir que par construction, la suite $\{f(x_n)\}_{n\in\mathbb{N}}$ est strictement décroissante. Donc

$$f(x_n) \succ f(x_{n0})$$

$$f(x_n) - f(x^*) = f(x_n) - f(x_{n0}) + f(x_{n0}) - f(x^*) \prec 0 + \varepsilon = \varepsilon.$$

Ceci implique que

$$\lim_{n\to\infty.} f(x_n) = f(x^*) \tag{4.13}$$

Considérons maintenant la suite $\{(x_{n+1})\}_{n\in\mathbb{N}_1.}$ D'après l'algorithme et les définitions données aux fonctions multivoques A, C, B, B_1et B_2 nous avons

$$x_{n+1} = C\left(x_n, y_{n,}, z_n\right), \quad \left(x_n, y_{n,}, z_n\right) \in B\left(x_n\right), \tag{4.14}$$

(dans l'algorithme nous avons $x_n = r_n$; $y_n = s_n$; $z_n = t_n$). Remarquons que les suite$\{x_n\}, \{y_n\}, \{z_n\}$, et$\{x_{n+1}\}$, appartiennent à l'ensemble compact $\delta\left(x_0\right)$: Il existe alors $N_2 \subset N_1$ tels que

$$x_n \underset{n\to\infty\ ,n\in\ \mathbb{N}_2}{\to} x^* \tag{4.15}$$

$$y_n \underset{n\to\infty\ ,n\in\ \mathbb{N}_2}{\to} y^* \tag{4.16}$$

$$z_n \underset{n\to\infty\ ,n\in\ \mathbb{N}_2}{\to} z^* \tag{4.17}$$

$$x_{n+1} \underset{n\to\infty\ ,n\in\ \mathbb{N}_2}{\to} \hat{x}$$

Rappelons que B est fermée au point x^* (que nous avions supposé ne pas appartenir à Ω). Les relations

(4.14),(4.15),(4.16),(4.17) et la définition de la fermeture de B au point x donnent

$$\left(x^*, y^*, z^*\right) \in B\left(x^*\right),$$

Puisque $x^* \notin \Omega\ \left(\nabla\ f\left(x^*\right) \neq\ 0\right)$, alors la direction $-\nabla f\ \left(x^*\right)$ est une direction de descente. Les définitions des fonctions multivoques B ;B_1 et B_2 impliquent que

$$f\left(y^*\right) \prec f\left(x^*\right) \tag{4.18}$$

$$f\left(z^*\right) \prec f\left(x^*\right) \tag{4.19}$$

Maintenant, étant donné que

$$x_{n+1} = C\left(x_n, y_{n,}, z_n\right),$$

alors la construction même de l'algorithme donne

$$f(x_{n+1}) \leq f(x_n)$$

$$f(x_{n+1}) \leq f(y_n)$$

$$f(x_{n+1}) \leq f(z_n)$$

Soit en passant à la limite quand $n \to \infty$, $n \in N_2$

$$f(\hat{x}) \leq f(x^*)$$

$$f(\hat{x}) \leq f(y^*)$$

$$f(\hat{x}) \leq f(z^*)$$

Les relations (4.18) et (4.19) impliquent

$$f(\hat{x}) \prec f(x^*) \tag{4.20}$$

Puisque

$$f(x_{n+1}) \underset{n \to \infty \, , n \in \mathbb{N}_2}{\to} f(\hat{x})$$

et

$$f(x_n) \underset{n \to \infty \, , n \in \mathbb{N}}{\to} f(x^*)$$

alors on aurait du avoir

$$f(\hat{x}) = f(x^*) \tag{4.21}$$

relation (4.21) est en contradiction avec la relation (4.20). Donc $x^* \in \Omega$; ou encore $\nabla f(x^*) = 0$:

Convergence de l'epsilon steepest descent algorithme : cas des recherches linéaires inexactes

Théorème 4.6 (convergence de l'ε- steepest descent algorithme avec la recherche linéaire d'Armijo).

Soit $f : R^n \to R$, telle que $\nabla f(x)$ est continuement Lipschitzien de constante G dans l'ensemble $\delta(x_0)$ $=\{x \in R^n : f(x) \leq f(x_0)\}$, $x_0 \in R^n$quelconque. Considérons l'algorithme de la méthode "steepest descent avec la recherche linéaire inexacte d'Armijoo. Soit $\{x_k\}$ une suite générée par cet algorithme. Alors ou bien il s'arrête après un nombre fini d'itérations en un point x_{k0} tel que $\nabla f(x_{k0}) = 0$ ou bien il génère une suite infinie$\{x_k\}_{k\in\mathbb{N}}$ telle que

$$\nabla f(x_k) \underset{k\to\infty}{\to} 0$$

Démonstration :

Supposons que l'ε-steepest descent algorithme génère une suite $\{x_k\}_{k\in\mathbb{N}}$ infinie. A l'étape principale de l'algorithme deux recherches linéaires inéxactes sont nécessaires pour déterminer les deux successeurs $s_k$et t_k de x_k qui permettent par la suite de trouver x_{k+1} : On considère pour les deux recherches linéaires inexactes la recherche linéaire inexacte d'armijo.

