Chapitre II : Optimisation sans contraintes - méthodes locales

II.1 Méthodes de recherche unidimensionnelle

Il existe deux formes de d'optimisation

•*Minimisation exacte*

On cherche à trouver un minimum local *x** avec une précision donnée

- réduction itérative de l'intervalle de recherche : dichotomie
- réduction optimale : Fibonacci, nombre d'or

•*Minimisation approchée*

On cherche une réduction suffisante de la fonction sans déterminer précisément le minimum x*

- règles d'acceptation (Armijo, Goldstein, Wolfe)
- limitation du nombre d'évaluations de la fonction

II.1.1 Minimisation exacte

a- Recherche par dichotomie

Théorème :

Soit $f : \mathbb{R} \to \mathbb{R}$ telle que *f* est unimodale sur l'intervalle $[a, b]$: Soient $\alpha, \beta \in [a, b]$ tels que $\alpha < \beta$

1- Si $f(\alpha) \leq f(\beta)$; alors *f* est unimodale sur l'intervalle $[a, \beta]$

2- Si $f(\alpha) > f(\beta)$; alors *f* est unimodale sur l'intervalle $[\alpha, b]$

Puisque $\alpha < \beta$ et que *f* est strictement décroissante, alors on aura nécessairement $f(\alpha) > f(\beta)$

➢ Fonction unimodale

Soit $f : \mathbb{R} \to \mathbb{R}$ est dite unimodale sur l'intervalle $[a, b]$; s.il existe $\alpha^* \in]a, b[$ unique tel que :

a) *f* est strictement décroissante sur $[a, \alpha^*]$

b) *f* est strictement croissante sur $[\alpha^*, b]$

L'intervalle $[a, b]$ s'appelle intervalle d'unimodalité associé à la fonction *f*

Si *f* est unimodale sur l'intervalle $[a, b]$; alors *f* admet une solution minimale unique $\alpha^* \in]a, b[$ et *f* est strictement décroissante sur $[a, \alpha^*]$, et strictement croissante sur $[\alpha^*, b]$.

L'allure de la fonction *f* n'est pas connue. On suppose qu'il existe un minimum unique dans l'intervalle de recherche $[a, d]$

• On évalue la fonction aux extrémités *a* et *d* : $\to$ *f(a) , f(d)*

• On évalue la fonction en 2 points *b* et *c* : $a < b < c < d, \to f(b), f(c)$

• On conserve soit l'intervalle $[a, c]$, soit l'intervalle *[b, d]*

➢ Réduction optimale de l'intervalle

•On cherche *b* et *c* pour que l'intervalle restant soit le plus petit possible. La taille de l'intervalle restant ne doit pas dépendre du choix de *[a, c]* ou *[b, d]*.

•On note : $\Delta_1 = d - a,$

$\Delta_2 = c - a = d - b$

$\Rightarrow \ a + d = c + b$

$\Rightarrow \frac{a+d}{2} = \frac{c+b}{2}$

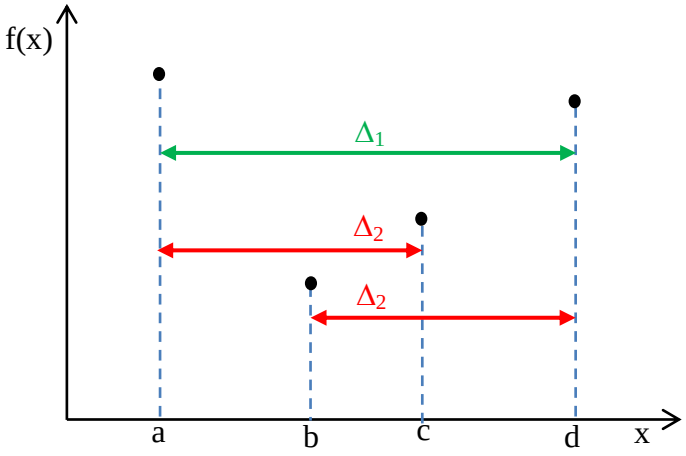

Figure 8 : conserve de l'intervalle $[a, c]$ ou l'intervalle [b, d]

→ Les points *b* et *c* doivent être symétriques par rapport au milieu de l'intervalle *[a, d]*.
On suppose que l'intervalle restant est *[a, c]*.

•Pour réutiliser le point *b* à l'itération suivante, on choisit le nouveau point *e* symétrique de *b* par rapport au milieu de l'intervalle *[a, c]*.

$\Delta_1 = d - a,$
$\Delta_2 = c - a = d - b,$
$\Delta_3 = b - a = c - e$
$d - a = (d - b) + (b - a)$
$\Rightarrow \ \Delta_1 = \Delta_2 + \Delta_3$

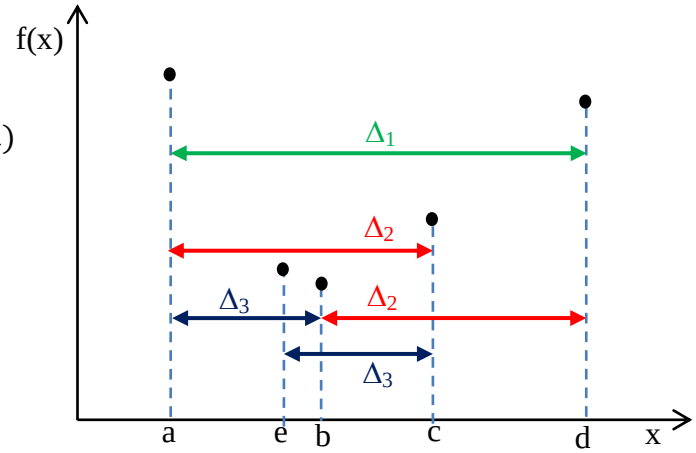

Figure 9 : Réduction optimale de l'intervalle

→ Les longueurs Δ_k des intervalles successifs vérifient :

$$\Delta_k = \Delta_{k+1} + \Delta_{k+2}$$

Après un nombre *N* d'itérations, on obtient un intervalle de longueur Δ_{N-1}.
•On définit la suite de nombres F_k par : $\Delta_k = F_{N-k}\Delta_{N-1}$ pour $k = 1, \dots, (N-1)$
•La suite $(F_n)_{n=1,\dots,(N-1)}$ vérifie

$$\Delta_k = \Delta_{k+1} + \Delta_{k+2} \Rightarrow \frac{\Delta_k}{\Delta_{N-1}} = \frac{\Delta_{k+1}}{\Delta_{N-1}} + \frac{\Delta_{k+2}}{\Delta_{N-1}}$$
$$\Rightarrow F_{N-k} = F_{N-k-1} + F_{N-k-2}$$
$$\Rightarrow F_n = F_{n-1} + F_{n-2} \text{ pour } n = N-k,\ n = 3, 4, \dots, (N-1)$$

$\Delta_{N-1} = F_1\Delta_{N-1}$ pour $k = N-1$ $\Rightarrow F_1$

•La suite est complètement déterminée par la valeur de F_2

$$\begin{cases} F_1 = 1 \\ F_2 \text{ à } choisir \\ F_n = F_{n-1} + F_{n-2} \text{ pour } n = 3, 4, \dots, (N-1) \end{cases}$$

b- Méthode de Fibonacci

La méthode de *Fibonacci* est un algorithme d'optimisation, c'est-à-dire de recherche de l'extremum d'une fonction, dans le cas d'une fonction *unimodale*, c'est-à-dire dans lequel l'extremum global recherché est le seul extremum local. S'il existe plusieurs extrema locaux, l'algorithme donne un extremum local, sans qu'il soit garanti que ce soit l'extremum absolu. Cet algorithme, ainsi que la méthode du *nombre d'or,* ont été mis au point par le statisticien ***Kiefer*** (1953).

