X. Algorithmes d'optimisation

1. Introduction

Matlab a une série d'algorithmes déjà implémentés pour trouver les racines (`root, fzero`), les moindres carrés (`lsqcurvefit, lsqlin`...), la solution de systèmes d'équations (`fsolve,fzero`) et la minimisation, en une et plusieurs dimensions. Pour minimiser une fonction à une variable dans un domaine on utilise `fminbnd` et si la fonction a plusieurs variables, on utilise `fminsearch`. Pour le cas de problèmes contraints on utilise `linprog` et `quadrprog` pour les cas linéaires et quadratiques respectivement. La fonction `fmincon` permet trouver le minimum d'un problème avec contraintes non linéaire et multi-variable

Matlab possède un toolkit d'optimisation (Optimization Toolbox) pour les problèmes plus compliqués, qui automatise via GUI (interface graphique) le procesus de choix de l'algorithme. Matlab utilise plusieurs algorithmes selon le type de problème a résoudre 'interior reflective Newton method', 'trust-region-dogleg', 'trust-region-reflective', 'levenberg-marquardt', 'simplex', 'BFGS', 'MinMax'… Mais, qu'est-ce qu'un algorithme d'optimisation ? Comment ça marche? Quels sont les paramètres à contrôler ? Quelles sont les limitations ?

2. Definitions

Un **algorithme d'optimisation** est une procédure mathématique qui permet d'obtenir les minimums (ou maximums)[1] d'une fonction réele f (que l'on appelle fonction objective)

$$\min_{x \in \Re^n} f(x)$$

En général la solution est un sous-espace A$\in \Re^n$ qui est soumis à un ensemble de contraintes (conditions sur les variables), qui sont exprimées comme un système d'équations et inéquations. Les éléments de A sont appelés solutions admissibles et souvent ont des bornes supérieures et inferieures $x_l \le x \le x_u \in$ A

Les problèmes d'optimisation peuvent être classés selon le type de restriction :

a) Minimisation **sans restrictions**

b) Minimisation avec **restrictions d'égalité** $g_i(x) = 0$ i = 1, …, m_e

[1] Maximiser une fonction f(x) = Minimiser – f(x)

c) Minimisation avec **restrictions d'égalité et d'inégalités** $g_i(x) \leq 0$ i = m_e +1… m

Les algorithmes d'optimisation sont des processus itératifs que génèrent une séquence de valeurs x_{n+1} à partir d'un point de départ x_0. Un algorithme est convergent quand pour n'importe quel point de départ, la séquence arrive à la solution (maximum ou minimum).

Les algorithmes d'optimisation ont besoin en général des dérivées de premier et deuxième dégré de la fonction Pour le calcul du gradient d'une fonction, on peut utiliser la dérivation directe, approximation par différences finies…Par exemple, la méthode de descente de gradient a besoin juste des 1^{eres} dérivées; la méthode de Newton nécessite les 2èmes dérivées de la fonction objective ; sans dérivée, on peut trouver les méthodes d'algorithme du simplexe, 'simulated annealing', 'neural networks', algorithmes génétiques…

2.1. Typologie de problèmes

Les algorithmes d'optimisation s'utilisent en de nombreux problèmes, pour trouver les zéros de fonctions, pour minimiser la distance entre des points de mesure et une courbe (moindres carrés), intersections de fonctions et pour résoudre des systèmes d'équations à une ou plusieurs variables. En général, il n'y a pas de méthode idéale et ça dépend de la forme de la fonction à étudier et du type de problème à analyser.

La plupart des problèmes en physique et en ingénierie peuvent être représentés sous la forme de systèmes linéaires d'équations :

$$A(x)u = b(x)$$

Ou u est le vecteur de variables d'état (déplacements en problèmes mécaniques, température en problèmes thermiques, concentration en problèmes de contaminants, hauteur d'eau en problèmes de fluides…) ; A est la matrice de rigidité ('stiffness matrix') qui représente les propriétés propres du matériel et peut aussi être une matrice de conductivité, perméabilité,… b est un vecteur représentant les actions ou forces externes sur un système.

