Optimisation sans contrainte de fonctions continues non linéaires

Guillaume Laurent

2012

Table des matières

1 Introduction

L'optimisation sans contrainte de fonctions continues non linéaires a pour objectif, étant donnée une fonction continue f de $\mathbb{R}^n$ dans $\mathbb{R}$, de trouver un vecteur X^* tel que $f(X^*)$ soit un extremum de f. Usuellement, f est appelée *fonction objectif* (ou *critère*, ou encore *coût*) et X variable de décision. Étant donné que la maximisation de f est équivalente à la minimisation de $-f$, tous les algorithmes présentés ici recherchent un minimiseur de la fonction objectif. Autrement dit, on cherche X^* tel que :

$$X^* \in \underset{X \in \mathbb{R}^n}{\arg\min}\, f(X) \tag{1}$$

Trouver un minimum global est en général un problème très difficile. En réalité, il n'existe pas d'algorithme d'optimisation parfait. En fonction du problème, on cherchera donc la méthode la mieux adaptée. Ce cours traite des méthodes numériques (algorithmes) permettant de trouver rapidement un minimum *local*. Ces algorithmes sont couramment utilisés par des méta-heuristiques (kangourou, recuit simulé, etc.) afin de trouver le « minimum des minima locaux ».

La présentation des algorithmes s'organise autour des critères suivants :

- les algorithmes utilisés sont différents selon que la fonction objectif est monovariable ou multivariable,
- les algorithmes qui utilisent uniquement la valeur de la fonction sont dits d'ordre zéro,
- les algorithmes qui utilisent la valeur de la fonction et la valeur de sa dérivée (de son gradient dans $\mathbb{R}^n$) sont dits d'ordre un,
- les algorithmes qui utilisent la valeur de la fonction, de sa dérivée et de sa dérivée seconde (de son hessien dans $\mathbb{R}^n$) sont dits d'ordre deux.

A la fin de chaque partie, une mise en pratique est proposé avec la toolbox optimisation du logiciel Matlab de Mathworks.

2 Optimisation unidimensionnelle

On s'intéresse dans cette partie à l'optimisation de fonctions objectifs monovariables (fonctions de $\mathbb{R}$ dans $\mathbb{R}$). L'intérêt des algorithmes d'optimisation unidimensionnelle ne vient pas seulement du fait que dans les applications on rencontre naturellement des problèmes unidimensionnels, mais aussi du fait que l'optimisation unidimensionnelle est un composant fondamental de bon nombre de méthodes d'optimisation multidimensionnelle.

2.1 Définitions

Définition 1 (minimum global)**.** Soit $f : \mathbb{R} \mapsto \mathbb{R}$ une fonction. $f(x^*)$ est *le minimum global* de f si et seulement si :

$$\forall x \in \mathbb{R},\ f(x) \geq f(x^*) \tag{2}$$

x^* est *un minimiseur global* de f.

Définition 2 (minimum local)**.** Soit $f : \mathbb{R} \mapsto \mathbb{R}$ une fonction. $f(x^*)$ est *un minimum local* de f si et seulement si :

$$\exists \epsilon,\ \forall x \in]x^* - \epsilon; x^* + \epsilon[,\ f(x) \geq f(x^*) \tag{3}$$

x^* est *un minimiseur local* de f.

Théorème 1 (condition suffisante de minimalité locale)**.** *Soit $f : \mathbb{R} \mapsto \mathbb{R}$ une fonction $\mathcal{C}^2$. Si $X^* \in \mathbb{R}$ vérifie les conditions :*

$$\begin{cases} f'(X^*) = 0 \\ f''(X^*) > 0 \end{cases} \tag{4}$$

Alors, $f(X^*)$ est un minimum local strict de f.

2.2 Méthodes du premier ordre (dichotomie)

L'idée la plus simple est de trouver un zéro de la dérivée de la fonction objectif à l'aide d'une recherche dichotomique (cf. algorithme 1).

Algorithme 1: Algorithme de recherche dichotomique

Données : f une fonction unimodale de classe $\mathcal{C}^1$ sur $[a; b]$ telle que $f'(a) < 0 < f'(b)$

```
tant que b − a > ε faire
    m ← (a+b)/2;
    si f'(m) ≥ 0 alors
        b ← m;
    sinon
        a ← m;
    fin
fin
```

Résultat : $\frac{a+b}{2}$

L'intérêt principal de cet algorithme est sa convergence linéaire avec le taux $0,5$. En revanche, il nécessite l'évaluation de la dérivée de la fonction objectif et impose de trouver un intervalle $[a; b]$ tel que $f'(a) < 0 < f'(b)$. Si la fonction n'est pas unimodale, l'algorithme peut converger vers un minimiseur local. Il se peut même que la valeur de la fonction au point trouvé soit supérieure à la valeur à l'une des bornes.

2.3 Méthodes du second ordre (méthode de Newton)

Il est possible d'utiliser la méthode de Newton pour rechercher un minimiseur (cf. algorithme 2). Le principe est de calculer une approximation p de f autour d'un point x_k par son développement de Taylor du second ordre :

$$p(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f''(x_k)(x - x_k)^2 \tag{5}$$

On calcule alors un nouveau point, x_{k+1}, qui minimise p soit :

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} \tag{6}$$

Algorithme 2: Algorithme de Newton (cas monovariable)

Données : f une fonction de classe $\mathcal{C}^2$ et x un point dans le voisinage d'un minimiseur de f

répéter
  $y \leftarrow x$;
  $x \leftarrow x - \frac{f'(x)}{f''(x)}$;
jusqu'à $|x - y| < \epsilon$;

Résultat : x

Si le point x_0 est assez proche d'un minimiseur local de f, alors on montre que la suite x_k converge de façon quadratique ($|x_{k+1} - x^*| \leq \beta |x_k - x^*|^2$).

D'un point de vue pratique, cette méthode souffre de nombreux inconvénients :

- la méthode peut diverger si le point de départ est trop éloigné de la solution,
- la méthode n'est pas définie si $f''(x_k) = 0$,
- la méthode peut converger indifféremment vers un minimum, un maximum ou un point de selle.

Cette méthode est donc peu utilisée. Son intérêt majeur est d'illustrer dans $\mathbb{R}$ le fonctionnement des algorithmes de minimisation multidimensionnelle du second ordre (cf. section 3.4).

2.4 Méthodes d'ordre zéro

Pour une fonction unimodale f, si l'on trouve trois points $a < b < c$ tels que $f(b) < f(a)$ et $f(b) < f(c)$, alors le minimum de f est atteint dans l'intervalle $]a; c[$. Les algorithmes de minimisation d'ordre zéro exploitent cette idée en encadrant de plus en plus précisément le minimiseur (cf. algorithme 3).

Il y a différentes possibilités pour le choix des points intérieurs. La méthode la plus utilisée est la méthode de la section d'or (cf. algorithme 4). Pour éviter d'évaluer la fonction en deux nouveaux points à chaque itération, l'astuce est de s'arranger pour en réutiliser un d'une itération à l'autre.