• Pour un nombre *N* d'itérations fixé, on choisit F_2 pour que Δ_{N-1} soit minimal.

$\Delta_{N-1} = \frac{\Delta_{N-1}}{F_{N-1}}$ minimal $\Rightarrow$ F_{N-1} maximal $\Rightarrow$ F_2 maximal

• Valeur maximale de F_2 : $\Delta_k \geq \frac{1}{2}\Delta_{k-1} \Rightarrow \Delta_{N-1} \geq \frac{1}{2}\Delta_{N-2} \Rightarrow F_1 \geq \frac{1}{2}F_2 \Rightarrow F_{2max} = 2$

• On obtient la *suite de Fibonacci*: 1, 2, 3, 5, 8, 13, 21, 34, 55,… (Voir dans la nature : fleur de tournesol, fruit d'ananas,…,)

La méthode de Fibonacci donne le nombre minimal d'itérations N pour obtenir la solution avec une *précision donnée*.

• La disposition des premiers points *b* et *c* dépend de $N : \frac{\Delta_1}{\Delta_2} = \frac{F_{N-1}}{F_{N-2}} \rightarrow \Delta_2$

c- Méthode du nombre d'or

❖ **Définition**

La méthode de *Fibonacci* nécessite de changer la disposition des points à chaque itération. La disposition des points n'est optimale que pour une précision donnée. La méthode du *nombre d'or* est plus générale et plus simple à mettre en œuvre.

• On impose un rapport de réduction fixe de l'intervalle à chaque itération.

$$\frac{\Delta_1}{\Delta_2} = \frac{\Delta_2}{\Delta_3} = \cdots \frac{\Delta_k}{\Delta_{k+1}} = \frac{\Delta_{k+1}}{\Delta_{k+2}} = \cdots = \gamma \text{ avec } \Delta_k = \Delta_{k+1} + \Delta_{k+2}$$

$$\Rightarrow \frac{\Delta_k}{\Delta_{k+1}} = 1 + \frac{\Delta_{k+2}}{\Delta_{k+1}} \Rightarrow \gamma = 1 + \frac{1}{\gamma} \Rightarrow \gamma^2 - \gamma - 1$$

• On obtient pour le rapport γ le nombre d'or : $\gamma = \frac{1+\sqrt{5}}{2} \approx 1.618033988$

Méthode du nombre d'or (ou de la section dorée)

❖ **Optimalité**

• La méthode du nombre d'or n'est pas optimale pour un nombre d'itérations donné *N*.

• Pour un nombre d'itérations grand : $\lim_{N\to\infty} \frac{F_N}{F_{N-1}} = \gamma$

La disposition des points de Fibonacci tend vers celle du nombre d'or.

d- Les algorithmes de recherche itérative

Un algorithme itératif est d´défini par une application vectorielle $f : \mathbb{R} \rightarrow \mathbb{R}$ qui génère une suite de champs de vecteurs (x_k), $k \geq 0$ par une construction typiquement de la forme

Choisir x_0 (phase d'initialisation de l'algorithme)
Calculer $x_{k+1} = f(x_k)$, ($k^{ième}$ itération)

1. Algorithme de recherche dichotomique sans dérivées

La méthode de dichotomie est simple. Elle se base sur le théorème précèdent. Le choix des points α_k et β_k se fait de la façon suivante :

Recherche dichotomique sans dérivées

Initialisation : Fonction objective *f(x)* $\varepsilon > 0$, l : longueur finale de l'intervalle d'incertitude, $[a_1, b_1]$ Intervalle initiale
Poser $k = 1$ et aller à l'étape 1
Etape 1
Si $b_k - a_k < l$ **stop**, Le minimum appartient à $[a_k, b_k]$
Sinon poser :

$$\alpha_k = \frac{a_k + b_k}{2} - \varepsilon$$

$$\beta_k = \frac{a_k + b_k}{2} + \varepsilon$$

Et aller à l'étape 2
Etape 2 :
Si $f(\alpha_k) > f(\beta_k)$ **Poser** $a_{k+1} = \alpha_k$ et $b_{k+1} = b_k$
Sinon posez $a_{k+1} = a_k$ et $b_{k+1} = \beta_k$
$k = k - 1$ et aller à l'étape 1

2. Algorithme de recherche Fibonacci (Fibonacci search)

L'algorithme de Fibonacci se base sur le choix de x_1 et x_2 de telle sorte qu'après avoir conservé le triplet a, x_1, x_2 (ou encore le triplet x1, x2, b si $f(x_2) < f(x_1)$), le point x_1 joue le rôle d'un des deux points intermédiaires dans le nouveau intervalle $[a, b]$. Ceci évite de recalculer $f(x_1)$. Le principe est le suivant. On dénote $d^k = b^k - a^k$, de sorte que $d^{k+1} = b^k - x_1^k = x_2^k - a^k$ et $d^{k+2} = x_1^k - a^k = b^k - x_2^k$ les longueurs des intervalles de trois itérations consécutives. Alors, $d^k = d^{k+1} + d^{k+2}$ et on reconnaît la récurrence inverse de celle des nombres de Fibonacci.

Recherche de Fibonacci (Fibonacci search)

Initialisation : Fonction objective *f(x)* $\varepsilon > 0$, $[a_1, b_1]$ Intervalle initiale
F_1=2 F_2=3
n=2
Tant que $b - a \geq \varepsilon$ **faire**
$d = b - a$

$$x_1 = b - d\frac{F_{n-1}}{F_n}$$

$$x_2 = a + d\frac{F_{n-1}}{F_n}$$

Si $f(x_2) \leq f(x_1)$ **alors**
$b = x_2$
Sinon
$a = x_1$
Fin si
$n = n + 1$

$F_n = F_{n-1} + F_{n-2}$
Fin tant que

Exemple de programmation sur Matlab :

```
function y=f(x)
y=5*x.^2-3*x+4;
endfunction
a=0; b=3;epsilon=0.5;
F(1)=2; F(2)=3;
n=2;
while b-a>=epsilon do
   d=b-a;
   x(1)=b-d*(F(n-1)/F(n));
   x(2)=a+d*(F(n-1)/F(n));
if f(x(1))<=f(x(2)) then
   b=x(2)
else
   a=x(1)
end
n=n+1;
F(n)=F(n-1)+F(n-2);
end
```

3. Algorithme de recherche du nombre d'or ou de section d'orée (Golden section search)

L'algorithme de la section d'orée est une méthode simple mais efficace, n'utilisant pas de dérivée, pour trouver l'optimum d'une fonction unimodale à une dimension.