En général A et b (et pourtant aussi u) sont dépendants d'une série de variables de design ou paramètres du système que l'on veut définir. On peut trouver différents types de problèmes :

a) **Control optimal** : Déterminer les actions b nécessaires sur un système pour que u s'approche dans un état défini comme optimal. Par exemple, quelle pression ou quelle température dois-je appliquer pour qu'un système soit en équilibre.

b) **Design optimal** : Le but est de trouver les variables de design x (par exemple design d'une structure, d'un produit) que suivent une série de critères d'optimalité (couts, volume ou poids minimal) et qui satisfait une série de conditions (par exemple valeurs maximales préétablies)

c) **Estimation de paramètres** ou problème inverse (problèmes du type moindres carrés) : Trouver les paramètres du modèle afin de correspondre une fonction aux observations disponibles (valeurs calculées qui s'approchent des valeurs mesurées dans un cas réel).

Tous ces types de problèmes impliquent la minimisation d'une fonction dépendante de x par u et ont des restrictions (conditions) sur x et u. La résolution du système d'équations peut se compliquer dans le cas de problèmes non linéaires ou temporels.

2.1. Paramètres d'un algorithme d'optimisation

2.1.1. Approximation Initiale

Pour initialiser l'algorithme, il est nécessaire d'avoir une approximation initiale a la solution x_0. (Point de départ). Le choix d'une bonne approximation initiale conditionne la convergence ou pas à la solution.

2.1.2. Nombre d'Itérations

Un algorithme d'optimisation utilise un processus récursif, calcule une nouvelle approximation (itération) à la solution réelle jusqu'à ce que les critères de convergence soient atteints. En programmation, c'est une boucle de répétition où la nouvelle approximation est construite à partir les approximations antérieures.

2.1.3. Vitesse de convergence

Quand on parle de convergence proche d'une solution, on parle de la vitesse à laquelle les termes de l'itération approchent sa limite.

$$\lim_{n\to\infty} \frac{|x_{n+1} - \xi|}{|x_n - \xi|^q} = \mu$$, ou μ >0 et q est l'ordre de convergence

En général, les ordres de convergences son linéaires (p=1), quadratiques (p=2), cubiques (p=3), quartiques (p=4)… Une méthode d'optimisation avec un ordre de convergence supérieur arrive à la solution avec peu d'itérations. Le choix d'une méthode avec une haute convergence est important pour les problèmes d'une certaine taille ou avec de multiples paramètres. Par exemple, pour une convergence quadratique, on peut dire que le nombre de chiffres corrects est double (au minimum) à chaque pas de calcul. Ou dit sous une autre forme, l'erreur diminue quadratiquement à chaque itération.

Si un algorithme ne converge pas, ça ne veut pas dire qu'il n'existe pas de solution. Il n'existe aucun algorithme universel dont la convergence soit garantie, en général il dépend du choix de l'initialisation x_0 et de les propriétés de la fonction (continuité, dérivabilité)

2.1.4. Critère d'arrêt

Critères pour arrêter le processus de calcul. Il existe plusieurs critères d'arrêt. Les plus utilisées :

a) Nombre maximal d'itérations N_{max}

b) $\|f(x_n)\| < \varepsilon_1$ Valeur de la fonction

c) $\|x_{n+1} - x_n\| < \varepsilon_2$ Différence entre deux approximations successives

Où $\varepsilon_1, \varepsilon_2\ \varepsilon_2 \in \Re$ sont les tolérances et sont choisies en fonction du type de problème. En général, ce sont des valeurs négligeables ($\varepsilon_i \approx 10^{-4}$-$10^{-6}$).

3. Rappel

3.1. Points critiques : maximums, minimums

Dans un ensemble ordonné, le plus grand élément (ou le plus petit) d'une partie de cet ensemble est un extremum maximum (ou minimum) s'il est supérieur (ou inferieur) à tous les autres éléments de la partie. Ce groupe d'éléments sont connus sous le nom de points critiques ou points extremum définis sur un domaine d'étude D (espace topologique).

f(a) est un **maximum global** si $\forall x \in D \quad f(x) \leq f(a)$

f(a) est un **minimum global** si $\forall x \in D \quad f(x) \geq f(a)$

f(a) est un **maximum local (ou relatif)** s'il existe un voisinage V de a tel que $\forall x \in V, f(x) \leq f(a)$

f(a) est un **minimum local (ou relatif)** s'il existe un voisinage V de a tel que $\forall x \in V, f(x) \geq f(a)$