Pour la première recherche linéaire, la règle d'Armijo est :

$$\theta\left(\frac{\bar{\lambda}}{2^{t_1}}\right) \leq \hat{\theta}\left(\frac{\bar{\lambda}}{2^{t_1}}\right)$$

avec

$$\hat{\theta}(\lambda) = \theta(0) + \lambda m\theta'(0)$$

et

$$\theta(\lambda) = f(x_k \ + \lambda d_k), \lambda \geq 0$$

Ces relations sont équivalentes à :

$$\theta\left(\frac{\bar{\lambda}}{2^{t_1}}\right) = f\left(x_k \ + \frac{\bar{\lambda}}{2^{t_1}} d_k\right) = f(s_k)$$

$$\begin{aligned} &\leq \quad \theta\left(\frac{\bar{\lambda}}{2^{t_1}}\right) = \theta(0) + \frac{\bar{\lambda}}{2^{t_1}} m\theta'(0) \\ &= \quad f(x_k) - \frac{\bar{\lambda}}{2^{t_1}} m \left\| \nabla f(x_k) \right\|^2 \end{aligned}$$

On a alors :

$$f(s_k) - f(x_k) \leq -\frac{\bar{\lambda}}{2^{t_1}} m \left\| \nabla f(x_k) \right\|^2$$

D'autre part, en développant f au voisinage du point x_k ; on obtient

$$f(s_k) - f(x_k) = (s_k - x_k) \nabla f(\tilde{x}) ; \tilde{x} = \alpha x_k + (1-\alpha) s_k ; \alpha \in]0 \ , 1[\, ,$$

ou encore en remarquant que $\quad s_k = x_k - \lambda_k \ \nabla f(x_k)$

$$\begin{aligned} f(s_k) - f(x_k) &= -\lambda_k \nabla f(x_k)^t \nabla f(\tilde{x}) ; \tilde{x} = \alpha x_k + (1-\alpha) s_k ; \alpha \in]0 \ , 1[\\ &= -\lambda_k \nabla f(x_k)^t \left[\nabla f(x_k) - \nabla f(x_k) + \nabla f(\tilde{x})\right] ; \tilde{x} = \alpha x_k + (1-\alpha) s_k ; \alpha \in]0 \ , 1[\\ &= -\lambda_k \left\| \nabla f(x_k) \right\|^2 + \lambda_k \nabla f(x_k)^t \left[\nabla f(x_k) - \nabla f(\tilde{x})\right] ; \tilde{x} = \alpha x_k + (1-\alpha) s_k ; \alpha \in]0 \ , 1[\end{aligned}$$

soit en utilisant l'inégalité de Cauchy Schwartz et le fait que le gradient est lipschitzien de constante G,

$$\begin{aligned} f(s_k) - f(x_k) &\leq -\lambda_k \|\nabla f(x_k)\|^2 + \lambda_k \|\nabla f(x_k)\| . \|\nabla f(x_k) - \nabla f(\tilde{x})\| \\ &\leq -\lambda_k \|\nabla f(x_k)\|^2 + \lambda_k \|\nabla f(x_k)\| .G \|x_k - \tilde{x}\| \\ &\leq -\lambda_k \|\nabla f(x_k)\|^2 + \lambda_k \|\nabla f(x_k)\| .G \|x_{k+1} - x_k\| \\ &\leq -\lambda_k \|\nabla f(x_k)\|^2 + \lambda_k \|\nabla f(x_k)\| .\lambda_k G \|\nabla f(x_k)\| \\ &\leq -\lambda_k \|\nabla f(x_k)\|^2 [1 - \lambda_k .G] \end{aligned}$$

Finalement

$$f(s_k) - f(x_k) \leq -\frac{\bar{\lambda}}{2^{t_1}} \|\nabla f(x_k)\|^2 \left[1 - \frac{\bar{\lambda}}{2^{t_1}} .G\right]. \tag{4.22}$$

Choisissons maintenant le plus petit entier t_1 de sorte que

$$\frac{\bar{\lambda}}{2^{t_1}} \|\nabla f(x_k)\|^2 \left[1 - \frac{\bar{\lambda}}{2^{t_1}} .G\right] \geq \frac{\bar{\lambda}}{2^{t_1}} m \|\nabla f(x_k)\|^2,$$

c'est à dire que t_1 vérifie en même temps

$$1 - \frac{\bar{\lambda}}{2^{t_1}} .G \geq m \tag{4.23}$$

et

$$1 - \frac{\bar{\lambda}}{2^{t_1}} .G \prec m \tag{4.24}$$

Ceci implique que

$$\frac{\bar{\lambda}}{2^{t_1}} m \succ \frac{m(m-1)}{2G}. \tag{4.25}$$

Bien sur n'oublions pas qu'avec ce choix et avec (4.23) on a aussi (4.22), qui donne en prenant en considération (4.26)

$$f(s_k) - f(x_k) \prec -\frac{m(m-1)}{2G} \|\nabla f(x_k)\|^2. \tag{4.26}$$

Posons

$$C = \frac{m\,(m-1)}{2G}$$

$C \prec 0$ on a alors

$$f\,(s_k) - f\,(x_k) \prec C.\,\|\nabla f\,(x_k)\,\|^2\,. \tag{4.27}$$

En procédant de la même manière pour la deuxième recherche linéaire inexacte d'Armijo on obtient :

$$f\,(t_k) - f\,(s_k) \prec C.\,\|\nabla f\,(s_k)\,\|^2\,. \tag{4.28}$$

Montrons que

$$\lim_{n\to\infty.} \|\nabla f\,(x_k)\,\| = 0$$
.