Algorithme 3: Algorithme d'ordre zéro

Données : f une fonction unimodale sur $[a; b]$

tant que $b - a > \epsilon$ **faire**
  Choisir deux points x^- et x^+, tels que $a < x^- < x^+ < b$;
  si $f(x^-) \leq f(x^+)$ **alors**
    $b \leftarrow x^+$;
  sinon
    $a \leftarrow x^-$;
  fin
fin

Résultat : $\frac{a+b}{2}$

Soit $[a_k; b_k]$ l'intervalle de recherche à l'instant k, on pose :

$$x_k^- = a_k + (1-\rho)(b_k - a_k) = \rho a_k + (1-\rho) b_k \tag{7}$$
$$x_k^+ = a_k + \rho(b_k - a_k) = (1-\rho) a_k + \rho b_k \tag{8}$$

avec $0,5 < \rho < 1$.

A l'itération suivante, si le minimum est à droite de x_k^-, on a $a_{k+1} = x_k^-$ et $b_{k+1} = b_k$. On souhaite réutiliser x_k^+, c'est-à-dire que l'on veut :

$$x_{k+1}^- = x_k^+ \tag{9}$$

donc :

$$\rho a_{k+1} + (1-\rho) b_{k+1} = (1-\rho) a_k + \rho b_k \tag{10}$$
$$\rho(\rho a_k + (1-\rho) b_k) + (1-\rho) b_k = (1-\rho) a_k + \rho b_k \tag{11}$$
$$\rho^2 + \rho - 1 = 0 \tag{12}$$

Comme $0,5 < \rho < 1$, on obtient : $\rho = \frac{\sqrt{5}-1}{2}$. ρ est l'inverse du nombre d'or $\varphi = (1+\sqrt{5})/2$.

La convergence de cette méthode est linéaire avec le taux $\rho \approx 0.61803$, ce qui est moins bien que la recherche dichotomique ou la méthode de Newton. Le principal avantage de cette méthode est qu'elle ne nécessite pas de calculer la dérivée.

Cette méthode est souvent améliorée en utilisant quand cela est judicieux une interpolation parabolique ou cubique. On peut notamment citer la méthode de Brent utilisée par les *Numerical Recipes* et Matlab [3, 18].

2.5 Encadrement d'un minimiseur

Il est parfois nécessaire de trouver un encadrement d'un minimiseur à partir d'un point et d'une direction. Cet algorithme de recherche est notamment employé par les algorithmes de descente multivariables (cf. section 3.2).

Algorithme 4: Algorithme de la section d'or

Données : f une fonction unimodale sur $[a; b]$ et $\rho = \frac{\sqrt{5}-1}{2} \approx 0.62$

$x^- \leftarrow \rho a + (1-\rho)b$;
$x^+ \leftarrow a + b - x^-$;
$v^- \leftarrow f(x^-)$;
$v^+ \leftarrow f(x^+)$;
tant que $b - a > \epsilon$ **faire**
    si $v- \leq v+$ **alors**
        $b \leftarrow x^+$;
        $x^+ \leftarrow x^-$;
        $x^- \leftarrow a + b - x^+$;
        $v^+ \leftarrow v^-$;
        $v^- \leftarrow f(x^-)$;
    sinon
        $a \leftarrow x^-$;
        $x^- \leftarrow x^+$;
        $x^+ \leftarrow a + b - x^-$;
        $v^- \leftarrow v^+$;
        $v^+ \leftarrow f(x^+)$;
    fin
fin

Résultat : $\frac{a+b}{2}$

Partant d'un point a, comment trouver un point c tel que $[a; c]$ contienne un minimum ? Le principe de l'algorithme d'encadrement (cf. algorithme 5) est de rechercher deux points b et c vérifiant $a < b < c$ et tels que $f(b) < f(a)$ et $f(b) < f(c)$ (on suppose ici que le minimum est à droite, i.e. $f'(a) < 0$).

Algorithme 5: Algorithme d'encadrement d'un minimiseur (vers la droite)

Données : f une fonction telle que $f'(a) < 0$, K une constante (souvent 2 ou φ) et λ une constante d'échelle

```
c ← a + λ;
si f(c) < f(a) alors
    répéter
        b ← c;
        c ← a + K * (c − a);
    jusqu'à f(c) > f(b);
fin
```

Résultat : a et b

Comme pour l'algorithme de la section d'or, la recherche peut être accélérée en utilisant des extrapolations paraboliques ou cubiques [18, 5].

2.6 Mise en pratique avec Matlab

On propose de trouver un minimiseur au problème « Agent 007 » tiré de l'ouvrage de Bierlaire [2] à l'aide la toolbox optimisation de Matlab. La fonction objectif est définie par :

$$f(x) = 0.2x + 0.45\sqrt{(100 - x)^2 + 1600} \tag{13}$$

Tout d'abord, il s'agit de définir la fonction objectif. On écrit pour cela une fonction dans un fichier `.m`, par exemple :

```
function f=Agent007(x)

f=0.2*x+0.45*sqrt((100-x)^2+1600);
```

`f` est la valeur de la fonction au point `x`.

On utilise ensuite la fonction, `fminbnd` dont le paramétrage est défini à l'aide de la fonction `optimset` :

```
options=optimset('MaxFunEvals',3,'MaxIter',10,'TolX',1e-3);

[x,fval]=fminbnd('Agent007',0,100,options);
```

La fonction `fminbnd` utilise la méthode de Brent.

Avec quelle précision peut-on espérer approcher un minimiseur ? On pourrait penser que si ϵ est la précision de la machine et x^* un minimiseur, on pourra obtenir une valeur entre $(1 - \epsilon)x^*$ et $(1 + \epsilon)x^*$. En réalité, cela est généralement impossible.

Au voisinage de x^*, la fonction objectif peut être estimée par son développement de Taylor au second ordre, sachant que $f'(x^*) = 0$, c'est-à-dire :

$$f(x) = f(x^*) + \frac{1}{2}f''(x^*)(x - x^*)^2 + o((x - x^*)^2) \tag{14}$$

Avec la méthode de la section d'or, il faut comparer les valeurs de la fonction. Si $|x - x^*|$ est de l'ordre de $o(\alpha)$ alors $|f(x) - f(x^*)|$ est de l'ordre de $o(\alpha^2)$ (en supposant que $f''(x^*)$ est de l'ordre de 1). Donc, si ϵ est la précision de la machine, alors on pourra obtenir une précision sur x de l'ordre de $o(\sqrt{\epsilon})$.

Pour la dichotomie, on doit calculer le signe de la dérivée. Si $|x - x^*|$ est de l'ordre de $o(\alpha)$ alors $|f'(x)|$ est de l'ordre de $o(\alpha)$ (en supposant que $f''(x)$ est de l'ordre de 1). Donc, si ϵ est la précision de la machine, alors on pourra obtenir une précision sur x de l'ordre de $o(\epsilon)$.

3 Optimisation multidimensionnelle

On s'intéresse maintenant à l'optimisation de fonctions objectifs multivariables (fonctions de $\mathbb{R}^n$ dans $\mathbb{R}$).

3.1 Définitions

Définition 3 (gradient)**.** Soit $f : \mathbb{R}^n \mapsto \mathbb{R}$ une fonction différentiable. La fonction notée $\nabla f(X) : \mathbb{R}^n \mapsto \mathbb{R}^n$ est appelée le *gradient* de f et est définie par :

$$\nabla f(X) = \begin{pmatrix} \frac{\partial f(X)}{\partial x_1} \\ \vdots \\ \frac{\partial f(X)}{\partial x_n} \end{pmatrix} \tag{15}$$

Définition 4 (pente)**.** Soit $f : \mathbb{R}^n \mapsto \mathbb{R}$ une fonction différentiable. Soient $X, D \in \mathbb{R}^n$. La quantité :

$$\frac{D^{\mathrm{T}} \nabla f(X)}{\|D\|} \tag{16}$$

représente la *pente* de la fonction f en X dans la direction D.

