Ecole de recherche ENS Lyon – janvier 2011
TP optimisation
Claude Lemaréchal, Jérôme Malick

# TP – algorithme d'optimisation différentiable : Quasi-Newton

**Objectif :** Ce TP consiste à programmer et tester une version simple d'un algorithme important d'optimisation différentiable sans contrainte : la méthode de quasi-Newton. Ce TP requiert peu de programmation, mais permet de manipuler les concepts de base de l'optimisation et de fixer les idées sur les ingrédients des algorithmes d'optimisation.

La méthode de quasi-Newton est un algorithme de descente, dont une itération est shématiquement :
- (estimation) calculer une direction de descente $d_k$,
- (correction) calculer un bon itéré suivant $x_{k+1} = x_k + t_k d_k$ le long de la demi-droite.

Nous nous concentrerons sur le calcul de la direction ; la fonction de calcul du pas (plus technique) est fournie. Récupérer le répertoire zippé avec les fonctions annexes à l'adresse :

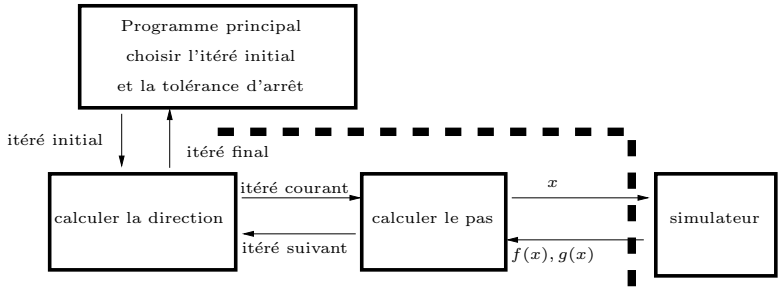
`http://bipop.inrialpes.fr/people/malick/ens11.zip`

Le travail se déroule en 3 étapes :

1. Création de fonctions-tests
2. Implémentation d'un algorithme de descente simple (gradient)
3. Modification pour en faire l'algorithme de Quasi-Newton, tests et comparaisons.

## 1. Fonctions-tests

L'ensemble d'un programme d'optimisation se présente avec deux parties bien distinctes :
- un simulateur : il est chargé de calculer la fonction (ainsi que le gradient et éventuellement le hessien) en chaque point décidé par l'algorithme. Souvent la fonction à minimiser n'est connue que via ce simulateur.
- l'algorithme d'optimisation proprement dit : il comporte deux boîtes principales correspondant au calcul de la direction, et au calcul du pas.




Dans cette première partie, on programme les simulateurs des deux fonctions sur lesquelles on testera les algorithmes. (Bien sûr, on trouve à la main les minimums globaux de ces fonctions !)

**Exercice 1 – Simulateurs.** Écrire un simulateur pour les fonctions suivantes :

$$\begin{aligned} f_1(x) &= \sum_{k=1}^{n} k\, {x_k}^2 \qquad \text{pour } x \in \mathbb{R}^n, \\ f_2(x) &= (1-x_1)^2 + 100(x_2 - {x_1}^2)^2 \qquad \text{avec } x = (x_1, x_2) \in \mathbb{R}^2. \end{aligned}$$

Les simulateurs se présentent sous le format

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

où `x` est la valeur de la variable pour laquelle il faut évaluer $f$ et son gradient, `f` vaut $f(x)$ et `g` vaut $g(x) = \nabla f(x)$. Recommendation : travailler avec des vecteurs colonnes.

**Exercice 2 – Lignes de niveaux.** Visualiser la géométrie des fonctions en affichant quelques lignes de niveaux de $f_1$ pour $n = 2$ et pour $f_2$. Le tracé des lignes de niveaux se fait avec la fonction `tracer.sci` (fournie). Pour la fonction $f_2$, tracer les lignes de niveaux $[1, 2, 5, 10, 20, 30, 50, 100, 200]$ pour bien visualiser la géométrie de la fonction (choisir une bonne résolution `nx,ny`).

## 2. Méthode de descente basique : gradient

Commençons sur un algorithme d'optimisation simple : la méthode du gradient à pas constant, dont l'itération est

$$x_{k+1} = x_k - t_k \nabla f(x_k).$$

**Exercice 3 – Gradient à pas constant.** Implementer la méthode de gradient avec $t_k = 1$. Faire donc une fonction qui, parmi ses paramètres d'entrée, comportera $x_0$ l'itéré courant et `sim` le nom formel du simulateur. Vous pouvez faire afficher à cette fonction des informations à chaque itération (comme la valeur de $f$, $\|g\|$, nombre d'évaluations du simulateur,...).

**Exercice 4 – Tests.**

**a)** Appliquer le programme précédent pour minimiser $f_1$ en dimension 2 à partir de $x_0 = (-1, -1)$. Tester plusieurs pas de descente : commencer par $t = 0.1$, puis $t = 0.5$ et $t = 1$. Que ce passe-t-il ?

**b)** De même pour $f_2$ : tester differents points initiaux, différentes valeurs de $t$ et différents nombres maximaux d'itérations. Prendre en particulier $x_0 = (-1, 1.2)$, $t = 0.001$ et un nombre d'itérations très important, genre 20000.

**c)** Améliorer (un peu) le choix du pas, par exemple en diminuant $t$ jusqu'à obtenir un $x_{k+1}$ meilleur que $x_k$, ou alors en prenant $t = 1/k$ à la $k$-ième itération.