Pour cela, considérons

$$\begin{aligned} f\,(x_{k+1}) - f\,(x_k) &= f\,(\varepsilon_k) - f\,(t_k) + f\,(t_k) - f\,(s_k) + f\,(s_k) - f\,(x_k) \\ &\prec f\,(t_k) - f\,(s_k) + f\,(s_k) - f\,(x_k)\,. \end{aligned} \tag{4.29}$$

car

$$f\,(\varepsilon_k) - f\,(t_k) \prec 0$$

D'après les relations (4.28) et (4.29) on a :

$$\begin{aligned} f\,(x_{k+1}) - f\,(x_k) &\prec C.\,\|\nabla f\,(s_k)\,\|^2 + C.\,\|\nabla f\,(s_k)\,\|^2 \\ &= C\left[\|\nabla f\,(s_k)\,\|^2 + \|\nabla f\,(s_k)\,\|^2\right] \end{aligned}$$

La suite $\{f\,(x_k)\}_{k\in\mathbb{N}}$ est une suite strictement décroissante, minorée (sinon $inf\ f\,(x) = -\infty$)), donc convergente. Par passage à $\overline{\lim}_{k\to\infty}$, on obtient :

$$0 \leq C.\overline{\lim_{k\to\infty}}\left[\|\nabla f\left(s_k\right)\|^2+\|\nabla f\left(x_k\right)\|^2\right]$$

ou encore

$$0 \leq C.\left[\overline{\lim_{k\to\infty}}\|\nabla f\left(s_k\right)\|^2+\overline{\lim_{k\to\infty}}\|\nabla f\left(x_k\right)\|^2\right]$$

On a alors :

$$\overline{\lim_{k\to\infty}}\|\nabla f\left(s_k\right)\|^2+\overline{\lim_{k\to\infty}}\|\nabla f\left(x_k\right)\|^2 \leq 0$$

d'autre part :

$$\|\nabla f\left(x_k\right)\|^2 \geq 0 \Longrightarrow \underline{\lim_{k\to\infty}}\|\nabla f\left(x_k\right)\|^2 \geq 0$$

$$\|\nabla f\left(s_k\right)\|^2 \geq 0 \Longrightarrow \underline{\lim_{k\to\infty}}\|\nabla f\left(s_k\right)\|^2 \geq 0$$

d'où :

$$0 \leq \underline{\lim_{k\to\infty}}\|\nabla f\left(s_k\right)\|^2+\underline{\lim_{k\to\infty}}\|\nabla f\left(x_k\right)\|^2 \leq \overline{\lim_{k\to\infty}}\|\nabla f\left(s_k\right)\|^2+\overline{\lim_{k\to\infty}}\|\nabla f\left(x_k\right)\|^2 \leq 0$$

Ceci implique

$$\underline{\lim_{k\to\infty}}\|\nabla f\left(x_k\right)\|^2=\overline{\lim_{k\to\infty}}\|\nabla f\left(x_k\right)\|^2=\lim_{k\to\infty}\|\nabla f\left(x_k\right)\|^2=0$$

$$\underline{\lim_{k\to\infty}}\|\nabla f\left(s_k\right)\|^2=\overline{\lim_{k\to\infty}}\|\nabla f\left(s_k\right)\|^2=\lim_{k\to\infty}\|\nabla f\left(s_k\right)\|^2=0$$

Finalement on obtient

$$\lim_{k\to\infty}\|\nabla f\left(x_k\right)\|^2=0$$

Chapitre 5

L'EPSILON STEEPEST DESCENT ALGORITHME ASSOCIE A LA RECHERCHE LINEAIRE INEXACTE DE WOLFE

5.1 INTRODUCTION

Dans ce chapitre on considère le problème d'optimisation sans contraintes suivant :

$$(P) \quad \min \{f(x) : x \in R^n\}.$$

où $f : R^n \to R$ est continument différentiable. Notons

$$g_k = \nabla f(x_k)$$

On définit dans cette partie un nouveau algorithme qui accélère la convergence de la méthode du gradient. On étudie la convergence globale du nouveau algorithme qu'on anommé Wolfe epsilon steepest descent algorithme, en utilisant la recherche linéaire inéxacte de Wolfe ([35],[36]). Dans [16] et [33], Benzine, Djeghaba et Rahali ont étudié le même problème en utilisant une recherche linéaire exacte et une recherche linéaire inéxacte d'Armijo. On a aussi effectué 700 tests numériques et nous avons montré

que le nouveau algorithme est plus pérformant que les deux autres déjà étudiés i.e. l'epsilon steepest algorithme avec des recherches linéaires exactes ou d'Armijo.

Comme on l'a vu aux chapitres suivant, on génère une suite $\{x_k\}_{k\in\mathbb{N}}$ de la forme suivante :

$$x_{k+1} = x_k + \alpha_k d_k \tag{5.1}$$

où d_k est une direction de descente et α_k est le pas obtenu en effectuant une optimisation unidimensionnelle. Dans les méthodes du gradient conjugué les directions de descente sont de la forme :

$$d_k = -g_k + \beta_k d_{k-1} \tag{5.2}$$

où le scalaire β_k caractérise les différentes variantes du gradient conjugué. Si $\beta_k = 0$, alors on obtient la méthode du gradient. Un autre choix de directions est donné par

$$d_k = -B_k^{-1} g_k \tag{5.3}$$

où B_k est une matrice non singulière symetrique. Comme cas importants :

$$\begin{aligned} B_k &= I \quad (M\acute{e}thode\ steepest\ descent\ ) \\ B_k &= \nabla^2 f(x_k)\ (Methode\ de\ Newton) \end{aligned}$$

Les méthodes quasi Newton sont aussi de la forme (5.3).