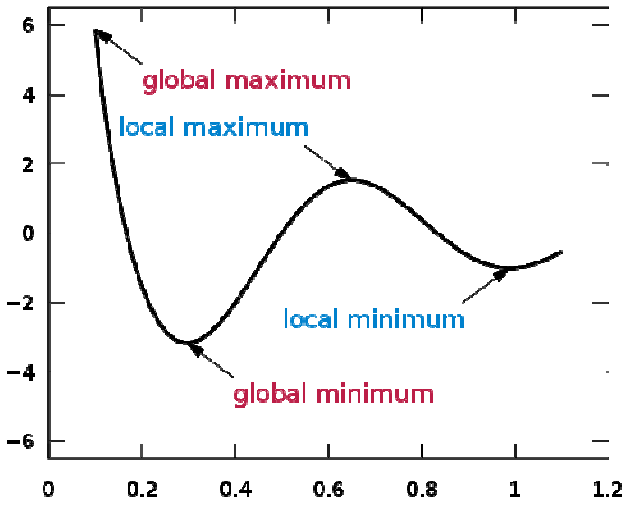


Figure 1- Points extrêmes (maximums et minimums locaux et globaux) sur une fonction

Pour trouver les maximums et les minimums d'une fonction, on utilise le calcul différentiel, et là ou la dérivée de la fonction s'annule, on trouve soit un maximum ou un minimum. Un minimum local est facile à trouver, mais il est difficile de trouver un minimum absolu.

En général, une fonction a plusieurs minimums. Pour arriver à la solution désirée (minimum global), il est très important d'analyser la fonction en détail avant de choisir un point de départ x_0 pour l'algorithme.

Pour trouver le type de point critique, on utilise les deuxièmes dérivées évaluées dans le point d'étude. Pour le cas d'une fonction à une variable f(x), si f''(a)>0, on trouve un minimum local en ce point, si f''(a)<0 on trouve un maximum local et si f''(a)>0, quand f''(a)=0 on n'a pas d'information, mais ça peut être un point de selle.

Par exemple, pour les fonctions continues et dérivables deux fois, les points stationnaires identifiés (là ou la dérivée est 0) sont classés selon la matrice Hessiene (minimum local si positif, maximum local si négatif et indéfini si s'agit d'un point de selle). Pour le cas d'une fonction à deux variables, on trouve $f_{xx}(x,y)$, $f_{yy}(x,y)$ et $f_{xy}(x,y)$ evalué au point (a,b). Le déterminant de la matrice Hessiene[2] $|H| = f_{xx}(x,y) * f_{yy}(x,y) - f^2{}_{xy}(x,y)$

$f_{xx}(a,b)*f_{yy}(a,b)-f^2_{xy}(a,b)$	$f_{xx}(a,b)$	Classification
>0	>0	Minimum Local
>0	<0	Maximum Local
<0	-	Point de Selle

Pour trouver les points extrêmes (ou points critiques) d'une fonction de deux variables, par exemple : f (x, y) = $x^3+y^3+3x^2-3y^2-8$, on doit trouver les points qui annulent les dérivés partielles de la fonction $\partial_x f(x,y)=0$ et $\partial_y f(x,y)=0$.

```
syms x y ;
f=x^3+y^3+3*x^2-3*y^2-8;
fx=diff(f,x)
fy=diff(f,y)
S=solve(fx,fy)
```

[2] La matrice Hessienne H(f) d'une fonction f est une matrice carrée de ses dérivées partielles secondes.

$$H_{ij}(f)=\frac{\partial^2 f}{\partial x_i \partial x_j}=\begin{vmatrix} \partial^2 f/\partial x_1^2 & \partial^2 f/\partial x_1 \partial x_2 & \cdots & \partial^2 f/\partial x_1 \partial x_n \\ \partial^2 f/\partial x_2 \partial x_1 & \partial^2 f/\partial x_2^2 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \partial^2 f/\partial x_n \partial x_1 & \partial^2 f/\partial x_n \partial x_2 & \cdots & \partial^2 f/\partial x_n^2 \end{vmatrix}$$

La commande solve trouve les solutions qui sont égales à zéro simultanément pour les deux fonctions dérivées. S est une structure variable. Pour voir les valeurs de S :

```
[S.x,S.y]
```

Le résultat montre les points critiques pour la fonction analysée {(0,0),(0,2),(-2,0).(-2,2)}.