Recherché de la section d'orée (Golden section search)

Initialisation : Fonction objective *f(x)* $\varepsilon > 0$, $[a_1, b_1]$ Intervalle initiale
$d = b - a$
Tant que $b - a \geq \varepsilon$ **faire**
$d = 0.618 \times d$
$x_1 = b - d$
$x_2 = a + d$
Si $f(x_2) \leq f(x_1)$ **alors**
$b = x_2$
Sinon
$a = x_1$
Fin si
Fin tant que

Exemple de programmation sur Matlab :

```
function y=f(x)
y=5*x.^2-3*x+4;
endfunction
a=0; b=3;epsilon=0.5;
d=b-a;
while b-a>=epsilon do
   d=0.618*d;
```

```
   x(1)=b-d;
   x(2)=a+d;
if f(x(1))<=f(x(2)) then
   b=x(2)
else
   a=x(1)
end
end
```

II.1.2 Minimisation approchée

On cherche une réduction suffisante de la fonction sans déterminer précisément le minimum x*, les méthodes approchée ou inexactes utilisées dans ces formes de minimisation sont des méthodes à direction de descente et recherche linéaire

a- Principes

• Il n'est pas utile de réaliser une minimisation exacte suivant la direction de descente :
- Elle nécessite un grand nombre d'évaluations de la fonction
- Elle n'apporte pas une amélioration significative loin de la solution

• On peut se contenter d'une minimisation approchée ou inexacte
- 2 *règles d'acceptation* d'un pas de déplacement

Soit $f: \mathbb{R} \to \mathbb{R}$: On considère le problème de minimisation sans contraintes (*PSC*) suivant :

$$min\{f(x), x \in \mathbb{R}^n\}$$

On suppose qu'à l'itération *k* on ait un point $x_k \in \mathbb{R}^n$ et une direction de descente $d_k \in \mathbb{R}^n$; et que la direction d_k vérifié

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

Ainsi la variation de la fonction *f* dans la direction de descente d_k est donnée par la formule de Taylor a l'ordre 1

$$f(x_k + \alpha_k d_k) = f(x_k) + \alpha_k \nabla f(x_k)^T d_k$$

$$x_{k+1} = x_k + \alpha_k d_k, \quad \alpha_k \geq 0$$

Ou α_k représente le pas de descente dans la direction d_k (à partir du point x_k). Le choix de d_k et α_k caractérisent l'algorithme d'optimisation.
On constate que si d_k est une direction de descente alors, pour α_k suffisamment petit, on a

$$f(x_k + \alpha_k d_k) < f(x_k)$$

- ***Définition***

Un pas α est dit admissible pour une direction suffisamment descendante *d* lorsqu'il satisfait aux deux inégalités suivantes, nommées critère d'*Armijo* et de *Wolfe* respectivement :

On voit que les pas qui satisfont aux conditions de la définition constituent un ensemble d'intervalles dans lesquels *f* est assez petite alors que ses dérivées directionnelles sont assez grandes

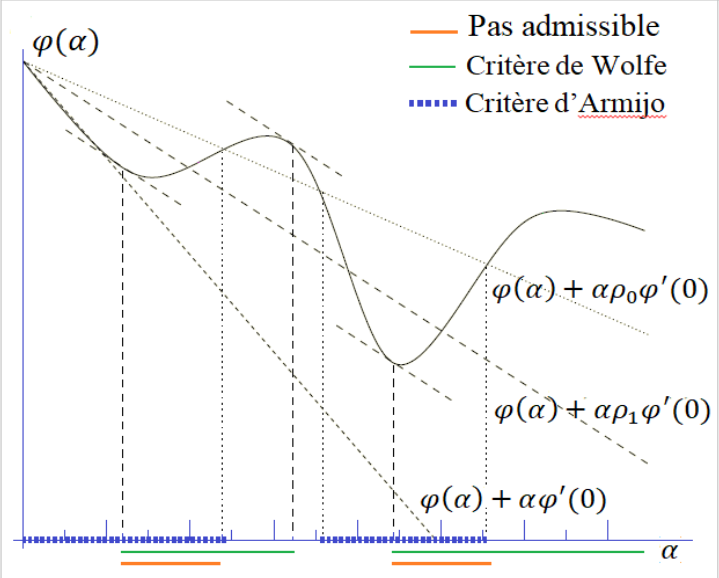

Figure 10 : Pas admissible

- ***Règles d'acceptation du pas***

• Diminution suffisante de la valeur de la fonction f c'est la condition d'***Armijo*** (1ère condition de ***Wolfe***)
• Déplacement suffisant par rapport au point initial c'est la condition de ***Goldstein*** (2ème condition de ***Wolfe***)

b- La recherche linéaire d'Armijo

- ***Définition***

Soient $f:\mathbb{R}^n \to \mathbb{R}$ une fonction différentiable, $x_k \in \mathbb{R}^n$, $d_k \in \mathbb{R}^n$ une direction de descente, vérifiant : $\nabla f(x_k)^T d_k < 0$. On dit que le pas $\alpha_k > 0$ vérifie la condition d'Armijo, si on a

$$f(x_k + \alpha_k d_k) \le f(x_k) + \alpha_k \rho_1 \nabla f(x_k)^T d_k,\ \rho_1 \in]0,1[$$

Si on note $\varphi_k(\alpha) = f(x_k + \alpha_k d_k)$
Alors on a $\varphi_k(0) = f(x_k)$
Et $\varphi_k'(\alpha) = \nabla f(x_k + \alpha_k d_k)^T d_k$
Ainsi $\varphi_k'(0) = \nabla f(x_k)^T d_k < 0$
Et par conséquent $\varphi_k(\alpha_k) \le \varphi_k(0) + \alpha_k \rho_1 \varphi_k'(0),\ \rho_1 \in]0,1[$

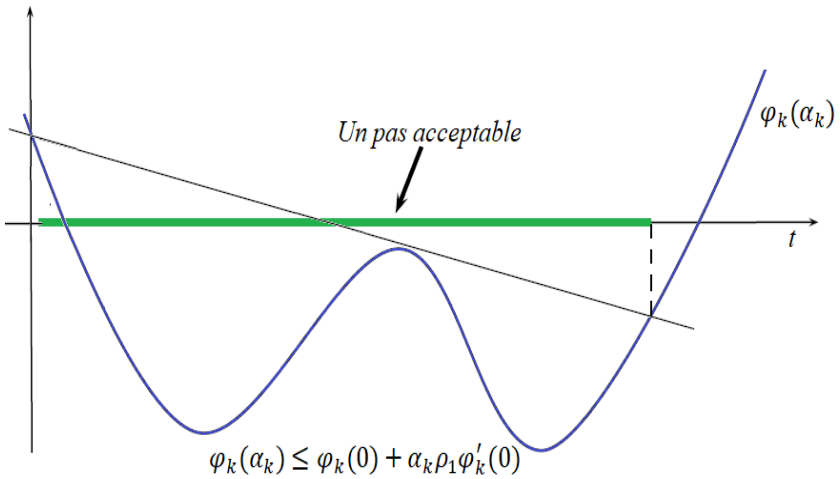

Figure 11 : La sélection du pas selon la règle d'*Armijo*.

