1 méthode de Newton

La méthode de Newton n'est pas une méthode d'optimisation à proprement parler.
C'est en réalité une méthode utilisée pour résoudre des équations non linéaires de la forme $F(x) = 0$ où F est une fonction de $\mathbb{R}^n$ dans $\mathbb{R}^n$.
Nous allons d'abord la décrire puis montrer comment on peut l'appliquer à la recherche de minimum.

Algorithme de Newton dans $\mathbb{R}$

1. Initialisation

$k = 0$: choix de $x_0 \in \mathbb{R}$ dans un voisinage de x^*

2. Itération k

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

3. Critère d'arrêt

Si $|x_{k+1} - x_k| < \varepsilon$, STOP

Sinon, on pose $k = k + 1$ et on retourne à 2.

Remarquons qu'il faut non seulement assurer la convergence de la suite x_k vers la solution x^*, mais aussi montrer que cette suite est bien définie, c'est-à-dire montrer que $f'(x_k) \neq 0$ à l'étape 2.
Nous pouvons maintenant généraliser à $\mathbb{R}^n$. Soit F une fonction de classe $\mathcal{C}^1$ de $\mathbb{R}^n$ à valeurs dans $\mathbb{R}^n$.
On suppose que l'équation

$$F(x) = 0$$

possède au moins une solution notée x^* et que la matrice jacobienne $DF(x^*)$ est une matrice inversible. La continuité de DF permet alors d'assurer l'inversibilité de $DF(x_k)$ pour tout x_k dans un voisinage de x^*. L'algorithme est décrit ci-dessous.

Algorithme de Newton dans $\mathbb{R}^n$

1. Initialisation

$k = 0$: choix de x_0 dans un voisinage de x^*

2. Itération k

$x_{k+1} = x_k - [DF(x_k)]^{-1} F(x_k)$

3. Critère d'arrêt

Si $||x_{k+1} - x_k|| < \varepsilon$, STOP

Sinon, on pose $k = k + 1$ et on retourne à 2.

L'étape 2. de la méthode revient à résoudre le système linéaire suivant:

$$[DF(x_k)]\delta_k = F(x_k)$$

puis à poser $x_{k+1} = x_k - \delta_k$

1.1 Exemple 1

On veut évaluer l'unique racine positive r de l'équation $f(x) = 0$
où

$$f(x) = e^x - x - 2$$

Prenons $x_0 = 1$
on a :

$$x_{k+1} = x_k - \frac{e^{x_k} - x_k - 2}{e^{x_k} - 1}$$

Pour $k = 0, 1, ..$

$$x_1 = x_0 - \frac{e^{x_0} - x_0 - 2}{e^{x_0} - 1} = \frac{2}{e - 1} = 1.1639...$$

$$x_2 = x_1 - \frac{e^{x_1} - x_1 - 2}{e^{x_1} - 1} = 1.1464...$$

Code : Scilab

(a) Implémentation de la méthode de Newton

function [x,iter]=Newton(x0,f,J,tol)
tol=$1.e-2$;
grad=1;
$x = x0$;
iter=1;
while grad > tol iter < 100
step = $J(x)(x)$;
grad=norm(step)
x = x-step;
iter=iter+1;
end
if iter>99 then
printf("pas de convergence dans Newton")
end
endfunction

(b) Recherche numérique des solutions :

function y=f(x)
y=exp(x)-x-2

endfunction

function y=J(x)
y=exp(x)-1
endfunction

[$y1, iter1$]=Newton(1,f,J,1.e-2)

(c) Comparer le résultat avec la fonction fsolve :

y=fsolve(1,f)

1.2 Exemple 2

On cherche à approcher numériquement une solution du sytème d'équations suivant :

$$\begin{cases} x^2 + y^2 &= 2 \\ x^2 - y^2 &= 1 \end{cases}$$

la matrice jacobienne s'écrit :

$$\begin{pmatrix} 2x_1 & 2x_2 \\ 2x_1 & -2x_2 \end{pmatrix}$$

Code : Scilab

function Y=f(X)
$Y = [X(1)^2 + X(2)^2 - 2; X(1)^2 - X(2)^2 - 1];$
endfunction

function Df=Jac(X)
$Df(1,1) = 2 * X(1);$
$Df(1,2) = 2 * X(2);$
$Df(2,1) = 2 * X(1);$
$Df(2,2) = -2 * X(2);$
endfunction

[$y1, iter1$]=Newton($[1;1], f, Jac, 1.e-2$)

y=fsolve($[1;1], f$)

2 La descente de gradient

La Descente de Gradient (DG) s'applique lorsque l'on cherche le minimum d'une fonction dont on connaît l'expression analytique, qui est dérivable, mais dont le calcul direct du minimum est difficile. C'est un algorithme fondamental à connaître car utilisé partout sous des formes dérivées.