Toutes ces méthodes sont implémentées en prenant en considération que d_k est une direction de descente i.e.

$$d_k^T g_k < 0$$

Les propriètées de convergence des méthodes à directions de descente et recherches linéaires dépendent du bon choix de d_k et du pas α_k. L'angle que fait la direction d_k et la direction du gradient $-g_k$ est fondamental. C'est pour celà qu'on définit

$$\cos(\theta_k) = \frac{-d_k^T g_k}{\|g_k\| \, \|d_k\|}$$

Nous choisirons α_k de sorte qu'on obtienne une décroissance suffisante de la fonction f, mais en même il faut que ce calcul ne soit pas coûteux en temps et en mémoire. Le choix optimal est obtenu en

choisissant α comme solution optimale de la fonction d'une variable $\varphi(\alpha)$ definie par

$$\varphi(\alpha) = f(x_k + \alpha d_k)$$

Les recherches linéaires exactes consistent à calculer α_k comme solution du problème unidimensionnel suivant :

$$f(x_k + \alpha_k d_k) = \min \{f(x_k + \alpha d_k) : \alpha > 0\}$$

Malheureusement, les recherches linéaires exactes sont difficiles à réaliser pratiquement et sont coûteuses en temps et en mémoire. La stratégie que nous allons appliquer dans cette partie consiste à choisir α_k vérifiant les deux conditions suivantes :

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \sigma_1 \alpha_k g_k^T d_k \tag{5.4}$$

$$g(x_k + \alpha_k d_k)^T d_k \geq \sigma_2 g_k^T d_k \tag{5.5}$$

où $0 < \sigma_1 < \sigma_2 < 1$. La première relation (5.4) (*condition d'Armijo ([5]),* assure que la fonction décroite suffisamment. La seconde condition (5.5) prévient que le pas α_k devienne trés petit. Les deux conditions (5.4) et (5.5) s'appellent conditions de Wolfe.

On peut aussi choisir α_k vérifiant les conditions suivantes :

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \sigma \alpha_k g_k^T d_k \tag{5.6}$$

$$f(x_k + \alpha_k d_k) \geq f(x_k) + (1 - \sigma) \alpha_k g_k^T d_k \tag{5.7}$$

où $0 < \sigma < \frac{1}{2}$. (5.6) et (5.7) s'appellent conditions de Goldstein.

Le théorème suivant est fondamental. On l'utilise pour démontrer la convergence globale des algorithmes utilisant des directions de descente et des recherches linéaires inéxactes. Ce résultat a été prouvé par Zoutendijk ([39]) et Wolfe ([35], [36]). L'Algorithme commence par un point quelconque x_1.

Théorème 5.1 ([39], [35], [36])

Supposons que f est minorée dans R^n et que f est continuement différentiable dans un voisinage N de l'ensemble $L = \{x : f(x) \leq f(x_1)\}$. Supposons aussi que le gradient est Lipshitzien , i.e., il existe $L > 0$ telle que

$$\|g(x) - g(y)\| \leq L \|x - y\| \tag{5.8}$$

pour $x, y \in N$. Considérons une suite de la forme (5.2), avec d_k une direction de descente et α_k

satisfaisant les conditions de Wolfe (5.4) et (5.5). Alors on a

$$\sum_{k=1}^{\infty} \cos^2(\theta_k) \|g_k\|^2 < \infty \tag{5.9}$$

La condition (5.9) s'appelle condition de Zoutendijk. Supposons que $\{x_k\}_{k\in\mathbb{N}}$ est de la forme suivante :

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

est telle que

$$\cos(\theta_k) \geq \delta > 0 \tag{5.10}$$

pour tout k. (5.9) implique

$$\lim_{k\to\infty} \|g_k\| = 0. \tag{5.11}$$

La méthode du gradient est l'une des plus simples et célèbres méthodes d'optimisation sans contraintes. Pour beaucoup de problèmes la méthode du gradient devient trés lente quans on s'approche d'un point stationnaire. IL existe beaucoup de méthodes qui y rémedient à ce problème. Au lieu de considerer $d_k = -\nabla f(x_k)$, on peut se déplacer le long de $d_k = -D_k \nabla f(x_k)$ ([8],[9],[10],[12],[13], [14], [15], [17], [21], [22],[23],[24], [25], [28],[31],[32],[33]), ou bien le long $d_k = -g_k + h_k$ ([18], [19],[20],[21],[27], [29],[30]), où D_k est une matrice choisie convenablement et h_k est un vecteur approprié.

Dans [16] et dans [33], Benzine, Djeghaba et Rahali ont essayé de résoudre ce problème par une autre méthode, en accélérant la convergence de la méthode du gradient.

Pour arriver à ce but, ils ont élaboré un nouveau Algorithme qu'ils ont nommé l'epsilon steepest descent algorithm, dans lequel la formule de Florent Cordellier et celle de Wynn ([11], [37], [38]) jouent un role essentiel. Ils ont aussi prouvé la convergence globale en utilisant des recherches linéaires exactes et d'Armijo.

Dans ce travail on accelère la convergence de la méthode du gradient et on étudie la convergence globale en utilisant l'Epsilon Algorithme et les recherches linéaires inexactes de wolfe vérifiant (5.4) et (5.5). Nous avons appelé le nouveau algorithme : Wolfe epsilon steepest descent algorithm.

700 tests numériques et nous avons montré que le nouveau algorithme est plus pérformant que les deux autres déjà étudiés i.e. l'epsilon steepest algorithme avec des recherches linéaires exactes ou d'Armijo ([16], [33]).

5.2 L' EPSILON ALGORITHME

L'Epsilon Algorithme est du à P. Wynn ([37], [38]).