- ***Algorithme (la règle d'Armijo)***

Algorithme (la règle d'Armijo)
Etape initiale : choisir un point initial $\alpha^* \in]0,\infty[$ et $\rho_1 \in]0,1[$. Calculer $\varphi_k(\alpha^*) = f(x_k + \alpha^* d_k)$, $\varphi_k(0) = f(x_k)$, $\varphi'_k(0) = \nabla f(x_k)^T d_k < 0$ et allez à Etape principale **Etapes principales** : **Etape 1** : Si $\varphi_k(2^n\alpha^*) \leq \varphi_k(0) + \alpha^*\rho_1\varphi'_k(0)$, aller à l'étape 2 Sinon aller à l'étape 3 **Etape 2 :** Choisir le plus grand entier $n \geq 0$ tel que : $\varphi_k(\alpha^*) \leq \varphi_k(0) + 2^n\alpha^*\rho_1\varphi'_k(0)$, et prendre $\alpha_k = 2^n\alpha^*$ **Etape 3 :** Choisir le plus petit entier $m \geq 0$ tel que : $\varphi_k(\frac{\alpha^*}{2^m}) \leq \varphi_k(0) + \frac{\alpha^*}{2^m}\alpha^*\rho_1\varphi'_k(0)$, et prendre $\alpha_k = \frac{\alpha^*}{2^m}$

c- La recherche linéaire de Goldstein

- ***Définition***

La règle de *Goldstein* s'applique lorsque le gradient de la fonction ne peut être évalué (ou est trop couteux à obtenir). En ajoutant une deuxième inégalité a la règle d'*Armijo* on obtient la règle de *Goldstein* & *Price*.

- ***Principe***

On suppose qu'à l'itération k, on ait le point $x_k \in \mathbb{R}^n$, le vecteur de descente $d_k \in \mathbb{R}^n$ $(\nabla f(x_k)^T d_k < 0)$. Etant données deux réels δ et σ tels que $0 < \delta < \sigma < 1$. On cherche le pas $\rho_k > 0$ de *Goldstein* vérifiant les deux conditions suivantes :

$$f(x_k + \rho_k d_k) \leq f(x_k) + \delta\rho_k\nabla f(x_k)^T d_k$$

$$f(x_k + \rho_k d_k) \geq f(x_k) + \sigma\rho_k\nabla f(x_k)^T d_k$$

Autrement dit :

$$\varphi_k(\rho_k) \leq \varphi_k(0) + \delta\rho_k\varphi'_k(0)$$

On pose $\sigma = (1-\delta)$: $\quad \varphi_k(\rho_k) \geq \varphi_k(0) + (1-\delta)\rho_k\varphi'_k(0)$

- ***Algorithme (la règle de Goldstein & Price)***

Algorithme (la règle de Goldstein & Price)
Etape initiale : On dispose de $\alpha_g = 0$, $\alpha_d = 0$ une valeur maximale quelconque, et soit $\alpha \in]\alpha_g, \alpha_d[$, $\rho \in]0,1[$, et $\delta \in]\rho, 1[$, poser $k = 1$, et aller à l'étape 1 **Etapes principales** : **Etape 1** : Calculer $\varphi_k(\alpha) = f(x_k + \alpha d_k)$ Si $\varphi_k(\alpha_k) \leq \varphi_k(0) + \rho\alpha_k\varphi'_k(0)$ alors aller à l'étape 2 Sinon prendre $\alpha_d = \alpha$ et aller à l'étape 4 **Etape 2** : Si $\varphi_k(\alpha_k) \geq \varphi_k(0) + \delta\alpha_k\varphi'_k(0)$ stop Sinon aller à l'étape 3 **Etape 3** : poser $\alpha_g = \alpha$ **Etape 4** : rechercher un nouveau $\alpha \in]\alpha_g, \alpha_d[$ et retourner à l'etape2.

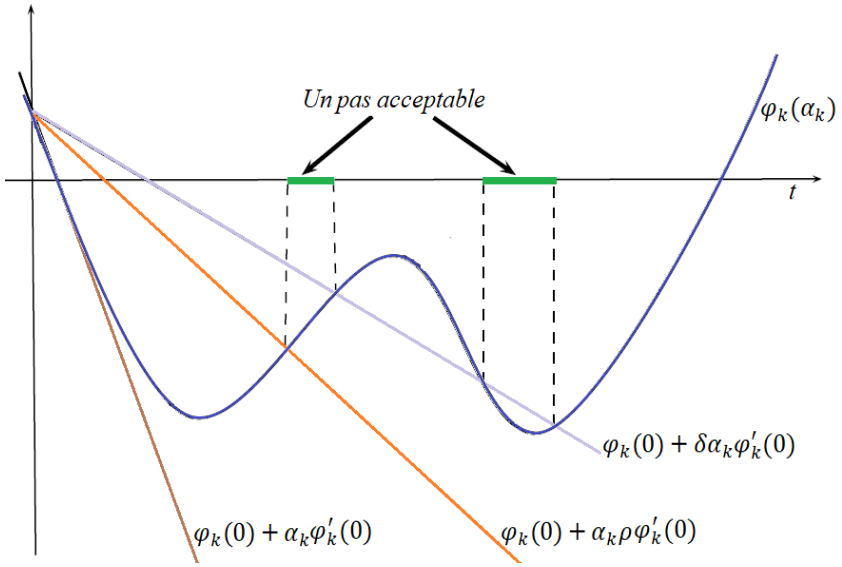

Figure 12 : La sélection du pas selon la règle de *Goldstein*.

d- La recherche linéaire de Wolf

- ***Définition***

La règle de Wolfe fait appel au calcul de $\varphi'_k(\alpha_k)$, elle est donc en théorie plus couteuse que la règle de Goldstein & Price. Cependant dans de nombreuse application, le calcul du $\nabla f(x_k)$ représente un faible cout additionnel en comparaison du coute d'évaluations de $f(x_k)$ c'est pourquoi cette règle est très utilisée

Etant données deux réels ρ et δ tels que $0 < \rho < \delta < 1$, α_k est acceptable par la recherche linéaire inexacte de wolfe si α_k vérifié les deux conditions suivantes :

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \rho\alpha_k \nabla f(x_k)^T d_k$$

$$\nabla^T f(x_k + \alpha_k d_k) d_k \geq \delta \nabla f(x_k)^T d_k$$

Autrement dit

$$\varphi_k(\alpha_k) \leq \varphi_k(0) + \delta\rho_k\varphi'_k(0)$$

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

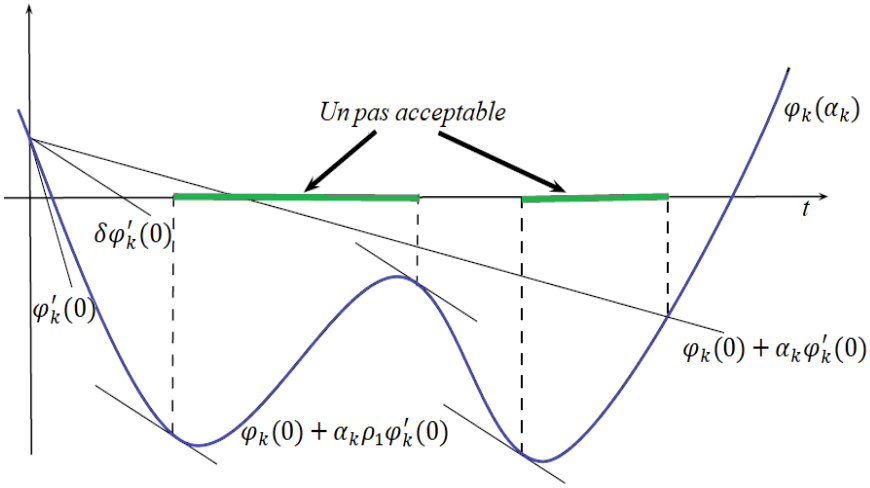

Figure 13 : La sélection du pas selon la règle de *wolfe*.