Définition 5 (point critique)**.** Soit $f : \mathbb{R}^n \mapsto \mathbb{R}$ une fonction différentiable. Tout vecteur $X \in \mathbb{R}^n$ tel que $\nabla f(X) = \mathbf{0}$ est appelé *point critique* ou *point stationnaire* de f.

Définition 6 (hessien)**.** Soit $f : \mathbb{R}^n \mapsto \mathbb{R}$ une fonction deux fois différentiable. La fonction notée $\nabla^2 f(X) : \mathbb{R}^n \mapsto \mathbb{R}^{n \times n}$ est appelée *matrice hessienne* ou *hessien* de f et

est définie par :

$$\nabla^2 f(X) = \begin{pmatrix} \frac{\partial^2 f(X)}{\partial x_1^2} & \frac{\partial^2 f(X)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(X)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(X)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(X)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(X)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(X)}{\partial x_n \partial x_1} & \frac{\partial^2 f(X)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(X)}{\partial x_n^2} \end{pmatrix} \tag{17}$$

La matrice hessienne est symétrique par définition.

Définition 7 (courbure)**.** Soit $f : \mathbb{R}^n \mapsto \mathbb{R}$ une fonction deux fois différentiable. Soient $X, D \in \mathbb{R}^n$. La quantité :

$$\frac{D^{\mathrm{T}} \nabla^2 f(X) D}{\|D\|^2} \tag{18}$$

représente la *courbure* de la fonction f en X dans la direction D.

Théorème 2 (condition suffisante de minimalité locale)**.** *Soit $f : \mathbb{R}^n \mapsto \mathbb{R}$ une fonction deux fois différentiable dans un sous-ensemble ouvert V de $\mathbb{R}^n$. Si $X^* \in V$ vérifie les conditions :*

$$\begin{cases} \nabla f(X^*) = 0 \\ \nabla^2 f(X^*) \text{ est définie positive (i.e. } \forall D \in \mathbb{R}^n \ D^{\mathrm{T}} \nabla^2 f(X^*) D > 0) \end{cases} \tag{19}$$

Alors, X^* est un minimiseur local strict de f.

3.2 Algorithme de descente

Définition 8 (direction de descente)**.** Soit $f : \mathbb{R}^n \mapsto \mathbb{R}$ une fonction différentiable. Soient $X, D \in \mathbb{R}^n$. La direction D est une *direction de descente* en X si

$$D^{\mathrm{T}} \nabla f(X) < 0. \tag{20}$$

La direction du gradient est celle dans laquelle la fonction a la plus forte pente. La direction opposée au gradient est donc celle dans laquelle la fonction a la plus forte descente.

Théorème 3 (descente)**.** *Soit $f : \mathbb{R}^n \mapsto \mathbb{R}$ une fonction différentiable. Soient $X \in \mathbb{R}^n$ tel que $f(X) \neq 0$ et $D \in \mathbb{R}^n$. Si D est une direction de descente en X alors il existe η tel que :*

$$\forall \alpha \in [0; \eta[, \qquad f(X + \alpha D) < f(X) \tag{21}$$

En utilisant ce résultat, on construit un algorithme général de minimisation, nommé *algorithme de descente* (cf. algorithme 6). Il consiste simplement à suivre une direction de descente de façon itérative jusqu'à l'obtention d'un « bon » minimiseur. On utilise souvent plus d'un critère d'arrêt. Habituellement, on trouvera :

– un critère sur le déplacement, si $\|X_{k+1} - X_k\|$ est très petit, c'est qu'on ne progresse plus beaucoup.
– un critère sur la progression de l'objectif, si $|f(X_{k+1}) - f(X_k)|$ est très petit, on peut être presque arrivé à un minimum. On peut aussi mesurer la norme du gradient.
– un critère sur le temps de calcul ou le nombre d'itérations.

Évidemment, rien ne vaut une bonne connaissance de la fonction à optimiser, quand il faut décider de critères d'arrêt.

Algorithme 6: Algorithme de descente

Données : f une fonction différentiable et X_0 un point initial

$k \leftarrow 0$;
tant que *critère d'arrêt non satisfait* **faire**
  Trouver une direction de descente D_k telle que ${D_k}^{\mathrm{T}} \nabla f(X_k) < 0$;
  Trouver un pas α_k tel que $f(X_k + \alpha_k D_k) < f(X_k)$;
  $X_{k+1} \leftarrow X_k + \alpha_k D_k$;
  $k \leftarrow k + 1$;
fin

Nous détaillerons plus loin les différentes méthodes de choix de la direction de descente. Une fois la direction déterminée, il reste à calculer une longueur de pas. Si l'on choisit naïvement :

$$\|\alpha_k D_k\| = \text{constante} \tag{22}$$

La précision de la recherche est alors limitée à $\|\alpha_k D_k\|$. S l'on choisit $\|\alpha_k D_k\|$ petit la convergence est lente.

Il faut donc trouver à chaque itération une valeur de pas adaptée. Cette étape est appelée *recherche en ligne* (*line search*).

Comme on cherche à minimiser f et que l'on dispose de méthodes efficaces pour minimiser une fonction monovariable, on peut chercher un pas α_k diminuant le plus f dans la direction d_k, c'est-à-dire :

$$\alpha_k \in \underset{\alpha \in \mathbb{R}^+}{\arg\min}\, f(X_k + \alpha D_k) \tag{23}$$

Pour ce faire, on commence par trouver un intervalle qui encadre un minimiseur (cf. section 2.5) puis on applique une méthode de minimisation à la fonction $g(\alpha) = f(X_k + \alpha D_k)$ [5, 18].

D'autres méthodes consistent à trouver rapidement un pas « acceptable », c'est la notion de progrès suffisant. On cherche un pas α diminuant *suffisamment* la fonction $g(\alpha)$ par rapport à $g(0)$.

Pour déterminer si un pas est « acceptable », il existe des tests comme le test d'Armijo, le test de Wolfe ou celui de Goldstein. Pour chaque test, il existe un algorithme ayant une convergence linéaire permettant de trouver une valeur qui passe le test sous la condition naturelle que $g'(0) < 0$ [2, 12].

3.3 Méthodes du premier ordre

3.3.1 Méthode de la plus forte pente (méthode du gradient simple)

L'idée la plus simple consiste à utiliser le gradient comme direction de descente, soit :

$$D_k = -\nabla f(X_k) \tag{24}$$

Dans le cas général, sous des hypothèses assez faibles de régularité, la méthode de la plus forte pente converge vers un point critique de la fonction objectif [12].

Si cette méthode est simple à mettre en oeuvre, les résultats sont décevants car sa vitesse de convergence est relativement basse et très sensible à la « forme » de la fonction objectif. *Grosso modo*, la vitesse de convergence est liée au conditionnement de la matrice hessienne. Si le hessien est mal conditionné, la fonction objectif ressemble à une « vallée » étroite. La pente est beaucoup plus forte dans la direction orthogonale à la vallée. Dans ces conditions, la direction du gradient est presque orthogonale à la direction de la vallée pour des points même proches du fond. L'algorithme suit donc une trajectoire en zigzag qui converge très lentement vers un minimiseur.