Pour visualiser les résultats on peut utiliser la fonction :

```
[x,y]= meshgrid (-3:0.1:3);
z= x.^3+y.^3+3*x.^2-3*y.^2-8;
mesh(x,y,z)
xlabel('x')
ylabel('y')
zlabel ('z=f(x,y)')
```

Ou aussi la fonction

```
surfc(x,y,z)
```

Parfois c'est aussi utile de visualiser les lignes de niveaux dans un graphique séparé

```
contour(x,y,z)
```

Ou dans le même graphique, on utilise pour dessiner les contours en dessous de la maille ou pour dessiner les courbes de niveau en dessus de la surface.

```
meshc(x,y,z)
surfc(x,y,z)
```

On peut changer le paramètre par défaut des courbes de niveau :

```
contour(x,y,z,20)
```

On observe mieux les deux points critiques (-2,0 et 2,0)

Matlab permet de dessiner les courbes a différentes hauteurs avec :

```
[c,h] = contour(x,y,z,-14 :-4) ;
clabel (c,h)
```

A partir de ces contours, on observe :

- Peu importe la direction d'approche du point (-2,0) les courbes de niveau augmentent. Par conséquent, on trouve un maximum local à (-2,0).
- Peu importe la direction d'approche du point (0,2) les courbes de niveau diminuent. Par conséquent, on trouve un minimum local au (0,2).

- Pour les points (0,0) et (-2,2), on observe une croissance et décroissance des courbes de niveau selon des directions opposées. On peut dire qu'y a un point de selle à (0,0) et à (-2,2).

3.2. Optimisation avec Matlab

3.2.1. Minimisation unidimensionnel : `fminbnd`

Les méthodes d'optimisation pour les fonctions à une variable s'appellent recherche par ligne ('line search'). Les algorithmes implémentés dans Matlab pour la fonction `fminbnd` sont le 'Golden section search' et l'interpolation parabolique.

On crée une fonction externe dans un fichier .m. Travailler avec des fichiers externes permet de simplifier et réduire les erreurs.

```
function y = f(x)
y = 1./((x-0.3).^2 + 0.01)+ 1./((x - 0.9).^2 + 0.04) -6;
```

Dans linvite de commande Matlab on peut observer la fonction

```
clear all
fplot('f', [-5 5])
grid on
```

On fait un zoom pour observer où se trouve notre minimum

```
fplot('f', [-5 5 -10 25])
grid on
```

Notre minimum se trouve entre $0.3 \le x_{\min} \le 1$

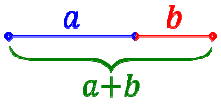
```
min = fminbnd('f',0.3,1,optimset('Display','iter'));
fplot('f',[0 2])
hold on ;
plot(min,f(min),'r*') ;
```

La fonction `fminbnd` permet de trouver le minimum de la fonction dans un intervalle donné. Dans les options, on peut voir les approximation successives et l'algorithme que Matlab utilise avec `optimset('Display','iter')`

`fminbnd` trouve minimums locaux. C'est important de choisir une bonne approximation initiale. `fminbbnd` a une convergence lente quand la solution est proche de l'intervalle. Essayez une restriction de la solution entre [0,0.6370]. Combien d'itérations sont nécessaires?

La **Golden Section Search** est juste utilisable quand la fonction est continue et unimodale (f(x) a juste un minimum dans un intervalle [a,b]).

On veut réduire l'intervalle qui contient la valeur minimum de la fonction. Le facteur optimal de réduction pour l'intervalle c de recherche est le nombre d'or φ



$$\frac{a+b}{b} = \frac{a}{b} = \varphi = \frac{1+\sqrt{5}}{2} = 0.6180$$

Figure 2 – Nombre d'Or

Pour x_1 et x_2 dans [a ; b], on peut distinguer deux cas :

a) Si $f(x_1) < f(x_2)$, alors [a,b] :=[a,x_2] $x_1 = x_2$ et on calcule le nouvel x_1 selon $x_1 = ca + (1-c)b$

b) Si $f(x_1) > f(x_2)$, alors [a,b] :=[x_1,b] $x_2 = x_1$ et on calcule le nouvel x_1 selon $x_2 = (1-c)a + cb$

pour minimiser $\|b-a\| < \varepsilon_2$ (ou ε_2 est dans Matlab la TolX = e-4). La vitesse de convergence est en ce cas linéaire.