- ***Algorithme (la règle de Wolfe)***

Algorithme (la règle de Goldstein & Price)
Etape initiale : On dispose de $\alpha_g = 0,\ \alpha_d = 0$ une valeur maximale quelconque, et soit $\alpha \in]\alpha_g, \alpha_d[$, $\rho \in]0, 1[$, et $\delta \in]\rho, 1[$, poser $k = 1$, et aller à l'étape 1 **Etapes principales** : **Etape 1** : Calculer $\varphi_k(\alpha) = f(x_k + \alpha d_k)$ Si $\varphi_k(\alpha_k) \leq \varphi_k(0) + \rho\alpha_k\varphi'_k(0)$ alors aller à l'étape 2 Sinon prendre $\alpha_d = \alpha$ et aller à l'étape 4 **Etape 2** : calculer $\varphi'_k(\alpha_k) = \nabla^T f(x_k + \alpha_k d_k)d_k$ Si $\varphi'_k(\alpha_k) \geq \delta\varphi'_k(0)$ stop Sinon aller à l'étape 3 **Etape 3** : poser $\alpha_g = \alpha$ **Etape 4** : rechercher un nouveau $\alpha \in]\alpha_g, \alpha_d[$ et retourner à l'etape 2.

II.2 Méthodes du gradient

a- Principe

La méthode du gradient fait partie des classes de méthodes de direction de descente. Considérons un point de départ x_k et cherchons à minimiser une fonction *f*. Puisque l'on veut atteindre x^* , nous cherchons à avoir : $f(x_1) < fx_2)$. Une forme particulièrement simple est de chercher x_1 tel que le vecteur $x_1 - x_0$ soit colinéaire à une direction de descente $d_0 \neq 0$. On le noterons : $x_1 - x_0 = \rho_0 d_1$, où ρ_0 représente le pas de descente de la méthode. On peut donner alors x_k, d_k et ρ_k et arriver à x_{k+1} par

Choisir x_k
Calculer $x_{k+1} = x_k + \rho_k d_k$

Avec $d_k \in \mathbb{R}^{n*}$ et $\rho_k \geq 0$

Pour une fonction différentiable $f : \mathbb{R}^n \to \mathbb{R}$, le gradient $\nabla f(x_0)$ de *f* au point x_0 se définit comme suit :

$$f(x_0 + \Delta x) - f(x_0) = \left(\nabla f(x_0)\right)^T \Delta x + R(\|\Delta x\|)$$

Où $$\lim_{\|\Delta x\| \to 0} \frac{R(\|\Delta x\|)}{\|\Delta x\|} = 0$$

Ainsi

$$x_{k+1} = x_k - \rho_k \nabla f(x_k), \quad \rho_k \geq 0$$

$d_k = -\nabla f(x_k)$ Indique la direction avec le plus *grand taux de décroissance* de *f* au point x_k

b- Méthode de la descente la plus rapide ou de la plus forte pente (steepest descent)

La direction de descente «naturelle» est celle du gradient égale à la plus forte dérivée directionnelle

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

Elle est caractérisée par :
- Comportement en zigzag

- Convergence très lente → méthode inefficace en général
- Améliorations possibles → gradient accéléré, méthode de NESTEROV, pré-conditionnement

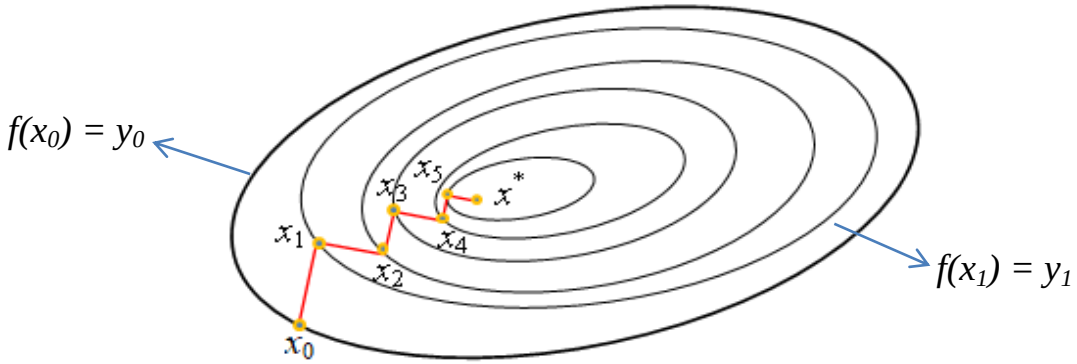

Figure 14 : Descente la plus forte pente

Avec $y_1 < y_0$

$$x_{k+1} = x_k - \rho_k \nabla f(x_k), \quad \rho_k \geq 0$$

$$\rho_k = arg \min_{\rho \geq 0} f(x_k - \rho \nabla f(x_k))$$

Donc $$(x_2 - x_1) \perp (x_1 - x_0)$$

D'une manière générale : $$(x_{k+1} - x_k) \perp (x_k - x_{k-1})$$

Critères d'arrêt

En théorie si $\nabla f(x_k) = 0$

En applique le critère d'arrêt avant car la convergence est trop lente et on a le choix entre ces trois conditions :

1- $\|\nabla f(x_k)\| < \varepsilon_1$

2- $|x_{k+1} - x_k| < \varepsilon_2$

3- $\left|\frac{f(x_{k+1}) - f(x_k)}{f(x_k)}\right| < \varepsilon_3$

c- Algorithme de descente de gradient

Algorithme de descente de gradient (steepest descent)

Choisir : Un point initial $x_0 \in \mathbb{R}^n$, un seuil de tolérance $\varepsilon > 0$
Résultat : Un point $x \in \mathbb{R}^n$ proche de x^* ?
Initialiser *x* :
$x = x_0$;
$k = 0$;
tant que $\|\nabla f(x_k)\| > \varepsilon$ **faire**
1. **Calculer** $d_k = -\nabla f(x_k)$
2. **Déterminer** le pas de descente $\rho_k > 0$ par la méthode exacte ou inexacte
3. **Mettre à jour** *x* :
$x_{k+1} = x_k + \rho_k d_k$;
$k = k + 1$;
fin

La méthode de descente du gradient est globalement convergente. Pour étudier la vitesse de convergence on va voir un le cas particulier d'une fonction quadratique.