Etant donnée une $\{x_k\}_{k\in\mathbb{N}}$, $x_k \in R^n$. Les cordonnées de x_k seront notées comme suit :

$$x_k^i = (x_k^1, x_k^2, ..., x_k^i, ..., x_k^n) \in R^n$$

Pour $i \in \{1, 2, ..., n\}$, l'Epsilon Algorithme calcule les quantités à deux indices suivantes $\varepsilon_j^{k,i}$ $(j, k = 0, 1, ...)$ comme suit :

$$\begin{aligned} &\varepsilon_{-1}^{k,i} = 0 \qquad \varepsilon_0^{k,i} = x_k^i \qquad\qquad k = 0, 1, ... \\ &\varepsilon_{j+1}^{k,i} = \varepsilon_{j-1}^{k+1,i} + \frac{1}{\varepsilon_j^{k+1,i} - \varepsilon_j^{k,i}} \qquad j, k = 0, 1, ... \end{aligned} \tag{5.12}$$

Pour $i \in \{1, 2, ..., n\}$, ces nombres peuvent être placées dans un tableau à double entrée comme suit :

$$\begin{array}{lllllllll}
\varepsilon_{-1}^{0,i} = 0 & & & & & & & & \\
& \varepsilon_0^{0,i} = x_0^i & & & & & & & \\
\varepsilon_{-1}^{1,i} = 0 & & \varepsilon_1^{0,i} & & & & & & \\
& \varepsilon_0^{1,i} = x_1^i & & \varepsilon_2^{0,i} & & & & & \\
\varepsilon_{-1}^{2,i} = 0 & & \varepsilon_1^{1,i} & & \varepsilon_3^{0,i} & & & & \\
& \varepsilon_0^{2,i} = x_2^i & & \varepsilon_2^{1,i} & & \varepsilon_4^{0,i} & & & \\
\varepsilon_{-1}^{3,i} = 0 & & \varepsilon_1^{2,i} & & \varepsilon_3^{1,i} & & \varepsilon_5^{0,i} & & \\
& \varepsilon_0^{3,i} = x_3^i & & \varepsilon_2^{2,i} & & \varepsilon_4^{1,i} & & \varepsilon_6^{0,i} & \\
\varepsilon_{-1}^{4,i} = 0 & & \varepsilon_1^{3,i} & & \varepsilon_3^{2,i} & & \varepsilon_5^{1,i} & & \varepsilon_7^{0,i} \\
& \varepsilon_0^{4,i} = x_4^i & & \varepsilon_2^{3,i} & & \varepsilon_4^{2,i} & & \varepsilon_6^{1,i} & & - \\
\varepsilon_{-1}^{5,i} = 0 & & \varepsilon_1^{4,i} & & \varepsilon_3^{3,i} & & \varepsilon_5^{2,i} & & \varepsilon_7^{1,i} \\
& \varepsilon_0^{5,i} = x_5^i & & \varepsilon_2^{4,i} & & \varepsilon_4^{3,i} & & \varepsilon_6^{2,i} & & - \\
\varepsilon_{-1}^{6,i} = 0 & & \varepsilon_1^{5,i} & & \varepsilon_3^{4,i} & & \varepsilon_5^{3,i} & & \varepsilon_7^{2,i} \\
& \varepsilon_0^{6,i} = x_6^i & & \varepsilon_2^{5,i} & & \varepsilon_4^{4,i} & & \varepsilon_6^{3,i} & & - \\
\varepsilon_{-1}^{7,i} = 0 & & \varepsilon_1^{6,i} & & \varepsilon_3^{5,i} & & \varepsilon_5^{4,i} & & \varepsilon_7^{3,i}
\end{array}$$

Ce tableau est appelé l'ε tableau.

Pour $i \in \{1, 2, ..., n\}$, l'Epsilon algorithm relie les quatres sommets du losange suivant :

$$\begin{array}{ccc} & \varepsilon_j^{k,i} & \\ \varepsilon_{j-1}^{k+1,i} & & \varepsilon_{j+1}^{k,i} \\ & \varepsilon_j^{k+1,i} & \end{array}$$

Pour calculer $\varepsilon_{j+1}^{k,i}$, on a besoin de $\varepsilon_{j-1}^{k+1,i}$, $\varepsilon_j^{k+1,i}$ et de $\varepsilon_j^{k,i}$.

5.3 L'ALGORITHME WOLFE EPSILON STEEPEST DESCENT

Pour construire notre algorithme on utilise les colonnes $\varepsilon_2^{k,i}$ $(i = 1, 2, ..., n)$. Etant donnée $\left\{x_k^i\right\}_{k\in\mathbb{N}}$ $(i = 1, 2, ..., n)$ F. Cordellier ([1], [6], [11]) a proposé une autre formule pour calculer $\varepsilon_2^{k,i}$ $(i = 1, 2, ..., n)$. Les quantités $\varepsilon_2^{k,i}$ sont caculées comme suit :

$$\varepsilon_2^{k,i} = x_{k+1}^i + \left[\frac{1}{x_{k+2}^i - x_{k+1}^i} - \frac{1}{x_{k+1}^i - x_k^i}\right]^{-1}, \quad (i = 1, 2, ..., n) \tag{5.13}$$

Pour calculer $\varepsilon_2^{k,i}$, on utilise x_k^i, x_{k+1}^i et x_{k+2}^i $(i = 1, 2, ..., n)$.

Nous sommes maintenant en mesure de donner le nouveau Algorithme :

Wolfe epsilon steepest descent algorithme

Initialisation : Choisir un vecteur initial $x_0 \in R^n$. Les coordonnées de x_0 sont notées comme suit :

$$x_0 = (x_0^1, x_0^2, ..., x_0^i, ..., x_0^n) \in R^n$$

Poser $k = 0$ et allez à l'étape principale.