De plus, même si l'étape de recherche en ligne est optimale (on obtient alors l'*algorithme de la plus forte pente à pas optimal*, cf. algorithme 7), la méthode n'est pas plus efficace car, dans ce cas, comme la pente est nulle en X_{k+1} dans la direction D_k, le gradient en X_{k+1} est normal à D_k soit :

$$D_k{}^{\mathrm{T}} \nabla f(X_{k+1}) = 0 \tag{25}$$

L'algorithme suit donc une trajectoire en zigzag à angles droits.

Algorithme 7: Algorithme de la plus forte pente à pas optimal

Données : f une fonction différentiable et X_0 un point initial

$X \leftarrow X_0$;
tant que *critère d'arrêt non satisfait* **faire**
  $D \leftarrow -\nabla f(X)$;
  Trouver un pas α qui minimise $f(X + \alpha D)$;
  $X \leftarrow X + \alpha D$;
fin

3.3.2 Méthode du gradient conjugué

La méthode du gradient conjugué est une méthode de descente à pas optimal permettant de minimiser une fonction quadratique de $\mathbb{R}^n$ dans $\mathbb{R}$ en au plus n itérations. [20].

Soit f une fonction quadratique :

$$f(X) = \frac{1}{2} X^{\mathrm{T}} Q X - B^{\mathrm{T}} X \tag{26}$$

avec Q une matrice définie positive. On a :

$$\nabla f(X) = (QX - B)^{\mathrm{T}} \qquad \nabla^2 f(x) = Q \tag{27}$$

On construit la suite :

$$X_{k+1} = X_k + \alpha_k D_k \tag{28}$$

avec :

$$\alpha_k \in \underset{\alpha \in \mathbb{R}^+}{\arg\min}\, f(X_k + \alpha D_k) \tag{29}$$

La première idée fondamentale de l'algorithme du gradient conjugué consiste à choisir chaque direction de descente conjuguée à la direction de descente précédente par rapport à Q.

Définition 9 (directions conjuguées)**.** Soit une matrice $Q \in \mathbb{R}^{n \times n}$ définie positive. Deux vecteurs U et V de $\mathbb{R}^n$ sont *conjugués* par rapport à Q si $U^{\mathrm{T}} Q V = 0$. Noter que, si Q est la matrice identité, les directions conjuguées sont orthogonales.

En dimension $n > 2$, il existe une infinité de directions conjuguées à une direction donnée. Pour trouver à l'itération k, une direction D_k conjuguée à D_{k-1}, la seconde idée fondamentale consiste à chercher D_k sous forme d'une combinaison linéaire de D_{k-1} et de la direction de la plus forte pente au point X_k, soit :

$$D_k = -\nabla f(X_k) + s_k D_{k-1} \tag{30}$$

en choisissant le coefficient s_k tel que les directions successives soient conjuguées par rapport à Q :

$${D_{k-1}}^{\mathrm{T}} Q D_k = 0 \tag{31}$$

c'est-à-dire :

$$-{D_{k-1}}^{\mathrm{T}} Q \nabla f(X_k) + s_k {D_{k-1}}^{\mathrm{T}} Q D_{k-1} = 0 \tag{32}$$

Ceci est toujours possible si D_{k-1} est non nul. On montre alors que le réel s_k se calcule par la formule suivante (Fletcher-Reeves) :

$$s_k = \frac{\|\nabla f(X_k)\|^2}{\|\nabla f(X_{k-1})\|^2} \tag{33}$$

L'équation de récurrence permettant de calculer une nouvelle direction est donc :

$$D_k = -\nabla f(X_k) + \frac{\|\nabla f(X_k)\|^2}{\|\nabla f(X_{k-1})\|^2} D_{k-1} \tag{34}$$

Par définition, cette direction est toujours une direction de descente puisque :

$$\nabla f(X_k)^{\mathrm{T}} D_k = -\|\nabla f(X_k)\| + s_k \nabla f(X_k)^{\mathrm{T}} D_{k-1} = -\|\nabla f(X_k)\| \tag{35}$$

A chaque itération, le pas est déterminé comme dans tout algorithme de descente par une recherche en ligne. On obtient ainsi un algorithme de minimisation appelé *algorithme de Fletcher-Reeves* (cf. algorithme 8) [10].

Lorsque la fonction objectif est quadratique, cet algorithme génère une suite de directions de recherche deux à deux conjuguées qui converge vers le minimiseur en au plus n itérations quel que soit x_0 !

Dans les autres cas, on peut raisonnablement penser que dans un voisinage de X^*, la fonction objectif n'est pas très différente d'une fonction quadratique et qu'on peut donc appliquer cette méthode. L'efficacité de l'algorithme repose essentiellement sur deux points :

– la recherche en ligne doit être exacte,
– les relations de conjugaisons doivent être précises.

La recherche en ligne peut être facilement rendu assez précise. En revanche, si la fonction est loin d'une fonction quadratique, la notion de conjugaison relative n'a pas de signification. Ainsi, il peut être judicieux de réinitialiser régulièrement la direction courante D par la direction de la plus grande pente $-\nabla f(X)$ (tous les m itérations avec m de l'ordre de n).

Par rapport aux algorithmes de quasi-Newton présentés plus loin, l'algorithme du gradient conjugué ne nécessite le stockage que de deux vecteurs de taille n. Il est donc particulièrement adapté aux problèmes de grandes dimensions.

Un variante a été proposée par Polak et Ribière en 1969. Le principe de l'algorithme est identique, seule la mise à jour de la direction est légèrement modifiée :

$$D_k = -\nabla f(X_k) + \frac{(\nabla f(X_k) - \nabla f(X_{k-1}))^{\mathrm{T}} \nabla f(X_k)}{\|\nabla f(X_{k-1})\|^2} D_{k-1} \tag{36}$$

Cette variante est réputée plus performante dans certaines applications.

3.4 Méthodes du second ordre

3.4.1 Méthode de Newton

La méthode de Newton pour les fonctions multivariables est identique à celle des fonctions univariées. Comme précédemment, une estimation $p(X)$ de f au point X_k est donnée par le développement de Taylor du second ordre, qui s'écrit pour une fonction multivariée :

$$p(X) = f(X_k) + (X - X_k)^{\mathrm{T}} \nabla f(X_k) + \frac{1}{2}(X - X_k)^{\mathrm{T}} [\nabla^2 f(X_k)](X - X_k) \tag{37}$$

Algorithme 8: Algorithme du gradient conjugué (Fletcher-Reeves)

Données : f une fonction différentiable et X_0 un point initial

$X \leftarrow X_0$;
$G \leftarrow \nabla f(X)$;
$D \leftarrow -G$;
$s \leftarrow \|G\|^2$;
tant que *critère d'arrêt non satisfait* **faire**
  Trouver un pas α qui minimise $f(X + \alpha D)$;
  $X \leftarrow X + \alpha D$;
  $G \leftarrow \nabla f(X)$;
  $t \leftarrow s$;
  $s \leftarrow \|G\|^2$;
  $D \leftarrow -G + \frac{s}{t} D$;
fin

Résultat : X

Si la matrice hessienne est inversible, on choisit X_{k+1} qui minimise p, soit :

$$X_{k+1} = X_k - \left[\nabla^2 f(X_k)\right]^{-1} \nabla f(X_k) \tag{38}$$

En pratique, la direction de descente $D = -\left[\nabla^2 f(X_k)\right]^{-1} \nabla f(X)$ est calculée sans inverser $\left[\nabla^2 f(X_k)\right]$ mais en résolvant :

$$\nabla^2 f(X_k) D = -\nabla f(X) \tag{39}$$

L'intérêt de cette suite est sa convergence quadratique vers un minimiseur local à la condition que X_0 soit assez proche d'un minimiseur. Néanmoins d'un point de vue pratique, cette méthode comporte les mêmes inconvénients que dans le cas monovariable :

- la méthode peut diverger si le point de départ est trop éloigné de la solution,
- la méthode n'est pas définie si la matrice hessienne n'est pas inversible,
- la méthode peut converger indifféremment vers un minimum, un maximum ou un point de selle.