Application :
On considère le problème d'optimisation quadratique $\min_{x\in\mathbb{R}^n} f(x)$, où f est une fonction fortement convexe quadratique $f(x) = \frac{1}{2}x^T Ax - b^T x$ et A une matrice symétrique d´définie positive de valeurs propres $0 < \lambda_1 \leq \cdots \leq \lambda_n$.
On a $\nabla f(x) = Ax - b$ et $\nabla^2 f(x) = A$. La fonction f admet un minimum global unique x^* qui est solution de $Ax = b$. Dans ce cas on peut déterminer le pas de descente optimal ρ_k et l'itération s'écrit :

$$x_{k+1} = x_k - \left[\frac{\|\nabla f(x_k)\|^T \|\nabla f(x_k)\|}{\nabla f(x_k)^T A \nabla f(x_k)}\right]\nabla f(x_k)$$

$$x_{k+1} = x_k - \left[\frac{\|\nabla f(x_k)\|^2}{\nabla f(x_k)^T A \nabla f(x_k)}\right]\nabla f(x_k)$$

$$\rho_k = \left[\frac{\|\nabla f(x_k)\|^2}{\nabla f(x_k)^T A \nabla f(x_k)}\right]$$

Exemple

f: $\mathbb{R}^3 \rightarrow \mathbb{R}$ définie par : $f(x_1, x_2) = 5x_1^2 + \frac{x_2^2}{2} - 3(x_1 + x_2)$

$$\nabla f(x_1, x_2) = \begin{bmatrix} 10x_1 - 3 \\ x_2 - 3 \end{bmatrix}, \quad A = H(f(x_1, x_2)) = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$x_0 = (-2, -7)$ et $x^* = (0.3,\ 3)$ après 5 itérations

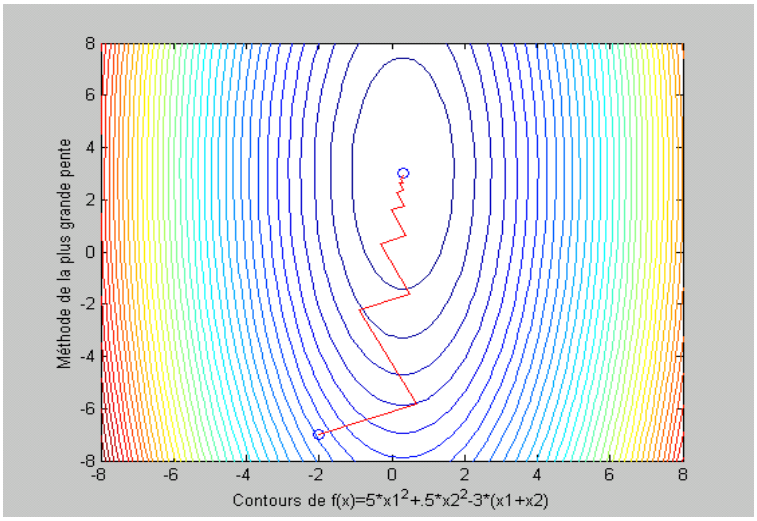

Figure 15 : Descente de la plus forte pente

Critère d'arrêt : $\left|\frac{f(x_k)-f(x^*)}{f(x^*)}\right| < 10^{-3}$

II.3 Méthode des directions conjuguées ou du gradient conjugué

Dans cette méthode, on utilise un modèle quadratique de *f*, on va donc d'abord présenter l'algorithme appliqué à une fonction fortement convexe quadratique.
Le but de la méthode des directions conjuguées est de minimiser une fonctionnelle quadratique
$f : \mathbb{R}^n \rightarrow \mathbb{R}$ de la forme

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c$$

Le point essentiel réside dans le fait que dans ce cas la famille des directions successives $d_0, \dots, d_k$ est orthogonale pour le produit scalaire associe à la matrice A, et en particulier est une famille libre dans un espace vectoriel de dimension finie. Cette méthode ingénieuse prend en compte la géométrie globale de la nappe représentative de la fonction. Cette propriété n'est plus vérifiée pour une fonction quelconque. Elle se généralise cependant à des fonctions non quadratiques par des méthodes telles que **Fletcher-Reeves** ou **Polak-Ribiere**.

La méthode du gradient conjugué est la méthode de descente définie par l'algorithme ci-dessous:

Méthode des directions conjuguées ou du gradient conjugué

Initilisation

Choisir x_0 et poser $k = 0$, un seuil de tolérance $\varepsilon > 0$

Etape1

$d_0 = -\nabla f(x_0),\ \nabla f(x_0) = Ax_0 - b$

Etape2

Tant que $\|d_k\| > \varepsilon$

$\alpha_k = -\dfrac{\nabla f(x_k)^T d_k}{{d_k}^T A d_k}$

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

Etape3

$\beta_{k+1} = \dfrac{\nabla f(x_{k+1})^T A d_k}{{d_k}^T A d_k}$

$d_{k+1} = -\nabla \mathrm{f}(x_{k+1}) + \beta_k d_k$

k=k+1

Exemple

f: $\mathbb{R}^2 \to \mathbb{R}$ définie par : $f(x_1, x_2) = 5x_1^2 + \frac{x_2^2}{2} - 3(x_1 + x_2)$ $\quad x^* = (0.3,\ 3)$

$$\nabla f(x_1, x_2) = \begin{bmatrix} 10x_1 - 3 \\ x_2 - 3 \end{bmatrix},\ A = H(f(x_1, x_2)) = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix},\ b = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\nabla f(x_1, x_2) = A\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - b = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 10x_1 - 3 \\ x_2 - 3 \end{bmatrix}$$

1- $x_0 = \begin{pmatrix} -2 \\ -7 \end{pmatrix}$ et $d_0 = -\nabla f(x_0) = \begin{pmatrix} 23 \\ 10 \end{pmatrix}$

2- $\alpha_0 = -\frac{\nabla f(x_0)^T d_0}{{d_0}^T A d_0} = \frac{\|(23,\ 10)\|^2}{(23,\ 10)\begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}\begin{pmatrix} 23 \\ 10 \end{pmatrix}} = \frac{629}{5390} = 0.1167$

3- $x_1 = x_0 + \alpha_0 d_0 = \begin{pmatrix} -2 \\ -7 \end{pmatrix} + 0.1167\begin{pmatrix} 23 \\ 10 \end{pmatrix} = \begin{pmatrix} 0.684 \\ -5.833 \end{pmatrix}$

4- $\nabla f(x_1) = \begin{pmatrix} 3.84 \\ -8.833 \end{pmatrix}$

5- $\beta_0 = \frac{\nabla f(x_1)^T A d_0}{{d_0}^T A d_0} = \frac{[3.84 \quad -8.833]\begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}\begin{Vmatrix} 23 \\ 10 \end{Vmatrix}}{(23,\ 10)\begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}\begin{pmatrix} 23 \\ 10 \end{pmatrix}} = \frac{[38.4 \quad -8.833]\begin{Vmatrix} 23 \\ 10 \end{Vmatrix}}{5390} = \frac{794.87}{5390} = 0.14747$