Etape principale : A l'itération k, supposons qu'on ait le vecteur x_k,

$$x_k = (x_k^1, x_k^2, ..., x_k^i, ..., x_k^n).$$

Si $\|\nabla f(x_k)\| = 0$, stop. Sinon, posons $r_k = x_k$ et calculons s_k et t_k en utilisant deux fois la méthode du

gradient avec la recherche linéaire inexacte de Wolfe, i.e.

$$s_k = r_k - \lambda_k \nabla f(r_k),$$

et

$$t_k = s_k - \beta_k \nabla f(s_k),$$

λ_k et β_k sont des scalaires obtenus avec la recherche linéaire inexacte de Wolfe (5.4) et (5.5).
Si

$$s_k^i - r_k^i \neq 0, \quad t_k^i - s_k^i \neq 0 \quad and \quad \frac{1}{t_k^i - s_k^i} - \frac{1}{s_k^i - r_k^i} \neq 0, \quad i = 1, ...n$$

posons

$$\varepsilon_2^{k,i} = s_k^i + \left[\frac{1}{t_k^i - s_k^i} - \frac{1}{s_k^i - r_k^i} \right]^{-1}, \qquad i = 1, ...n,$$

et

$$\varepsilon_2^k = \left(\varepsilon_2^{k,1}, ..., \varepsilon_2^{k,i}, ...\varepsilon_2^{k,n} \right).$$

Si $f\left(\varepsilon_2^k\right) < f(t_k)$, posons $x_k = \varepsilon_2^k$. Remplacez k par $k+1$ et allez à Etape principale.
Si $f\left(\varepsilon_2^k\right) \geq f(t_k)$ ou si

$$s_k^{i_0} - r_k^{i_0} = 0 \text{ ou } t_k^{i_0} - s_k^{i_0} = 0 \quad \text{ou} \quad \frac{1}{t_k^{i_0} - s_k^{i_0}} - \frac{1}{s_k^{i_0} - r_k^{i_0}} = 0, \ i_0 \in \{1, ..., n\}.$$

Poser $x_k = t_k$. Remplacez k par $k+1$ et allez à Etape principale.

5.4 CONVERGENCE GLOBALE DE L'ALGORITHME WOLFE EPSILON STEEPEST DESCENT

Le théorème suivant garantit la convergence globale de la méthode Wolfe epsilon steepest descent Algorithme.

Théorème 5.2

Considérons le problème de minimisation sans contraintes (P) et x_1 le point de départ de l'Algorithme Wolfe epsilon steepest descent. Supposons aussi que les conditions suivantes soient satisfaites : f est minorée dans R^n et que f est continuement différentiable dans un voisinage N de l'ensemble $L =$

$\{x : f(x) \leq f(x_1)\}$. Supposons aussi que le gradient est Lipshitzien , i.e., il existe $L > 0$ telle que

$$\|g(x) - g(y)\| \leq L \|x - y\|$$

pour $x, y \in N$. Alors la suite $\{x_k\}_{k\in\mathbb{N}}$ *générée par l'Algorithme Wolfe epsilon steepest descent satisfait l'une des deux conditions suivantes :* $\nabla f(x_{k_0}) = 0$ *pour un certain indice* $k_0 \in N$ *ou* $\|\nabla f(x_k)\| \underset{k\to\infty}{\longrightarrow} 0$.

Preuve

Supposons qu'on a une suite infinie $\{x_k\}_{k\in\mathbb{N}}$ générée par l'Algorithme Wolfe epsilon steepest descent. Suivant l'Algorithme, les vecteurs s_k et t_k sont obtenus en utilisant deux fois la méthode du gradient avec la recherche linéaire inexacte de Wolfe. Alors on a

$$f(s_k) < f(r_k) = f(x_k)$$

et

$$f(t_k) < f(s_k)$$

Maintenant et en reconsidérant les conditions de l'Algorithme et si le calcul de ε_2^k est possible, alors deux cas sont possibles :

a) $f\left(\varepsilon_2^k\right) < f\left(t_k\right)$. Alors on a

$$f\left(x_{k+1}\right) = f\left(\varepsilon_2^k\right) < f\left(x_k\right)$$

ou bien

b) $f\left(\varepsilon_2^k\right) \geq f\left(t_k\right)$ ou le calcul de ε_2^k n'est pas possible. Dans ce cas et d'après l'Algorithme, on a

$$f\left(x_{k+1}\right) = f(t_k) < f\left(x_k\right).$$

En conclusion l'Algorithme Wolfe epsilon steepest descent garantit :

$$f\left(x_{k+1}\right) < f\left(x_k\right), \quad k = 0, 1, 2, ... \tag{5.14}$$

D'un autre coté, ε_2^k s'écrit comme suit :

$$\varepsilon_2^k = x_{k+1} = x_k + \alpha_k d_k, \tag{5.15}$$

où d_k et α_k dépend de $s_k,\ r_k,\ t_k,\ \lambda_k$ et β_k.

(5.14) implique que d_k est une direction de descente. Par conséquent, d'après le théorème de Zou-

dentijk, on a

$$\sum_{k=1}^{\infty} \cos^2(\theta_k) \|g_k\|^2 < \infty \tag{5.16}$$

(5.16) implique

$$\lim_{k\to\infty} \cos^2(\theta_k) \|g_k\|^2 = 0. \tag{5.17}$$

L'Algorithme Wolfe Epsilon steepest descent est semblable à la méthode steepest descent, donc il n'est pas difficile de démontrer qu'il existe $\delta > 0$ telle que

$$\cos(\theta_k) \geq \delta > 0 \tag{5.18}$$

pour tout k.

Notons

$$u_k = \cos^2(\theta_k),\ v_k = \|g_k\|^2,\quad w_k = u_k.v_k \tag{5.19}$$

(5.18) et (5.19) impliquent

$$u_k \neq 0,\quad v_k = \frac{w_k}{u_k} \tag{5.20}$$

et (5.17) donne

$$\lim_{k\to\infty} w_k = 0 \tag{5.21}$$

(5.18) et (5.20) impliquent aussi

$$0 \leq v_k \leq \frac{1}{\delta^2} w_k \tag{5.22}$$

Finalement (5.21) et (5.22) nous donnent

$$\lim_{k\to\infty} v_k = 0$$

ou encore

$$\lim_{k\to\infty} \|g_k\| = 0. \ \blacksquare \tag{5.23}$$

Remarque 5.1

Dans le théorème 5.2, les hypothèses sur la fonction f sont minimales. Contrairement aux méthodes quasi newtonniennes (BFGS, DFP,...), on n'exige pas la convexité de f.