L'interpolation Parabolique : Pour obtenir une convergence superieure, on peut utiliser l'Interpolation parabolique dans l'intervalle [a,b]. On peut approximer une parabole qui passe par les trois points (a,b et c) et calculer l'approximation suivante comme le minimum de cette parabole (calculable analytiquement).

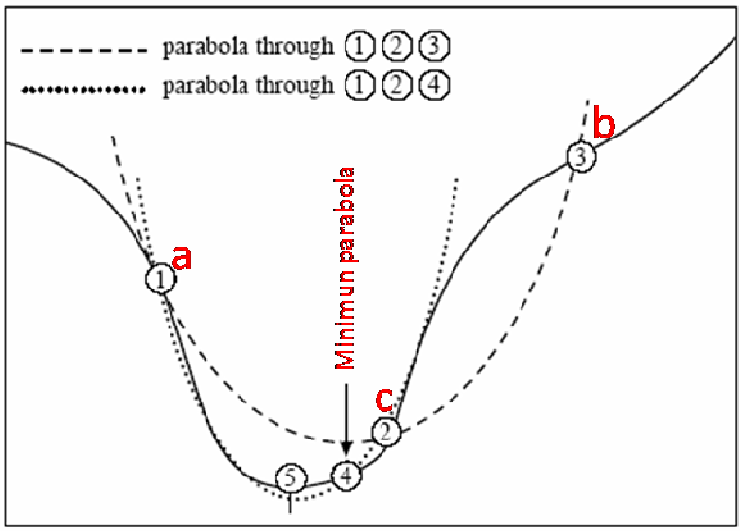


Figure 3 – Représentation d'algorithmes d'Interpolation parabolique pour l'optimisation en fonctions d'une variable.

Cet algorithme ne marche pas si on a une fonction linéaire. Pour certaines fonctions, l'erreur commise avec cette méthode est fixe parce que proche de la solution, la fonction devient localement linéaire, il suffit de considérer que la solution est le dernier point.

3.2.2. Minimisation multidimensionnelle : `fminsearch, fminunc`

Dans Matlab, pour minimiser une fonction à plusieurs variables, on utilise l'algorithme du Simplex qui est implémenté dans `fminsearch`.

Matlab utilise la **méthode Simplex** parce qu'on n'a besoin ni de gradient ni de calculer la matrice Hessienne à chaque itération. La méthode consiste à entourer le minimum dans un simplex. Un simplex est un ensemble de N+1 points qui entoure le minimum (en 1D est une ligne, en 2D c'est un triangle, et en 3D c'est une pyramide). Chaque simplexe est caractérisé pour n+1 vecteurs aux vertex du simplex. A chaque pas de calcul, un nouveau point est pris à l'intérieur ou à cote du simplex. La valeur de la fonction en ce point est comparée avec les valeurs des fonctions évaluées aux vertex et normalement un des vertex est remplacé par le nouveau point générant un nouveau simplex (par réflection, expansion ou contraction). Cette procédure est répété jusqu'a le diamètre du simplex soit inférieur a une tolérance spécifiée