6- $d_1 = -\nabla f(x_1)^T + \beta_0 d_0 = (-3.84, 8.833) + 0.14747\begin{pmatrix} 23 \\ 10 \end{pmatrix}$

$= (-3.84, 8.833) + (3.39181, 1.4747) = \begin{pmatrix} -0.448 \\ 10.31 \end{pmatrix}$

7- $\alpha_1 = -\frac{\nabla f(x_1)^T d_1}{{d_1}^T A d_1} = \frac{\|(0.448,\ -10.31)\|^2}{(-0.448,\ 10.31)\begin{bmatrix}10 & 0\\ 0 & 1\end{bmatrix}\begin{pmatrix}-0.448\\ 10.31\end{pmatrix}} = \frac{106.4968}{126.3665} = 0.8569$

8- $x_2 = x_1 + \alpha_1 d_1 = \begin{pmatrix}0.684\\ -5.833\end{pmatrix} + 0.8569\begin{pmatrix}-0.448\\ 10.31\end{pmatrix} = \begin{pmatrix}0.3\\ -3\end{pmatrix} = x^*$

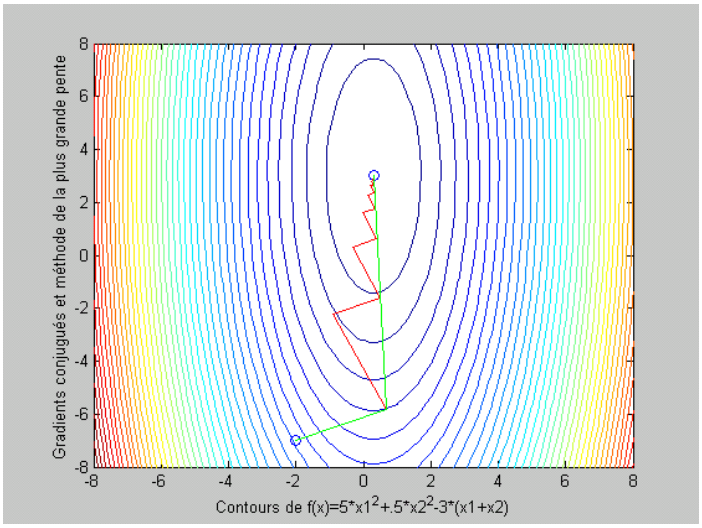

Figure 16 : Directions conjuguées ou du gradient conjugué

II.4 Méthode de Newton

a- Principe

Soit $f:\ \mathbb{R}^n \to \mathbb{R},\ f \in C^2$. Soit maintenant (*P*) le problème d'optimisation sans contraintes

$$min\{f(x), x \in \mathbb{R}^n\}$$

Le principe de la méthode de Newton consiste à minimiser successivement les approximations du second-ordre d'une fonction *f* ; plus précisément si

$$f(x) \cong f(x_k) + \nabla f(x_k)(x - x_k) + \frac{1}{2}(x - x_k)^T H(x_k)(x - x_k) = q(x),\ \ \forall x \in V(x_k)$$

$H(x_k)$ étant la matrice hessienne de *f* au point x_k alors une condition nécessaire de minimum pour *q* est que $\nabla q(x) = 0$, ou

$$\nabla f(x_k) + H(x_k)(x - x_k) = 0$$

Supposons que *H* est inversible, alors le successeur de x_k est donné par

$$x_{k+1} = x_k - H^{-1}(x_k)\nabla f(x_k)$$

Cette équation donne la forme générale des points générés par l'algorithme de Newton. Supposons que $\nabla f(x^*) = 0$ et que $H(x^*)$ est définie positive (x^* un minimum local), alors $H(x_k)$ reste définie positive en tout point voisin de x^*: Ceci assure que le successeur de x_k est bien définie.

b- Algorithme de descente de Newton

Algorithme de descente de Newton

Données : Un point initial $x_0 \in \mathbb{R}^n$ un seuil de tolérance $\varepsilon > 0$
Résultat : Un point $x \in \mathbb{R}^n$ proche de x^* ?
Initialiser *x* :
$x = x_0$;
$k = 0$;
Tant que $\|\nabla f(x_k)\| > \varepsilon$
Calculer le premier pas de descente :
$d_k = -H^{-1}(x_k)\nabla f(x_k)$;
$x_{k+1} = x_k + d_k$
$k = k + 1$;

Exemple

f: $\mathbb{R}^3 \rightarrow \mathbb{R}$ définie par : $f(x_1, x_2) = 2x_1^2 + 2x_2^2 + 2x_1x_2 - 4x_1 - 6x_2$ $\quad x_0 = \begin{pmatrix}1\\1\end{pmatrix}$

$$\nabla f(x_1, x_2) = \begin{bmatrix}4x_1 + 2x_2 - 4\\2x_1 + 4x_2 - 6\end{bmatrix}, \quad H(f(x_1, x_2)) = \begin{bmatrix}4 & 2\\2 & 4\end{bmatrix},$$

$$\nabla f(x_0) = \begin{pmatrix}2\\0\end{pmatrix}$$

$$H^{-1}\big(f(x_1, x_2)\big) = \frac{1}{12}\begin{bmatrix}4 & -2\\-2 & 4\end{bmatrix} = \begin{bmatrix}\frac{1}{3} & -\frac{1}{6}\\-\frac{1}{6} & \frac{1}{3}\end{bmatrix}$$

$$d_0 = -H^{-1}(x_0)\nabla f(x_0) = \begin{bmatrix}\frac{1}{3} & -\frac{1}{6}\\-\frac{1}{6} & \frac{1}{3}\end{bmatrix}\begin{pmatrix}2\\0\end{pmatrix} = \begin{pmatrix}-\frac{2}{3}\\\frac{1}{3}\end{pmatrix}$$

$$x_1 = x_0 + d_0 = \begin{pmatrix}1\\1\end{pmatrix} + \begin{pmatrix}-\frac{2}{3}\\\frac{1}{3}\end{pmatrix} = \begin{pmatrix}\frac{1}{3}\\\frac{4}{3}\end{pmatrix}$$

$\nabla f(x_1) = \begin{bmatrix}0\\0\end{bmatrix}$ Donc le point $x_1 = \begin{pmatrix}\frac{1}{3}\\\frac{4}{3}\end{pmatrix}$ est optimal