Partant d'un point x_0 arbitrairement choisi, un algorithme de descente va chercher à générer une suite d'itérés $(x_k)_{k\in\mathbb{N}}$ définie par :

$$\begin{cases} x_0 \text{ étant donné dans } \mathbb{R}^n \\ \text{calculer } x^{(k+1)} = x^{(k)} + \rho^{(k)} d^{(k)} \end{cases}$$

Le vecteur $d^{(k)}$ s'appelle direction de descente, $\rho^{(k)}$ le pas de la méthode à la $k-$ème itération.
Considérons une fonction définie par : $J : \mathbb{R}^n \to \mathbb{R}$
En pratique, on choisi $\rho^{(k)}$ et $d^{(k)}$ de façon à satisfaire l'inégalité suivante : $J(x^{(k+1)} \leq J(x^{(k+1)})$
De tels algorithmes sont souvent appelés algorithmes de descente.
Essentiellement, la différence entre ces algorithmes réside dans le choix de la direction de descente $d^{(k)}$.

Pour cet algorithme on choisit $d^{(k)} = -\nabla f(x^{(k)})$ comme direction de descente, l'algorithme complet s'écrit

ALGORITHME (DESCENTE DE GRADIENT)

choisir $x^{(0)} \in \mathbf{dom}\, f$ *et poser* $k = 0$

tant que $\left\|\nabla f(x^{(k)})\right\| > \varepsilon$

$d^{(k)} = -\nabla f(x^{(k)})$

déterminer $\sigma_k > 0$

$x^{(k+1)} = x^{(k)} + \sigma_k\, d^{(k)}$

$k = k + 1$

2.1 Exemple

Le problème est de trouver la valeur de x qui minimise $E(x)$. Dans cette exemple, on connaît l'expression analytique de la fonction E :

$$E(x) = x^4 - 11x^3 + 41x^2 - 61x + 30$$

On connaît aussi sa dérivée:

$$E'(x) = 4x^3 - 33x^2 + 82x - 61$$

Pour trouver analytiquement le minimum de la fonction E, il faut trouver les racines de l'équation $E'(x) = 0$, donc trouver les racines d'un polynôme de degré 3, ce qui est difficile .
Donc on va utiliser la DG. La DG consiste à construire une suite de valeurs x_i (avec x_0 fixé au hasard) de manière itérative:

$$x_{i+1} = x_i - \rho E'(x_i)$$

Code : Scilab

```
function [xn,fxn,iter]=gradient(x0,f,df,rho,stop,nmax)
iter=0;
xn = x0;
for i=1:nmax
xnp1 = xn - rho*df(xn);
fxn=f(xn);

if norm(xnp1-xn) ¡ stop then return;end
xn = xnp1;
iter=iter+1
end
mprintf('Arret sur nombre maximum d"itérations)
endfunction

function y=f(x)
y = x^4 - 11 * x^3 + 41 * x^2 - 61 * x + 30
endfunction

function y=df(x)
y = 4 * x^3 - 33 * x^2 + 82 * x - 61
endfunction

[xn, fxn, iter]=gradient(5,f,df,0.001,1.e-3,100)
```

3 Méthode du gradient conjugué

Soit $f(x) = \frac{1}{2}x^t Ax - b^t x$, avec A symétrique définie positive.
On sait alors qu'il existe un minimum unique sur $\mathbb{R}^n$ donné par $x^* = A^{-1}b$.
Dans la suite on note $r(x) = Ax - b = \nabla f(x)$ le résidu.

Dans l'algorithme du gradient conjugué linéaire on prend

$$d^{(k)} = -\nabla f(x^{(k)}) + \beta_k \, d^{(k-1)}$$

avec β_k tel que $d^{(k)t} A \, d^{(k-1)} = 0$, on en déduit

$$\beta_k = \frac{r(x^{(k)})^t A \, d^{(k-1)}}{d^{(k-1)t} A \, d^{(k-1)}} \, .$$

Comme $< d^{(k)}, \nabla f(x^{(k)}) >= - \left\| r(x^{(k)}) \right\|_2^2$, on obtient bien une direction de descente.

Algorithme (Gradient conjugué linéaire)

choisir $x^{(0)}$ *et poser* $k = 0$

$r^{(0)} = Ax^{(0)} - b$, $d^{(0)} = -r^{(0)}$, $k = 0$

tant que $\left\| r(x^{(k)}) \right\| > \varepsilon$

$$\sigma_k = \frac{r^{(k)^t} r^{(k)}}{d^{(k)^t} A\, d^{(k)}}$$

$$x^{(k+1)} = x^{(k)} + \sigma_k\, d^{(k)}$$

$$r^{(k+1)} = r^{(k)} + \sigma_k\, A\, d^{(k)}$$

$$\beta_{k+1} = \frac{r^{(k+1)^t} r^{(k+1)}}{r^{(k)^t} r^{(k)}}$$

$$d^{(k+1)} = -r^{(k+1)} + \beta_{k+1}\, d^{(k)}$$

$$k = k + 1$$