1 Algorithme de gradient

On cherche ici à programmer un algorithme de gradient à pas constant ou variable pour un problème quadratique :

$$\min_{x\in\mathbb{R}^N} \frac{1}{2}(x,Ax) - (b,x) \ , \tag{1}$$

où A est une matrice symétrique définie positive de taille $N \in \mathbb{N}^*$. On pose $J(x) = \frac{1}{2}(x,Ax) - (b,x)$. L'algorithme est le suivant :

$$\begin{cases} \text{Données : } J,\ \nabla J \\ \text{Initialisation : Choix de } x_0 \\ \text{Itération } k : x_k \text{ et le pas } \rho_k \text{ étant connus, calculer } d_k = -\nabla J(x_k), \text{ puis } x_{k+1} = x_k + \rho_k d_k \end{cases}$$

```
N=10;
A=toeplitz([2,-1,zeros(1,N-2)]);
b = rand(N,1);


function y=J(x,A,b)
  y = (1/2)* x'*A*x  - b'*x;
endfunction



function dy=gradJ(x,A,b)
  y = A*x-b;
endfunction
```

1.1 Programmation de l'algorithme de gradient à pas constant

Programmer un algorithme de gradient avec un pas constant.
Les données sont la fonction coût `J`, son gradient `gradJ`, le premier terme `x0` de la suite $(x_k)_k$ et les critères d'arrêt : un nombre d'itération maximal `nmax` ou une borne maximale pour l'erreur entre deux itérés `stop`. Le pas `rho` devra être donné dans le script de la fonction.

- Remarque: Il faut évaluer numériquement la borne maximale du pas ρ (cf exercice 1 du TD3).

```
function rho_max=rmax(A)
  // valeur propre maximale de A
```

```
  alpha = max(spec(A));
  // contrainte sur le pas de gradient
  rho_max= 2/alpha;
endfunction


function [c,xn,Jxn]=gradient_pasconstant(x0,J,dJ,stop,nmax)
  // a la fin de l'algorithme c est un vecteur colonne contenant toutes les valeurs J(xk)
  // xn et Jxn sont les valeurs finales obtenues
  // stop : valeur num\'erique du crit\`ere d'arret
  // nmax : nombre d'it\'erations maximales
  // rho : choisir une valeur strictement positive mais inf\'erieur \`a rho_max



endfunction
```

1.2 Vérification du code

Calculer la solution du problème d'optimisation revient ici à résoudre un système linéaire (cf TD1). On pourra comparer la solution du problème d'optimisation avec la solution obtenue en utilisant des algorithmes de résolution de systèmes linéaires de Scilab.

La solution exacte $x_* = A^{-1}b$ s'obtient avec la fonction Scilab `inv`

```
x_opt = inv(A)*b;
```

On peut aussi résoudre le système $Ax = b$ avec la commande \

```
x_opt = A \ b;
```

ou encore la fonction `linsolve`

```
x_opt = linsove(A , b, x0);
```

```
x0=zeros(N,1);
[c,xn,Jxn]=gradient_pasconstant(x0,J,dJ,rho,stop,nmax);
norm(xn-x_opt,2)
```

On pourra tracer au cours des itérations l'évolution de la fonction coût

```
Nit = size(c,1);
plot([0:Nit-1]',c)
```

1.3 Pas constant optimal

• Comparer les courbes des erreurs selon le choix du pas constant ρ. Pour cela vous devez modifier la fonction `gradient_pasconstant.sci`

```
function [c,xn,Jxn,errors]=gradient_pasconstant(x0,J,dJ,stop,nmax,x_opt)
  // a la fin de l'algorithme c est un vecteur colonne contenant toutes les valeurs J(xk)
  // a la fin de l'algorithme errors est un vecteur colonne contenant toutes les erreurs
  // ||xk-x_opt|| pour la norme euclidienne
  // xn et Jxn sont les valeurs finales obtenues
  // stop : valeur num\'erique du crit\'ere d'arret
  // nmax : nombre d'it\'erations maximales
  // rho : choisir une valeur strictement positive mais inf\'erieur \'a rho_max



endfunction
```