5.5 RESULTATS NUMERIQUES

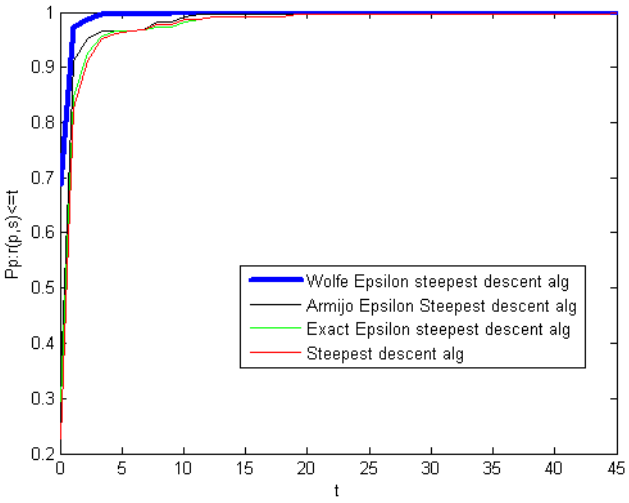
On va exposer dans cette partie des résultats numériques obtenus par implémentation de la méthode Wolfe Epsilon Steepest Descent algorithm. Pour obtenir nos résultats, nous avons utilisé les fonctions tests et leurs programmes en fortran tirés de ([2]). On utilise les mêmes critères que ceux utilisés dans ([3]). Les programmes ont été écrits en Fortran 90 et compilés dans une station Intel Pentium 4 with 2 GHz. Nous avons choisi 52 fonctions test pour des problèmes sans contraintes. Pour chaque fonction test, nous avons associé un nombre de variables croissant $n = 2, 10, 30, 50, 70, 100, 300, 500, 700, 900, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000$. On obtient ainsi 962 tests numériques. Notre algorithme implémente la méthode Wolfe Epsilon Steepest Descent algorithm avec les conditions de Wolfe (1.4), (1.5) ([35], [36]). L'algorithme s'arrête quand $\|\nabla f(x_k)\| < 10^{-6}$.

Pour comparer des algorithmes, nous avons utilisé le critère suivant : Soient f_i^{ALG1} et f_i^{ALG2} les valeurs optimales de f obtenus respectivement par ALG1 et ALG2, pour les problèmes $i = 1, ..., 962$. Pour le test i, nous dirons que l'algorithme1 : ALG1 est plus performent que le deuxième algorithme2 : ALG2 si :

$$\left|f_i^{ALG1} - f_i^{ALG2}\right| < 10^{-3}$$

et si le temps CPU accompli par ALG1 est inferieur au temps CPU correspondant à ALG2. On pourrait comparer aussi la performance par le nombre d'itérations.

Nous comparons à travers les tests numériques effectués, l'algorithme Wolfe Epsilon Steepest Descent algorithm et l'Algorithme Armijo Epsilon Steepest descent, l'algorithme Epsilon steepest descent algorithm version recherche linéaire exacte et enfin la méthode du gradient. Dans la Figure 1 on trouve le procédé de Dolan et Moré CPU performance. Ce procédé montre clairement que la méthode Wolfe Epsilon Steepest Descent algorithm est plus performente que l'Algorithme Armijo Epsilon Steepest descent, l'algorithme Epsilon steepest descent algorithm version recherche linéaire exacte et la méthode du gradient.


Pp:r(p,s)<=t
t
Wolfe Epsilon steepest descent alg
Armijo Epsilon Steepest descent alg
Exact Epsilon steepest descent alg
Steepest descent alg

$fig1$

5.6 CONCLUSIONS ET PERSPECTIVE

RESULTATS

Dans cette thèse, on a acceléré la convergence de la méthode du gradient et on a étudié la convergence globale en utilisant l'Epsilon Algorithme et les recherches linéaires inexactes de wolfe.

Avec 700 tests numériques nous avons montré que le nouveau algorithme est plus pérformant que les deux autres déjà étudiés i.e. l'epsilon steepest algorithme avec des recherches linéaires exactes ou d'Armijo ([16], [33]).

PERSPECTIVES

Problème ouvert 1 Nous avons démontré que l'algorithme Wolfe epsilon steepest descent convergeait globalement . Nous avons montré numériquent qu'il est plus performent que d'autres algorithmes de la même classe. Reste à prouver ce résutat théoriquement, ou encore montrer que notre algorithme a une vitesse de convergence superlinéaire. Nous conjecturons que celà est possible.

Bibliographie

[1] Aitken, A.C. ″On Bernoulli's numerical solution of algebraic equations″, *Proc. Roy. Soc. Edinburgh,* 46, pp.289-305, 1926.

[2] Andrei, N. An unconstrained optimization test functions collection. Advanced Modeling and Optimization, 10 (2008), pp. 147-161.

[3] Andrei, N. *Another conjugate gradient algorithm for unconstrained optimization.* Annals of Academy of Romanian Scientists, Series on Science and Technology of Information, vol. 1, nr.1, 2008, pp.7-20

[4] Bongartz, I, Conn, A.R., Gould, N.I.M., Toint, P.L., "Cute : constrained and unconstrained testing environments", ACM transactions on Mathematical software 21(1995) 123-160

[5] Armijo, L., " Minimization of functions having Lipschitz continuous first partial derivatives," *Pacific J. Mathematics,* 16, 1, pp. 1-3, 1966

[6] Brezinski, C. " *Acceleration de la convergence en analyse numérique* " *Lecture Notes in Mathematics,* 584, Springer Verlag (1977)

[7] Bongartz, I., Conn, A.R., Gould, N.I.M. and Toint, P.L. CUTE : constrained and unconstrained testing environments, ACM Trans. Math. Software, 21, pp.123-160, 1995.