**Exercice 5 – Gradient à pas intelligent.** Récupérer la fonction scilab `wolfe.sci` qui effectue une recherche linéaire de wolfe (correction d'une méthode de descente). Utiliser cette fonction pour calculer le pas $t_k$. Observer l'amélioration sur $f_1$ et $f_2$.

## 3. Méthode de descente de type quasi-Newton : BFGS

La méthode de BFGS consiste à réaliser l'itération

$$x_{k+1} = x_k - t_k W_k \nabla f(x_k)$$

où $t_k$ est donné par la recherche linéaire de Wolfe et la matrice symétrique définie positive $W_k$ est calculée par la formule de récurrence

$$W_{k+1} = W_k - \frac{s_k y_k^T W_k + W_k y_k s_k^T}{y_k^T s_k} + \left[1 + \frac{y_k^T W_k y_k}{y_k^T s_k}\right] \frac{s_k s_k^T}{y_k^T s_k},$$

avec $s_k = x_{k+1} - x_k$ et $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$. Le schéma général d'une méthode de quasi-Newton est alors :

1. avec l'itéré initial $x_0$, se donner une matrice initiale $W_0$ symétrique définie positive ;
2. connaissant le gradient $\nabla f(x_k)$, calculer la direction $d_k = -W_k \nabla f(x_k)$ ;
3. calculer le pas $t_k$ par recherche linéaire de Wolfe ;
4. connaissant le nouvel itéré $x_{k+1}$, appeler le simulateur et calculer la nouvelle matrice $W_{k+1}$.

**Exercice 6 – BFGS.** Programmer la méthode BFGS : pour cela, récuperer la méthode du gradient avec recherche lineaire de Wolfe, et modifier la direction en ajoutant la matrice de Quasi-Newton. On prendra à la matrice identité (commande `eye` en scilab) pour $W_0$ et un pas initial égal à 1 en entrée de la recherche linéaire de Wolfe.

**Exercice 7 – Tests.**

**a)** Évaluer ce code avec $f_2$ partant du point initial $(-1, 1.2)$. Comparer avec les autres méthodes (gradient à pas constant, gradient avec Wolfe, quasi-Newton à pas constant...). Même chose avec d'autres points initiaux (par exemple $(0, 1.2)$).

**b)** Estimer les vitesses de convergence asymptotique - en tracant par exemple la suite des valeurs $\log f(x_k)$. Comparer la vitesse de convergence avec celle de la methode de gradient avec Wolfe (que l'on récupère en enlevant la mise à jour de $W_k$ dans quasi-Newton).

**c)** Visualiser les déplacements des itérés sur un graphique de la fonction.

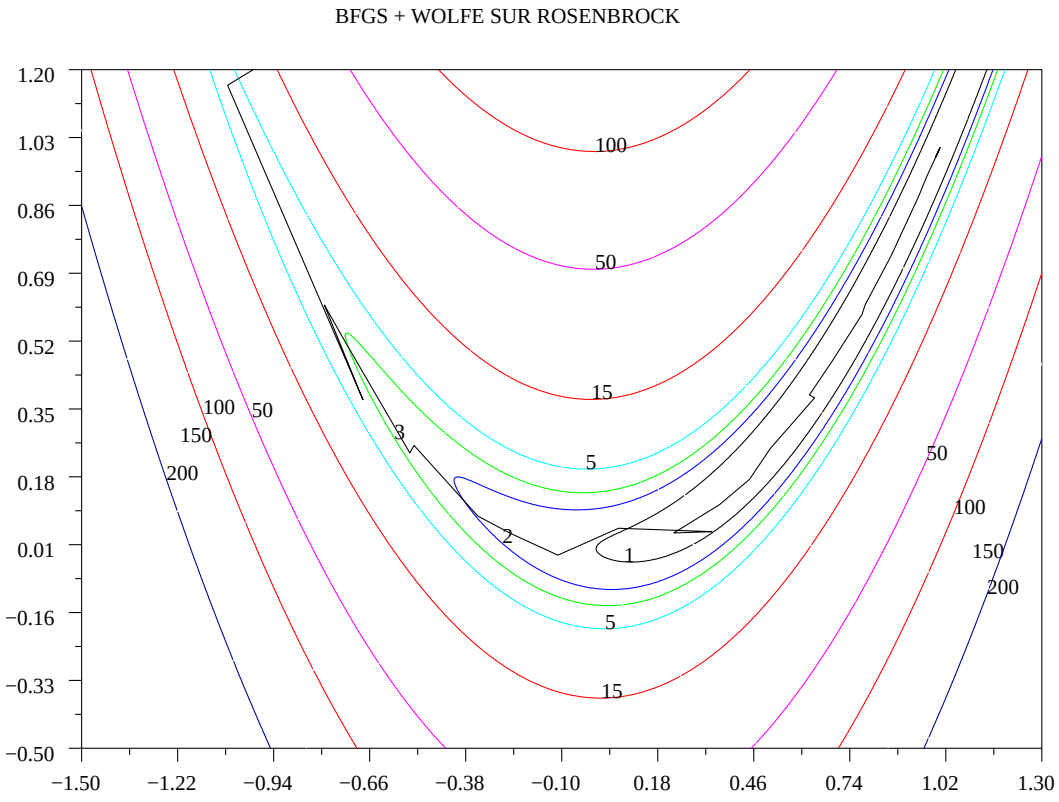



**Fig. 1** Minimisation de la fonction Rosenbrock par BFGS, en partant de $x_0 = (-0.9, 1.2)$