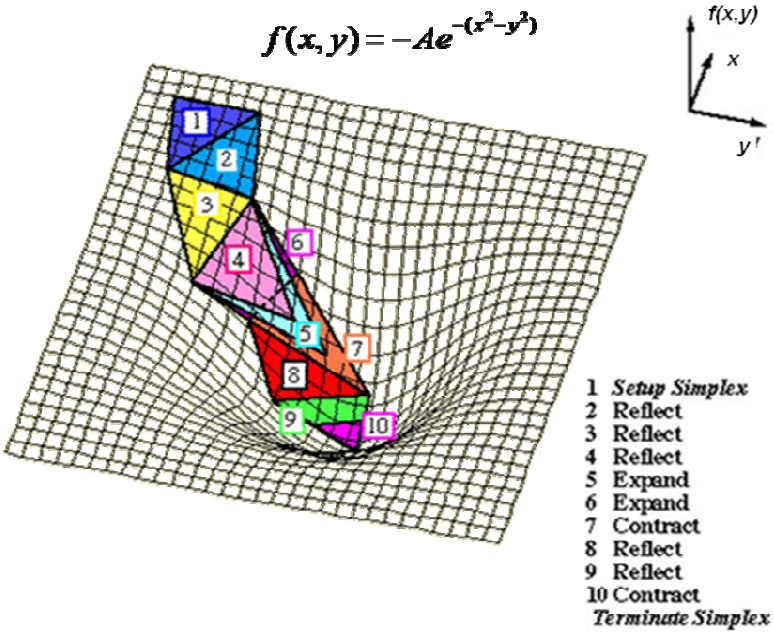


Figure 4- Représentation de l'Algorithme du Simplexe pour l'optimisation en fonctions de deux variables.

Exemple : On définit dans le fichier f.m la fonction « banana » que doit être sur le « Current Directory » $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$:

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

Dans l'invite de commandes de Matlab, on peut minimiser la fonction

```
>> x0 = [5 5];
>> [x, fval, exitflag, output] = fminsearch(f,x0)
```

Une autre fonction avec une syntaxe similaire est le `fminunc` qui solutionne des problèmes d'optimisation non linéaires et multi-variables et sans restrictions. Cette fonction permet de changer entre algorithmes différents par exemple de méthode de Interior Reflective Newton (si on connait les dérivées) ou le Méthode BFGS en cas contraire. L'algorithme BFGS approxime la matrice Hessienne (methode quasi-Newton).

Exemple dans le fichier f.m on defines la function suivante $f(x) = x_1^2 x_2 + x_1 x_2^2 + x_1^2 + x_2^2$:

```
function y = f(x)
y= x(1)^2*x(2)+x(1)*x(2)^2+x(1)^2+x(2)^2;
```

a) Dans l'invite de commandes de Matlab, pour le cas BFGS, qui approxime la matrice Hessiane à chaque pas de calcul :

```
>> x0 = [1 -1];
>> [x,fval,exitflag,output,gradient,hessian] =
fminsearch(@myfun,x0)
```

b) Si on connait le gradient on peut le définir explicitement

```
function [y,dy] = f(x)
y = x(1)^2*x(2)+x(1)*x(2)^2+x(1)^2+x(2)^2;
dy(1) = 2*x(1)*x(2)+x(2)^2+2*x(1);
dy(2) = x(1)^2+2*x(1)*x(2)+2*x(2);
```

Dans le command de Matlab:

```
>> options = optimset('GradObj','on');
>> x0 = [1 1];
>> [x,fval,exitflag,output] = fminunc(@f,x0,options)
```

Pour de grands problèmes, Matlab recommande d'utiliser la méthode 'interior reflective Newton' avec des gradients conjugués preconditionnés parce que c'est un algorithme qui converge plus rapidement que l'antérieur. Pour des problèmes moyens, on utilise souvent le BFGS quasi-Newton. Matlab permet aussi d'utiliser la méthode de la descente maximale utilisant comme calcul de la matrice Hessienne : `HessUpdate 'steepdesc'`.

Les méthodes implémentées en Matlab sont complexes afin de pouvoir solutionner une grande variété de problèmes avec la performance maximale (moins itérations). Les algorithmes optimisent aussi l'espace de mémoire sur l'ordinateur (les problèmes réels impliquent en général matrices vides que l'on doit stocker en forme vectorielle). Les chapitres suivants expliquent de manière simple les méthodes de Newton et de Descente Maxime pour les problèmes d'optimisation.

4. Methode de Newton

Le méthode de Newton (ou méthode de Newton-Raphson) est la méthode la plus conue pour trouver les racines (solutions) d'une fonction de variable réelle. On choisit une valeur inital x0 proche de le solution. La fonction calcule la tangente au point (derivée de la function). La prochaine valeur à utiliser est l'intersection de la tangente et l'axe x. C'est un processus iteratif (iiterations succesives) qui va s'arrêter quand les critères de convergence soint atteints.