Algorithme 9: Algorithme de Newton (cas multivariable)

Données : f une fonction différentiable et X_0 un point initial

$X \leftarrow X_0$;
tant que *critère d'arrêt non satisfait* **faire**
  Calculer D solution de $\nabla^2 f(X) D = -\nabla f(X)$;
  $X \leftarrow X + D$;
fin

3.4.2 Méthode de Newton avec recherche en ligne

A partir de la méthode de Newton, on peut en déduire un algorithme plus robuste (cf. algorithme 10). L'idée est d'utiliser la direction de Newton quand elle est définie et sinon la direction de la plus forte pente. Puis, on effectue à chaque itération une recherche en ligne dans cette direction.

En pratique, la direction de descente D est calculée en résolvant :

$$(\nabla^2 f(X) + \Delta)D = -\nabla f(X) \tag{40}$$

où Δ est une matrice diagonale telle que $\nabla^2 f(X)+\Delta$ soit définie positive afin de garantir que la direction soit une direction de descente.

Algorithme 10: Algorithme de Newton avec recherche en ligne

Données : f une fonction différentiable et X_0 un point initial

$X \leftarrow X_0$;
tant que *critère d'arrêt non satisfait* **faire**
  Calculer D solution de $(\nabla^2 f(X) + \Delta)D = -\nabla f(X)$;
  Trouver un pas α qui minimise $f(X + \alpha D)$;
  $X \leftarrow X + \alpha D$;
fin

3.4.3 Méthodes quasi-Newton

Pour des problèmes de grandes dimensions, le calcul du hessien est trop coûteux. On peut alors utiliser des algorithmes, dits quasi-Newton, qui calculent donc une approximation B_k de $\left(\nabla^2 f(X_k)\right)^{-1}$ en fonction de B_{k-1}, $\nabla f(X_k)$, $\nabla f(X_{k-1})$, X_k et X_{k-1}.

On trouve notamment deux méthodes de calcul :

– la formule de Davidon-Fletcher-Powell (DFP) [7, 9] :

$$B_k = B_{k-1} - \frac{B_{k-1} Y_k {Y_k}^{\mathrm{T}} B_{k-1}}{{Y_k}^{\mathrm{T}} B_{k-1} Y_k} + \frac{\bar{D}_{k-1} \bar{D}_{k-1}^{\mathrm{T}}}{\bar{D}_{k-1}^{\mathrm{T}} Y_k} \tag{41}$$

– la formule de Broyden-Fletcher-Goldfarb-Shanno (BFGS) [4, 8, 11, 19] :

$$B_k = B_{k-1} + \left(1 + \frac{{Y_k}^{\mathrm{T}} B_{k-1} Y_k}{{Y_k}^{\mathrm{T}} \bar{D}_{k-1}}\right) \frac{\bar{D}_{k-1} \bar{D}_{k-1}^{\mathrm{T}}}{{Y_k}^{\mathrm{T}} \bar{D}_{k-1}} - \frac{B_{k-1} Y_k \bar{D}_{k-1}^{\mathrm{T}} + \bar{D}_{k-1} {Y_k}^{\mathrm{T}} B_{k-1}}{{Y_k}^{\mathrm{T}} \bar{D}_{k-1}} \tag{42}$$

avec

$$Y_k = \nabla f(X_k) - \nabla f(X_{k-1}) \tag{43}$$

Ces formules donnent toujours des matrices définies positives. A l'aide de ces estimations, on établit un algorithme à convergence très efficace et particulièrement robuste. Son unique inconvénient est la nécessité du stockage en mémoire d'une matrice de taille $n \times n$. Dans la pratique, on utilise en général la méthode BFGS qui est plus efficace.

Algorithme 11: Algorithme de quasi-Newton avec mise à jour BFGS

Données : f une fonction différentiable et X_0 un point initial

$X \leftarrow X_0$;
$B \leftarrow I$;
$G \leftarrow \nabla f(X)$;
tant que *critère d'arrêt non satisfait* **faire**
  $D \leftarrow -BG$;
  Trouver un pas α qui minimise $f(X + \alpha D)$;
  $\bar{D} \leftarrow \alpha D$;
  $X \leftarrow X + \bar{D}$;
  $Y \leftarrow -G$;
  $G \leftarrow \nabla f(X)$;
  $Y \leftarrow Y + G$;
  $B \leftarrow B + \left(1 + \frac{Y^{\mathrm{T}} B Y}{Y^{\mathrm{T}} \bar{D}}\right) \frac{\bar{D}\bar{D}^{\mathrm{T}}}{Y^{\mathrm{T}} \bar{D}} - \frac{B Y \bar{D}^{\mathrm{T}} + \bar{D} Y^{\mathrm{T}} B}{Y^{\mathrm{T}} \bar{D}}$;
fin

Résultat : X

3.5 Méthodes d'ordre zéro (simplexe de Nelder et Mead)

Existe-t-il une méthode performante n'utilisant pas le gradient de la fonction objectif ? La réponse est oui. La méthode de Nelder et Mead [17, 13] souvent appelée « méthode du simplexe »[1] est une méthode efficace qui n'utilise que la valeur de la fonction.

Définition 10 (simplexe)**.** Un simplexe de dimension k est l'enveloppe convexe de $k+1$ vecteurs $X_1, \ldots, X_{k+1}$ de $\mathbb{R}^n$, $k = n$, affinement indépendants, c'est-à-dire que les k vecteurs $X_1 - X_{k+1}, X_2 - X_{k+1}, \ldots, X_k - X_{k+1}$ sont linéairement indépendants.

Par exemple, trois points non alignés dans $\mathbb{R}^2$, ou quatre points non coplanaires dans $\mathbb{R}^3$ sont affinement indépendants, et définissent des simplexes de dimension 2 et 3, respectivement.

Le principe de l'algorithme de Nelder et Mead est de faire évoluer un simplexe vers un minimiseur de la fonction objectif par des expansions ou des contractions successives du simplexe en fonction de la topologie locale (cf. algorithme 12 et figure 1).

Pour déterminer un simplexe initial autour d'un point X_0, on pose $X_{n+1} = X_0$ et on calcule pour tout i de 1 à n :

$$X_i = x_0 + \lambda E_i \tag{44}$$

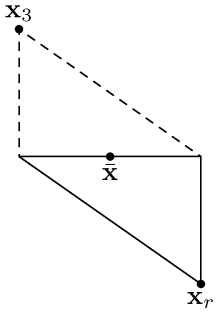
où E_i est le ième vecteur unité et λ une constante d'échelle.