[8] Broyden, C.G., "Quasi-Newton Methods and their application to Function Minimization" *Mathematics of Coputation,* 21, pp. 368-381, 1967

[9] Broyden, C.G., " The convergence of a class of double rank minimization algorithms 2. The new algorithm." *J. Institute of Mathematics and its applications,* 6, pp. 222-231, 1970.

[10] Broyden, C.G. , Dennis, J.E. Jr. and Moré, J .J. *"On the local and superlinear convergence of quasi-Newton methods,* J .Inst. Math.Appl . ,12 (1973) 223-246 .

[11] Cordellier, F. " Transformations de suites scalaires et vectorielles" Thèse de doctorat d'état soutenue à l'université de Lille I, 1981

[12] Davidon, W.C. "Variable Metric Method for Minimization", *AEC research Development*, Report ANL-5990, 1959.

[13] Dennis, J.E. Jr and Moré, J.J. "A characterization of superlinear convergence and its application to quasi-Newton methods," *Math.Comp.* 28, (1974), 549-560.

[14] Dennis, J.E. and Moré, JJ. "Quasi-Newton methods, motivation and theory", *SIAM .Rev.* 19 (1977) 46-89.

[15] Dixon, L .C .W. "Variable metric algorithms : necessary and sufficient conditions for identical behavior on nonquadratic functions", *J.Opt.Theory Appl.* 10 (1972) 34-40

[16] Djeghaba, N. and Benzine, R. "Accélération de la convergence de la méthode de la plus forte pente", *Demonstratio Mathematica.*Vol.39, N°1(2006), pp.169-181.

[17] Fletcher, R. "A new approch to Variable Metric Algorithms" *Computer Journal,* 13, pp. 317-322, 1970

[18] Fletcher, R, "Practical methods of Optimization," Second Edition , *John Wiley & Sons , Chichester* ,1987.

[19] Fletcher, R. "An overview of unconstrained optimization", *in Algorithms for Continuous Optimization* : the State of Art, E .Spedicato, ed., Kluwer Academic Publishers, 1994 .

[20] Fletcher, R. and M. Reeves. "Function minimization by conjugate gradients". *Computer J.7,* pp.149-154, 1964.

[21] Fletcher, R. and POWELL, MM., "A rapidly Convergent Descent Method for Minimization", *Computer Journal,* 6 pp.163-168, 1963.

[22] Forsythe G.E. '"On the asymptotic directions of the s-dimentional optimum gradient method," *Numerische Mathematik* 11, pp. 57-76

[23] Gill, P.E. and Marray, W. "Quasi-Newton Methods for unconstrained optimization," *J.Inst. Maths applics,* vol 9, pp 91-108, 1972 .

[24] Griewank, *A.,* "The global convergence of partitioned BFGS on problems with convex decompositions and Lipschitz gradients", *Math. Prog.* 50 (1991) 141-175 .

[25] Goldfarb, D., " A Family of Variable Metric Methods Derived by Variational Means," *Mathematics of Computation,* 24, pp. 23-26, 1970.

[26] Goldstein, A. A., and Price, J.F., "An effective Algorithm for Minimization" *Numerische Mathematik, 10,* pp. 184-189, 1967

[27] Hestenes, M.R and E. Stiefel. "Methods of conjugate gradients for solving linear systems". *J.Res. Nation-bureau Stand.* 49, N°6, 409-436, 1952

[28] Nocedal, J. and Wright, S.J. "Numerical Optimization" *Springer. Second edition.* 2006.

[29] Polak, E. and G. Ribiere "Note sur la convergence de méthodes de directions conjuguées" *Revue Française Informatique et Recherche opérationnelle*, 16, pp.35-43, 1969.

[30] Polyak, B.T., " The Method of Conjugate Gradient in Extremum Problems", *USSR Computational Mathematics and Mathematical Physics*(English Translation), 9pp. 94-112, 1969.

[31] Powell, .M.J.D, "On the convergence of the variable metric algorithms," *J.Inst. Math .Appl.* 7 (1971), 21-36.

[32] Powell, .M.J.D, "Some global convergence properties of variable metric algorithms for minimization without exact line searches," *in Nonlinear Programming, SIAM -AMS Proceedings,* VOL.IX, R . W .Cottle, and C .E Lemke, eds, SIAM 1976.

[33] Rahali, N., Djeghaba, N., Benzine, R "Global Convergence of the Armijo Epsilon Steepest Descent Algorithm." *Rev. Anal. Numer. Theor. Approx.* **41** (2012), no. 2, 169-180(2013

[34] Shanno, D.F., " Conditionning of quasi-Newton Methods for function minimization ", *Mathematics of Computation,* 24, pp. 641-656, 1970.

[35] Wolfe, P ."Convergence conditions for ascent méthods". *Siam Review* ,11,pp.226-235 ,(1969).

[36] Wolfe, P ."Convergence conditions for ascent méthods II : Some corrections" *Siam Review* ,13,pp.185-188 ,(1971).

[37] Wynn, P. "On a device for computing the em(Sn) transformation", *M.T.A.C.,* 10(1956), 91-96.

[38] Wynn, P., "Upon systems of recursions which obtain among quotients of the Padé table," *Numer. Math.,* 8(1966) pp. 264-269.

[39] Zoutendijk.G, Nonlinear programming computational methods, in : J. Abadie (Ed.), Integer and Nonlinear Programming, North-Holland, Amsterdam, 1970, pp. 37–86.

[40] Y. H. Dai and Y. Yuan (1998), Some properties of a new conjugate gradient method, in : Advances in Nonlinear Programming, ed . Kluwer Academic, Boston, pp. 251-262.