Remarque : la norme euclidienne de x, $\sqrt{\sum_{i=1}^{N} x_i^2}$ se calcule en entrant `norm(x,2)`

• Vérifier si les meilleurs résultats sont obtenus lorsque $\rho = \dfrac{2}{\lambda_1 + \lambda_n}$, où λ_1 et λ_n sont respectivement la plus petite et la plus grande valeur propre (cf exercice 1 du TD3).

```
function rho_opt=pas_cst_opt(A)
// pas optimal pour la matrice A
vp_max = max(spec(A));
vp_min  = min(spec(A));
rho_opt= 2/(vp_max+vp_min);
endfunction
```

1.4 Algorithme de gradient à pas optimal

Ici le choix du pas change à chaque itération et vérifie

$$\rho_k := \underset{\rho>0}{\operatorname{argmin}}\ J(x_k + \rho d_k)\ .$$

Nous avions trouvé:

$$\rho_k = \frac{d_k^{\mathsf{T}} d_k}{d_k^{\mathsf{T}} A d_k}.$$

• Ecrire une fonction qui calcul le pas variable optimal étant donnés d_k et A, puis Implémenter la méthode du gradient à pas optimal.

```
function rho_opt=pas_var_opt(dk, A)
// pas variable optimal pour la matrice A
```

```
endfunction

function [c,xn,Jxn,errors]=gradient_pasoptimal(x0,J,dJ,stop,nmax,x_opt)
   // a la fin de l'algorithme c est un vecteur colonne contenant toutes les valeurs J(xk)
  // a la fin de l'algorithme errors est un vecteur colonne contenant toutes les erreurs
  // ||xk-x_opt|| pour la norme euclidienne
  // xn et Jxn sont les valeurs finales obtenues
  // stop : valeur num\'erique du crit\'ere d'arret
  // nmax : nombre d'it\'erations maximales
  // rho : le pas est calcul\'e \'a chaque it\'eration \'a l'aide de la fonction pas_var_opt



endfunction
```

• Comparer les vitesses de convergence des deux méthodes de gradient.

2 Recherche linéaire inexacte

La recherche linéaire consiste à déterminer, à chaque itération $k \geq 1$, le pas ρ_k à effectuer le long de la direction de descente d_k. Il existe différentes méthodes de recherche linéaire: la recherche linéaire exacte, la règle d'Armijo, la règle de Goldstein ou la règle de Wolfe.

La recherche linéaire *exacte* consiste à déterminer

$$\rho_k := \underset{\rho \leq 0}{\operatorname{argmin}} f(x_k + \rho\, d_k) \ .$$

Pour les fonctions quadratiques, on peut déterminer l'expression exacte du pas optimal (cf TD3), ce qui donne lieu à la méthode du *gradient à pas optimal*. Cependant, dans le cas général, on ne peut que déterminer numériquement (avec une méthode de Newton par exemple) une approximation de ce pas optimal. Cette étape peut s'avérer très couteuse sans grandement améliorer la convergence des algorithmes d'optimisation. On utilise alors d'autres méthodes de recherche linéaire dite *inexacte* qui sont toutes aussi performantes.

2.1 La règle de Goldstein

Soient m_1 et m_2 deux nombres réels tels que $0 < m_1 < m_2$. À l'itération $k \geq 1$, on pose $\varphi(\rho) = J(x_k + \rho d_k)$. Les conditions de Goldstein sont les suivantes

(G1) $$\text{Si} \quad \varphi(\rho) > \varphi(0) + m_1\varphi'(0)\rho \ , \quad \text{alors } \rho \text{ est trop grand} \ ,$$

(G2) $$\text{Si} \quad \varphi(\rho) < \varphi(0) + m_2\varphi'(0)\rho \ , \quad \text{alors } \rho \text{ est trop petit} \ ,$$

(G3) $$\text{Si} \quad \varphi(0) + m_2\varphi'(0)\rho \leq \varphi(\rho) \leq \varphi(0) + m_1\varphi'(0)\rho \ , \quad \text{alors } \rho \text{ convient} \ .$$