L'algorithme de Nelder et Mead est souvent moins efficace en terme de nombre d'évaluations de la fonction objectif mais fonctionne bien dans la plupart des cas sans nécessiter le calcul du gradient. Cette méthode permet donc d'obtenir une bonne solution

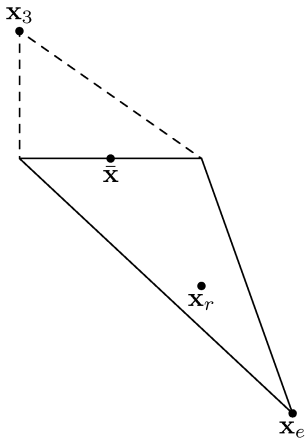
1. à ne pas confondre avec l'algorithme du simplexe en programmation linéaire.

sans trop d'effort. Dans certaines configurations, le simplexe peut dégénérer, les points ne sont alors plus affinement indépendants. On peut, dans ce cas, utiliser la méthode de Torczon qui maintient la géométrie du simplexe.

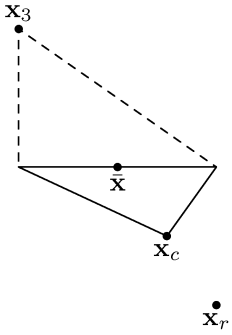
En terme d'occupation mémoire, l'algorithme nécessite une place équivalente à un algorithme de quasi-newton, c'est-à-dire $n + 1$ vecteurs de dimensions n.



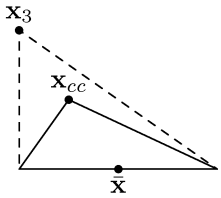
(a) Réflexion si $f(X_r) < f(X_1)$ et $f(X_r) \leq f(X_e)$ ou si $f(X_1) \leq f(X_r) < f(X_n)$



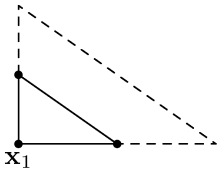
(b) Expansion si $f(X_e) < f(X_r) < f(X_1)$



(c) Contraction externe si $f(X_n) \leq f(X_r) < f(X_{n+1})$ et $f(X_c) < f(X_r)$



(d) Contraction interne si $f(X_{n+1}) \leq f(X_r)$ et $f(X_{cc}) < f(X_r)$



(e) Rétrécissement dans les autres cas

FIGURE 1 – Evolution du simplexe de Nelder et Mead (le simplexe original est en pointillé)

Algorithme 12: Algorithme de Nelder et Mead

Données : f une fonction et $n+1$ points affinements indépendants $X_1, \ldots, X_{n+1}$ (simplexe)

tant que *critère d'arrêt non satisfait* **faire**
- Renuméroter les points tels que : $f(X_1) \leq f(X_2) \leq \cdots \leq f(X_{n+1})$;
- $\bar{X} \leftarrow \frac{1}{n} \sum_{i=1}^{n} X_i$;
- $D \leftarrow \bar{X} - X_{n+1}$;
- $X_r \leftarrow X_{n+1} + 2D$;
- **si** $f(X_r) < f(X_1)$ **alors**
  - $X_e \leftarrow X_{n+1} + 3D$;
  - **si** $f(X_e) < f(X_r)$ **alors**
    - $X_{n+1} \leftarrow X_e$; `(expansion)`
  - **sinon**
    - $X_{n+1} \leftarrow X_r$; `(réflexion)`
  - **fin**
- **sinon**
  - **si** $f(X_1) \leq f(X_r) < f(X_n)$ **alors**
    - $X_{n+1} \leftarrow X_r$; `(réflexion)`
  - **sinon**
    - **si** $f(X_n) \leq f(X_r) < f(X_{n+1})$ **alors**
      - $X_c \leftarrow X_{n+1} + \frac{3}{2}D$;
      - **si** $f(X_c) < f(X_r)$ **alors**
        - $X_{n+1} \leftarrow X_c$; `(contraction externe)`
      - **sinon**
        - **pour** i *de* 2 *à* $n+1$ **faire** $X_i \leftarrow X_i + \frac{1}{2}(X_i - X_1)$; `(rétrécissement)`
      - **fin**
    - **sinon**
      - $X_{cc} \leftarrow X_{n+1} + \frac{1}{2}D$;
      - **si** $f(X_{cc}) < f(X_r)$ **alors**
        - $X_{n+1} \leftarrow X_{cc}$; `(contraction interne)`
      - **sinon**
        - **pour** i *de* 2 *à* $n+1$ **faire** $X_i \leftarrow X_i + \frac{1}{2}(X_i - X_1)$; `(rétrécissement)`
      - **fin**
    - **fin**
  - **fin**
- **fin**

fin

Résultat : x_1

3.6 Mise en pratique avec Matlab

On s'intéresse à minimiser la fonction de Rosenbrock souvent utilisée pour illustrer le comportement des différents algorithmes. La fonction de Rosenbrock est définie par :

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2 \tag{45}$$

Comme pour la minimisation monovariable, on définit la fonction objectif dans un fichier `.m`, par exemple :

```
function [f,g]=Rosenbrock(x)

f=100*(x(2)-x(1)^2)^2+(1-x(1))^2;

if nargout>1
  g=[100*(4*x(1)^3-4*x(1)*x(2))+2*x(1)-2 100*(2*x(2)-2*x(1)^2)];
end
```

`f` est la valeur de la fonction au point `x`, `g` la valeur de son gradient [2].

Les algorithmes de descente sont accessibles via la commande `fminunc` dont le paramétrage s'effectue aussi grâce la fonction `optimset`.

Pour effectuer une descente selon la plus forte pente, on écrit par exemple :

```
options=optimset('LargeScale','off','gradobj','on',...
   'HessUpdate','steepdesc','MaxFunEvals',1000,'TolFun',1e-3);

[x,fval] = fminunc('Rosenbrock',[-1.9 2],options);
```

Pour appliquer un quasi-newton avec mise à jour BFGS :

```
options=optimset('LargeScale','off','gradobj','on',...
   'MaxFunEvals',1000,'TolFun',1e-3);

[x,fval] = fminunc('Rosenbrock',[-1.9 2],options);
```

Avec la mise à jour DFP :

```
options=optimset('LargeScale','off','gradobj','on'...
   'HessUpdate','dfp','MaxFunEvals',1000,'TolFun',1e-3);

[x,fval] = fminunc('Rosenbrock',[-1.9 2],options);
```

Remarque : pour éviter d'avoir à définir le gradient de la fonction objectif, on peut demander à Matlab d'évaluer le gradient par différences finies en passant l'option `gradobj` à `off`.