Le méthode de Newton peut être utilisée dans le cas d'une fonction continue et différentiable dans l'interval de recherche [a,b].

$f : [a,b] \rightarrow \Re$

La condition de différentiabilité assure la présence d'une droite tangente a notre fonction. La condition de continuité assure qu'il n'y a pas de sauts dans la fonction.

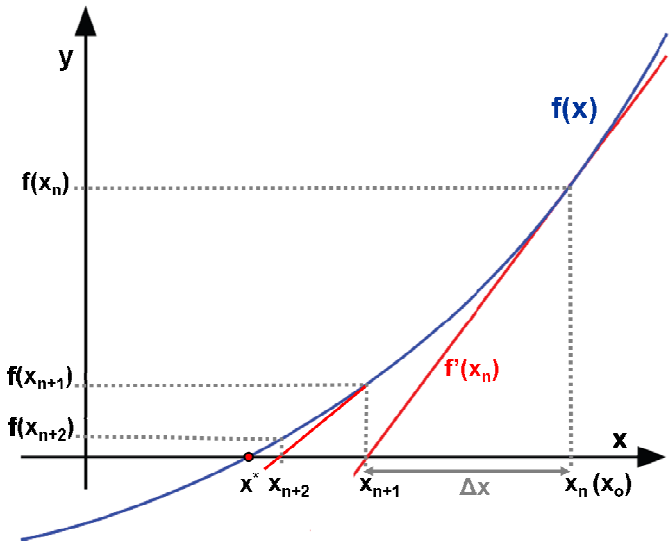


Figure 5 – Methode de Newton pour trouver zeros de fonctions

La tangente de la courbe (derivée de la fonction) s'obtient :

$$f'(x_n) = \frac{\Delta y}{\Delta x} = \frac{f(x_n)}{x_n - x_{n+1}}$$

et si on réorganize les termes, on peut écrire la méthode de Newton comme:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad \text{ou } n \geq 0$$

L'intervalle de recherché [a,b], peut être choisi en utilisant le **théorème des valeurs intermédiaires** qui énonce que pour une fonction f continue dans un intervalle [a,b] si f(a)>0 et f(b)<0 (ou l'inverse), il existe au moins une valeur $\varepsilon \in [a,b]$ qui confirme f(ε)=0

1.- Choisir une approximation initiale x_0.

2.- Calculer la valeur de la fonction en ce point f (x_0)

3.- Calculer la dérivée de la fonction f' et évaluer la valeur au point f'(x_0)

4.- L'approximation x_1 s'obtient avec $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$

5.- Répéter la procédure avec x_1, x_2, x_3… jusqu'à trouver la racine (solution) x*

En général, cette procédure a une convergence très rapide (convergence d'ordre quadratique), surtout quand la valeur initiale est proche de la solution. Malgrès ça, elle présente quelques difficultés :

a) Pour les approximations initiales loin de la solution, la méthode peut présenter une non convergence.

b) La dérivée de la fonction doit être calculée directement. Parfois, cette dérivée n'est pas exacte et on utilise alternativement des méthodes approximatives du type secant (on prend la secant[3] au lieu de la tangent pour chaque itération) avec un ordre de convergence linéaire.

c) Si la dérivée de la fonction n'est pas continue ou nulle, la méthode n'est pas utilisable. Pour les dérivées proches de zéro (tangent horizontal), la méthode peut bypasser la solution.

Pour la méthode de Newton, dans le cas de systèmes d'équations à plusieurs variables, on utilise la matrice jacobienne F'(x_n). Dans ce cas, au lieu de diviser par la pente évaluée au point, on multiplie par l'inverse de la matrice jacobienne : F'(x_n) (x_{n+1}-x_n) = -F(x_n)

Le méthode de Newton peut être aussi utilisée en optimisation de façon directe pour trouver les minimums et maximums locaux. Quand on atteint un minimum ou un maximum, la solution x* est un point stationnaire de la fonction f(x) (x* est une solution de la dérivée f'(x))

L'expansion de Taylor est une approximation d'une fonction construite avec des termes infinis, avec les valeurs de la fonction et les dérivées en un point a, avec un voisinage où la fonction est définie. C'est une approximation d'une fonction, en un point, très utilisée en analyse numérique parce qu'elle simplifie la description de fonctions complexes.