II.5 Méthode de Levenberg-Marquardt

a- Principe

Cette méthode qui combine la méthode du gradient et la méthode de Newton est souvent utilisée afin de résoudre le problème de l'inversion de de la hessienne $H = \nabla^2 f(x_k)$ et d'obtenir une matrice définie positive. Dans ce cas, la matrice $\nabla^2 f(x_k)$ est remplacée par $H_k = \nabla^2 f(x_k) + \mu I_d > 0$ avec $\mu > 0$ un paramètre de régularisation défini par l'utilisateur tel que $\nabla^2 f(x_k) > 0$.
Dans ce cas, μ_k optimal est obtenu en résolvant l'équation de la direction de descente

$$d_k = -[H_k]^{-1}\nabla f(x_k) = -[\nabla^2 f(x_k) + \mu I_d]\nabla f(x_k)$$

$$x_{k+1} = x_k - \rho_k[H_k]^{-1}\nabla f(x_k) = x_k + \rho_k d_k$$

b- Algorithme de descente de Newton

Méthode de Levenberg-Marquardt

Données : Un point initial $x_0 \in \mathbb{R}^n$ un paramètre de régularisation $\mu_0 > 0$, $\rho_k \in]0,1[$
Résultat : Un point $x \in \mathbb{R}^n$ proche de x^* ?
Initialiser *x* :
$x = x_0$;
$k = 0$;
Calculer : $\nabla f(x_0),\ H_0(x_0)$
Etape1
Calculer : $H_k = H(x_k) + \mu_k I_d$
Tant que $H_k < 0$ alors
$\mu_k = c_0 \mu_k$ et aller à l'etape1
Etape2
$x_{k+1} = x_k - \rho_k [H_k]^{-1} \nabla f(x_k)$
Calculer : $f(x_{k+1})$
Etape3
Si $f(x_{k+1}) \geq f(x_k)$ alors
$\mu_k = c_1 \mu_k,\ x_{k+1} = x_k$ et aller à l'etape1 // $c_0, c_1, c_2 > 0$
Si non $\mu_{k+1} = \frac{\mu_k}{c_2}$
$k = k + 1$; et aller à l'etape1

II.6 Méthodes quasi-Newton

a- Principe

Souvent, dans la pratique, la matrice hessienne H^{-1} est très difficile à évaluer dans le cas où la fonction *f* n'est pas analytique. Le gradient quant à lui est toujours plus ou moins accessible (méthodes inverses). Pour des problèmes de grandes dimensions, le calcul du Hessien est trop couteux. On peut alors utiliser des algorithmes, dits quasi-Newton, et on souhaite donc ne pas calculer exactement la hessienne, mais simplement on évalue une approximation de l'inverse de la matrice Hessienne. Parmi les méthodes d'approximation du hessien deux sont retenues ici : la méthode *BFGS* pour **Broyden** - **Fletcher** - **Goldfarb** - **Shanno** et la méthode *DFP* pour **Davidon** - **Fletcher** - **Powell.**
L'idée ici est d'imiter l'algorithme de Newton tout en évitant de calculer *H* et «son inverse ». Plus précisément, cela signifie que, à l'itération *k*, nous allons chercher à construire une **approximation** S_k ***symétrique définie positive*** qui remplace la matrice $H^{-1}(x_k)$ et ρ_k un paramètre positif par un algorithme de minimisation unidimensionnel le long de la direction $d_k = -S_k \nabla f(x_k)$ tel que l'itération classique $x_{k+1} = x_k - [H_k]^{-1} \nabla f(x_k)$ Soit remplacée par une itération plus simple et beaucoup moins couteuse $x_{k+1} = x_k - \rho_k S_k \nabla f(x_k)$

b- Méthode de Davidson-Fletcher-Powell (*DFP*)

Le but est de calculer «une bonne approximation » S_k de $H^{-1}(x_k)$, de telle manière que la différence soit petite pour une norme matricielle. Cette méthode permet notamment, pour une fonctionnelle quadratique, de construire l'inverse du Hessien. Elle engendre également des directions conjuguées. Cette méthode utilise la formule de correction (de rang 2) suivante :

$$S_{k+1} = S_k + \frac{\delta_k \delta_k^T}{\delta_k^T \gamma_k} + \frac{S_k \gamma_k \gamma_k^T S_k}{\gamma_k^T S_k \gamma_k}$$

Où le point x_{k+1} est obtenu à partir de x_k par déplacement dans la direction : $d_k = -S_k \nabla f(x_k)$

Et / où :

$$\delta_k = x_{k+1} - x_k \ , \quad \text{et} \quad \gamma_k = \nabla f(x_k) - \nabla f(x_k)$$

Le résultat suivant montre que, sous certaines conditions, la formule de correction (de rang 2) précédente conserve la caractéristique 'définie-positivité' des matrices

Algorithme de Davidson-Fletcher-Powell (*DFP*)

Algorithme de Davidson-Fletcher-Powell (DFP)

Choisir S_0, 'matrice symétrique définie-positivité'
Poser $k = 0$
Choisir $x_0 \in \mathbb{R}^n$ Un point initial
Choisir $\varepsilon > 0$
Tant que $\|x_{k+1} - x_k\| \geq \varepsilon$ et $(k \leq k_{max})$ faire
Calculer ρ_k par une recherche linéaire sur $d_k = -S_k \nabla f(x_k)$
Poser $x_{k+1} = x_k + \rho_k d_k$
Poser $\delta_k = \rho_k d_k$
Calculer $\gamma_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
Calculer $S_{k+1} = S_k + \frac{\delta_k \delta_k^T}{\delta_k^T \gamma_k} + \frac{S_k \gamma_k \gamma_k^T S_k}{\gamma_k^T S_k \gamma_k}$
Fin tant que

c- **Méthode de Broyden-Fletcher-Goldfarb-Shanno (*BFGS*)**

Cette dernière méthode est considéré comme très robuste et moins sensible aux erreurs dans les recherches linéaires. Elle a été développé indépendamment par Broyden (1970), Fletcher (1970), Goldfarb (1970) et Shanno (1969) utilise, pour construire une approximation de l'inverse du Hessien, une formule de *correction de rang 2* directement dérivée de la formule précédente de Davidson-Fletcher-Powell (*DFP*):

$$S_{k+1} = S_k + \left[1 + \frac{\gamma_k^T S_k \gamma_k}{\delta_k^T \gamma_k}\right] \frac{\delta_k \delta_k^T}{\delta_k^T \gamma_k} - \frac{\delta_k \gamma_k^T S_k + S_k \gamma_k \delta_k^T}{\delta_k^T \gamma_k}$$

Algorithme de Broyden-Fletcher-Goldfarb-Shanno (*BFGS*)

Algorithme de Broyden-Fletcher-Goldfarb-Shanno (BFGS)

Choisir S_0, 'matrice symétrique définie-positivité'
Poser $k = 0$
Choisir $x_0 \in \mathbb{R}^n$ Un point initial
Choisir $\varepsilon > 0$
Tant que $\|x_{k+1} - x_k\| \geq \varepsilon$ et $(k \leq k_{max})$ faire
Calculer ρ_k par une recherche linéaire sur $d_k = -S_k \nabla f(x_k)$
Poser $x_{k+1} = x_k + \rho_k d_k$
Poser $\delta_k = \rho_k d_k$
Calculer $\gamma_k = \nabla f(x_k) - \nabla f(x_k)$
Calculer $S_{k+1} = S_k + \left[1 + \frac{\gamma_k^T S_k \gamma_k}{\delta_k^T \gamma_k}\right] \frac{\delta_k \delta_k^T}{\delta_k^T \gamma_k} - \frac{\delta_k \gamma_k^T S_k + S_k \gamma_k \delta_k^T}{\delta_k^T \gamma_k}$
Fin tant que