Enfin, l'algorithme de Nelder et Mead est accessible via la commande `fminsearch` :

```
options=optimset('MaxFunEvals',1000,'TolFun',1e-3);

[x,fval] = fminsearch('Rosenbrock',[-1.9 2],options);
```

2. Ce calcul peut être omis si on utilise `fminsearch`.

4 Problème des moindres carrés non linéaires

4.1 Définition

Les problèmes de moindres carrés non linéaires visent à minimiser des fonctions objectifs de la forme :

$$f(X) = \frac{1}{2}\sum_{i=1}^{m} g_i^2(X) \tag{46}$$

On utilise généralement la fonction vectorielle $F : \mathbb{R}^n \mapsto \mathbb{R}^m$ définie par :

$$F(X) = \begin{pmatrix} g_1(X) \\ \vdots \\ g_m(X) \end{pmatrix} \tag{47}$$

On a donc :

$$f(X) = \frac{1}{2}F(X)^{\mathrm{T}}F(X) \tag{48}$$

4.2 Méthode de Gauss-Newton

Le gradient de f est :

$$\nabla f(X) = \sum_{i=1}^{m} g_i(X)\nabla g_i(X) = J_F(X)^{\mathrm{T}}F(X) \tag{49}$$

avec $J_F(X)$ la matrice jacobienne de F.

Le hessien de f s'écrit :

$$\nabla^2 f(X) = \sum_{i=1}^{m} \nabla g_i(X)\nabla g_i(X)^{\mathrm{T}} + \sum_{i=1}^{m} g_i(X)\nabla^2 g_i(X) \tag{50}$$

$$= J_F(X)^{\mathrm{T}}J_F(X) + \sum_{i=1}^{m} g_i(X)\nabla^2 g_i(X) \tag{51}$$

Le deuxième terme est très coûteux à calculer et peut être négligé en pratique (car il est relativement faible au voisinage d'un minimum). On approche le hessien par le premier terme uniquement qui est une matrice semi définie positive [14] :

$$R(X) = J_F(X)^{\mathrm{T}}J_F(X) \tag{52}$$

La méthode qui consiste utiliser cette approximation dans le cadre de la méthode de Newton est nommée méthode de Gauss-Newton (cf. algorithme 13).

En pratique, la direction de descente $D = -R^{-1}(X)\nabla f(X)$ est calculée sans inverser $R(X)$ mais en résolvant :

$$(R(X) + \Delta)D = -\nabla f(X) \tag{53}$$

où Δ est une matrice diagonale telle que $R(X) + \Delta$ soit définie positive afin de garantir que la direction soit une direction de descente.

Contrairement à l'algorithme de Newton, l'algorithme de Gauss-Newton est un algorithme robuste car la matrice $J_F(X)^{\mathrm{T}} J_F(X)$ est presque toujours définie positive et ainsi la direction calculée est une direction de descente [14]. On doit cependant initialiser l'algorithme en un point assez proche d'un minimiseur local.

Algorithme 13: Algorithme de Gauss-Newton

Données : f une fonction différentiable telle que $f(X) = \frac{1}{2}F(X)^{\mathrm{T}}F(X)$ et X_0 un point initial

$X \leftarrow X_0$;
tant que *critère d'arrêt non satisfait* **faire**
  $R \leftarrow J_F(X)^{\mathrm{T}} J_F$;
  Calculer D solution de $R(X) + \Delta)D = -J_F(X)^{\mathrm{T}}F(X)$;
  Trouver un pas α qui minimise $f(X + \alpha D)$;
  $X \leftarrow X + \alpha D$;
fin

Résultat : X

4.3 Méthode de Levenberg-Marquardt

Remarquant le fait que la méthode de la plus forte pente est généralement efficace loin d'un minimum là où une méthode de Newton diverge, il est intéressant d'essayer de tirer le meilleur parti des deux approches. La méthode de Levenberg-Marquardt [15, 16] utilise comme direction de descente, le vecteur D_k solution de l'équation :

$$(R(X) + \lambda_k I)D_k = -\nabla f(X) \quad \text{avec } \lambda_k > 0 \tag{54}$$

Si λ_k est nul, la direction est celle d'un algorithme de Gauss-Newton. Quand λ_k tend vers l'infini, la direction est celle de la plus forte pente.

λ_k est calculé à chaque itération et favorise la direction de la plus forte pente dans les cas où la méthode de Gauss-Newton n'est pas adaptée. On peut par exemple diminuer λ_k si tout se passe bien (la fonction objectif diminue) et l'augmenter si il y a divergence (la fonction objectif augmente).

La méthode de Levenberg-Marquardt est particulièrement robuste et efficace. Elle est devenue l'algorithme de référence pour la minimisation de problèmes de moindres carrés non linéaires.

4.4 Mise en pratique avec Matlab

Pour utiliser les méthodes des moindres carrés non linéaires sous Matlab, on doit définir la fonction vectorielle F et son jacobien dans un fichier `.m`, par exemple :

```
function [F,J]=Rosenbrock_mc(x)
F=[ 100*(x(2)-x(1)^2)^2
    (1-x(1))^2          ];

if nargout>1
   J=[ 100*(4*x(1)^3-4*x(1)*x(2)) 100*(2*x(2)-2*x(1)^2)
       2*x(1)-2                   0                     ];
end
```

Les algorithmes de Gauss-Newton et de Levenberg-Marquardt sont accessibles via la commande `lsqnonlin` dont le paramétrage s'effectue aussi grâce la fonction `optimset`.

Pour utiliser la méthode de Gauss-Newton, on écrit par exemple :

```
options=optimset('LargeScale','off','Jacobian','on',...
   'LevenbergMarquardt','off','MaxFunEvals',1000,'TolFun',1e-3);

[x,fval,residual,exitflag,output] = ...
   lsqnonlin('Rosenbrock_mc',x,[],[],options);
```

Pour utiliser la méthode de Levenberg-Marquardt (algorithme par défaut) :

```
options=optimset('LargeScale','off','Jacobian','on',...
   'MaxFunEvals',1000,'TolFun',1e-3);

[x,fval,residual,exitflag,output] = ...
   lsqnonlin('Rosenbrock_mc',x,[],[],options);
```

4.5 Méthodes récursives

Dans certain cas, on ne dispose pas de tous les coûts partiels $\frac{1}{2}g_i^2(X)$ de la fonction objectif. C'est notamment le cas de l'identification en ligne des paramètres d'un système physique. A chaque instant k, on dispose de nouvelles données permettant de calculer un nouveau coût partiel $\frac{1}{2}g_k^2(X)$. On définit alors :

$$f_k(X) = \sum_{i=1}^{k} \frac{1}{2} g_i^2(X) \tag{55}$$

On souhaite trouver un minimiseur X_k^* à chaque instant k :

$$X_k^* = \arg\min_X f_k(X) \tag{56}$$

On utilise alors une méthode permettant de prendre en compte le nouveau coût partiel dans le calcul du minimiseur en effectuant un minimum de calculs. L'objectif est de parvenir à une mise à jour du minimiseur de la forme :

$$X_k^* = X_{k-1}^* + \delta_k(g_k) \tag{57}$$

Ce genre de méthode est qualifié de récursif, car on utilise le minimiseur X_{k-1}^* minimisant f_{k-1} pour calculer le nouveau minimiseur X_k^* minimisant f_k.

4.5.1 Méthode du gradient stochastique

La méthode du gradient stochastique (appelé aussi gradient incrémental) est une méthode récursive du premier ordre introduite par Widrow et Hoff [21].