[3] La dérivée est approximée en prenant la pente entre deux points de la fonction. Le méthode de la sécant a un ordre de convergence inférieur a 2.

Si on écrit l'expansion de Taylor pour la fonction f(x) :

$$f(x+\Delta x) = f(x) + f(x)\cdot \Delta x + \frac{1}{2} f''(x) \cdot \Delta x^2$$

On trouve le valeur extrême pour :

$$\frac{\partial f(x+\Delta x)}{\partial \Delta x} = 0 \rightarrow f'(x)\cdot \Delta x + f''(x)\cdot \Delta x = 0$$

Si on a une fonction dérivable deux fois et l'approximation initiale est suffisamment proche de la solution, on peut définir l'algorithme (analogue au cas antérieur mais en substituant la valeur de la fonction par sa dérivée, et sa dérivée par sa 2eme dérivée):

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} \text{ , ou } n \geq 0$$

Dans le cas de solution de systèmes d'équations linéaires, (généralisation à plusieurs dimensions), on remplace la dérivée par le gradient f(x), et la 2eme dérivée avec l'inverse de la matrice Hessienne[4]

5. Méthode de la Descente de Gradient ou Descente Maxime

Cette méthode consiste à progresser en direction opposé du gradient (ou de son approximation) de la fonction au point évalué. La méthode de la descente de gradient est également connue sous les noms de descente de plus forte pente ou descente maximale. On peut progresser à pas constants ou en évaluant le meilleur pas et la direction d'avance. A chaque itération, le point d'arrêt devient le point de départ où la fonction est réévaluée et une nouvelle direction est suivie. Le processus est répété jusqu'à ce que le minimum soit atteint.

Schématiquement on peut résumer le processus ainsi :

1. Définir une approximation initiale (point de départ) x_0.

2. Déterminer la direction de descente maximale en x_0 en calculant le gradient de la fct f

3. Calculer la nouvelle valeur x_1 de sorte que $g(x_1) < g(x_0)$

$x_{n+1} = x_n - \alpha \cdot f'(x_n)$, où $n \geq 0$ et α est le pas d'avance, qui peut être fixé ou optimisé.

4. Répéter les étapes 2-4 avec x_0 remplacé par x_1

[4] En général les dérivées secondes sont complexes ou impossibles à calculer pour certaines fonctions et on utilise des approximations aux dérivées ou autres algorithmes.

La méthode de la descente maximale est un algorithme moins rapide que l'antérieur, qui convergera généralement linéairement à la solution. L'avantage est que cette méthode convergera même avec une conjecture initiale éloigné de la solution, mais il peut présenter des problèmes pour approcher des fonctions qui ont des formes avec pentes peu marquées.

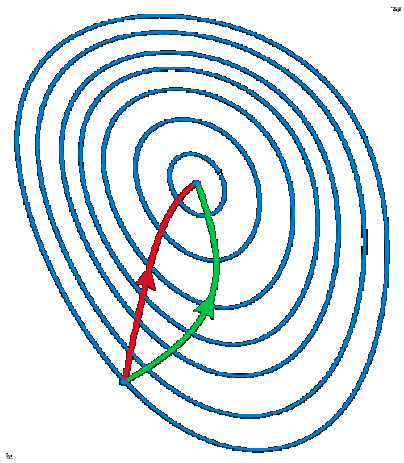


Figure 6 Comparaison entre la méthode de Newton et la méthode de la descente maxime. Extrait de [2]

6. Conclusions

Pour conclure,

- C'est très important de visualiser la fonction que l'on veut optimiser avant de choisir l'un ou l'autre des algorithmes et pour déterminer la valeur initiale à introduire pour notre analyse.

- Le même algorithme peut s'utiliser pour résoudre différents types de problèmes, optimisation, calcul de racines, solutions d'un système d'équations…

- La non convergence d'un algorithme n'implique pas la non existence de solution, il faut essayer avec un autre point de départ x_0 ou utilisant un autre type d'algorithme.

7. Références

[1] MATLAB Help

[2] Wikipedia

[3] Amos, Gilat, 2007. *Matlab, an introduction with application,* John Willey and Sohn, Inc.

[4] Faires and Burden, *Numerical Analysis, 2004 8th edition.*