L'idée est de trouver une formulation récursive du gradient :

$$\nabla f_k(X) = \sum_{i=1}^{k} g_i(X)\nabla g_i(X) \tag{58}$$

$$= \nabla f_{k-1}(X) + g_k(X)\nabla g_k(X) \tag{59}$$

Partant de X_{k-1}, on effectue une descente selon la plus forte pente pour trouver X_k, on écrit donc :

$$X_k = X_{k-1} - \alpha_k \nabla f_k(X_{k-1}) \tag{60}$$

$$= X_{k-1} - \alpha_k[\nabla f_{k-1}(X_{k-1}) + g_k(X_{k-1})\nabla g_k(X_{k-1})] \tag{61}$$

En supposant que X_{k-1} minimisait bien f_{k-1} à l'instant $k-1$, on a $\nabla f_{k-1}(X_{k-1}) = 0$ et donc :

$$X_k = X_{k-1} - \alpha_k g_k(X_{k-1})\nabla g_k(X_{k-1}) \tag{62}$$

La convergence de cette méthode n'est pas toujours assurée et est en pratique très lente. Néanmoins, des conditions de convergence sont disponible dans [1].

De nombreuses méthodes d'adaptation du pas ont été proposées (voir par exemple [6]). Une autre modification populaire de la méthode du gradient stochastique consiste à ajouter un terme inertiel dans la mise à jour :

$$X_k = X_{k-1} - \alpha_k g_k(X_{k-1})\nabla g_k(X_{k-1}) + \beta_k(X_{k-1} - X_{k-2}) \tag{63}$$

4.5.2 Version récursive de la méthode de Gauss-Newton

L'idée est de trouver des formulations récursives du gradient et du hessien. On note V_k le gradient de la fonction objectif f_k et R_k son hessien. On a :

$$V_k = \sum_{i=1}^{k} g_i(X_k)\nabla g_i(X_k) \tag{64}$$

$$= V_{k-1} + g_k(X_k)\nabla g_k(X_k) \tag{65}$$

Si on suppose que X_{k-1} minimisait bien f_{k-1} à l'instant $k-1$, alors $V_{k-1} = 0$ et donc :

$$V_k = g_k(X_k)\nabla g_k(X_k) \tag{66}$$

Pour le hessien (approché par Gauss-Newton), il vient :

$$R_k = \sum_{i=1}^{k} \nabla g_i(X_k) \nabla g_i(X_k)^{\mathrm{T}} \tag{67}$$

$$= R_{k-1} + \nabla g_k(X_k) \nabla g_k(X_k)^{\mathrm{T}} \tag{68}$$

On trouve alors un minimiseur X_k de f_k en effectuant une mise à jour de type Newton (ce qui suppose que l'on est assez près du minimum pour l'atteindre en un coup...) :

$$X_k = X_{k-1} - R_k^{-1} V_k \tag{69}$$

En pratique pour éviter l'inversion de R_k à chaque fois, on introduit :

$$P_k = R_k^{-1} \tag{70}$$

puis on utilise le lemme d'inversion qui stipule que :

$$[A + BCD]^{-1} = A^{-1} - A^{-1}B[DA^{-1}B + C^{-1}]^{-1}DA^{-1} \tag{71}$$

En posant $A = R_{k-1}$, $B = D^T = \nabla g_k(X)$ et $C = 1$, on obtient la mise à jour suivante :

$$P_k = P_{k-1} - \frac{P_{k-1} \nabla g_k(X_k) \nabla g_k(X_k)^{\mathrm{T}} P_{k-1}}{1 + \nabla g_k(X_k)^{\mathrm{T}} P_{k-1} \nabla g_k(X_k)} \tag{72}$$

De nombreuses variantes de cet algorithme ont été proposées (avec facteur d'oubli, avec réinitialisation).

Algorithme 14: Version récursive de l'algorithme de Gauss-Newton

Données : f une fonction différentiable telle que $f(X) = \frac{1}{2} F(X)^{\mathrm{T}} F(X)$ et X_0 un point initial

1 $P_0 = cI$ avec c un grand nombre (typiquement entre 10^4 et 10^8);
2 $k = 1$;
3 **tant que** *critère d'arrêt non satisfait* **faire**
4 Recueillir g_k;
5 $P_k = P_{k-1} - \frac{P_{k-1} \nabla g_k(X_k) \nabla g_k(X_k)^{\mathrm{T}} P_{k-1}}{1 + \nabla g_k(X_k)^{\mathrm{T}} P_{k-1} \nabla g_k(X_k)}$;
6 $V_k = g_k(X_k) \nabla g_k(X_k)$;
7 $X_k = X_{k-1} - P_k V_k$;
8 $k = k + 1$;
9 **fin**

Résultat : X_{k-1}

Références

[1] Dimitri Bertsekas. *Nonlinear Programming : 2nd Edition*. Athena Scientific, 1999. 24

[2] Michel Bierlaire. *Introduction à l'optimisation différentiable*. Presses polytechniques et universitaires romandes, 2006. 7, 11

[3] Richard P. Brent. *Algorithms for Minimisation Without Derivatives*. Prentice Hall, 1973. 5

[4] C. G. Broyden. The convergence of a class of double-rank minimization algorithms 2 : the new algorithm. *Journal of the institute of Mathematics and its Applications*, 6 :222–231, 1970. 15

[5] Thomas Coleman, Mary Ann Branch, and Andrew Grace. *Optimization Toolbox User's Guide*. The Mathworks Inc., 1999. 7, 10

[6] Christian Darken, Joseph Chang Z, and John Moody. Learning rate schedules for faster stochastic gradient search. In *Proc. of Advances in Neural Information Processing Systems*. IEEE Press, 1992. 24

[7] W. C. Davidon. Variable metric method for minimization. Technical Report ANL-5990, A.E.C. Research and Development Report, 1959. 15

[8] R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13 :317–322, 1970. 15

[9] R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *Computer Journal*, 6 :163–168, 1963. 15

[10] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Computer Journal*, 7 :149–154, 1964. 13

[11] D. Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computation*, 24 :23–26, 1970. 15

[12] Anatoli Juditsky. *Cours d'optimisation*. Magistère de Mathématiques, Université Joseph Fourier, Grenoble. http ://ljk.imag.fr/membres/Anatoli.Iouditski/, consulté en septembre 2006. 11

[13] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the nelder-mead simplex algorithm in low dimensions. *SIAM Journal on Optimization*, 9(1) :112–147, 1998. 16

[14] Gérard Lebourg. *Cours d'optimisation*. Licence Génie Mathématique et Informatique, Université Paris-Dauphine. http ://cours.ufrmd.dauphine.fr/lebourg/optiiup1.html, consulté en septembre 2007. 21, 22

[15] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quartely Journal of Applied Mathematics II*, 2(2) :164–168, 1944. 22

[16] D. W. Marquardt. An algorithm for least squares estimation of non-linear parameters. *Journal of the Society of Industrial and Applied Mathematics*, 11(2) :431–441, 1963. 22

[17] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7 :308–313, 1965. 16

[18] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes : The Art of Scientific Computing*. Cambridge University Press, 2007. 5, 7, 10

[19] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24 :647–656, 1970. 15

[20] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994. Available at http ://www.cs.cmu.edu/˜jrs/jrspapers.html. 11

[21] B. Widrow and M. E. Hoff. Adaptive switching circuits. *IRE Wescon Convention Record*, 4 :96–104, 1960. Reprinted in Anderson & Rosenfeld (eds), Neurocomputing : foundations of research, MIT Press, 1988. 24