• Écrire un script qui détermine un pas ρ satisfaisant la règle de Goldstein. L'algorithme est le suivant

$$\left\{\begin{array}{l} \text{Données : } J,\ \nabla J(x_k),\ x_k,\ d_k, m_1 \text{ et } m_2 \\ \text{Initialisation : } Rit := 0,\ \alpha := 0,\ \beta := +\infty,\ \text{ choisir } \rho_k \neq 0 \\ \text{Tant que } \rho_k \text{ ne satisfait pas les conditions de Goldstein (G3):} \\ \qquad \text{(i) si } \rho_k \text{ statisfait la condition (G1) alors on pose } \beta = \rho_k \text{ puis } \rho_k = \frac{\alpha+\beta}{2} \\ \qquad \text{(ii) sinon, si } \rho_k \text{ satisfait (G2) alors on pose } \alpha = \rho_k \text{ puis } \rho_k = \left\{\begin{array}{l} \frac{\alpha+\beta}{2} \text{ si } \beta < +\infty \\ 2\rho_k \text{ si } \beta = +\infty \end{array}\right. \\ \qquad \text{(iii) } Rit := Rit + 1 \\ \text{fin!} \end{array}\right.$$

NB: Vous devez calculer $\varphi(0) = J(x_k)$ et $\varphi'(0) = \nabla J(x_k) \cdot d_k$.

```
function rhok=Goldstein(J,dJxk,xk,dk,m1,m2)
 // calcule le pas rhok d'apr\'es la r\'egle de Goldstein


endfunction
```

• La valeur $+\infty$ s'obtient avec Scilab en entrant `%inf` .

• On prend en général $m_1 = 0.1$ et $m_2 = 0.7$.

2.2 La règle de Wolfe

Les conditions de Wolfe sont les suivantes

(W1) $$\text{Si} \quad \varphi(\rho) > \varphi(0) + m_1\varphi'(0)\rho\ , \quad \text{alors } \rho \text{ est trop grand ,}$$

(W2) $$\text{Si} \quad \varphi(\rho) \leq \varphi(0) + m_1\varphi'(0)\rho \quad \text{et} \quad \varphi'(\rho) < m_2\varphi'(0)\ , \quad \text{alors } \rho \text{ est trop petit ,}$$

(W3) $$\text{Si} \quad \varphi(\rho) \leq \varphi(0) + m_1\varphi'(0)\rho \quad \text{et} \quad m_2\varphi'(0) \leq \varphi'(\rho)\ , \quad \text{alors } \rho \text{ convient .}$$

• Écrire un script qui détermine un pas ρ satisfaisant la règle de Wolfe. Il suffit de modifier les instructions conditionnelles dans le script précédent.

NB: Vous devez calculer $\varphi'(\rho) = \nabla J(x_k + \rho d_k) \cdot d_k$.

```
function rhok=Wolfe(J,dJ,xk,dk,m1,m2)
 // calcule le pas rhok d'apr\'es la r\'egle de Wolfe


endfunction
```

• On prend en général $m_1 = 10^{-4}$ et $m_2 = 0{,}9$.

2.3 Généralisation des méthodes de Gradient aux fonctions non quadratiques

• Implémenter une méthode de gradient à pas *intelligent*. L'algorithme est le suivant

$$\begin{cases} \text{Données : } J,\ dJ,\ m_1,\ m_2 \\ \text{Initialisation : Choix de } x_0 \\ \text{Itération } k : x_k \text{ étant connu,} \\ \qquad\qquad \text{calculer } d_k = -\nabla J(x_k), \\ \qquad\qquad \text{déterminer le pas } \rho_k \text{ selon la règle de votre choix,} \\ \qquad\qquad \text{puis } x_{k+1} = x_k + \rho_k d_k \end{cases}$$

Il suffit de modifier le choix du pas dans le script du gradient à pas optimal.

```
function [xn,fxn,fx_all]=Gradient(x0,J,dJ,stop,nmax,type,m1,m2)
  // a la fin de l'algorithme fx_all contiendra les valeurs f(xk)
  // xn et fxn sont les valeurs finales obtenues
  // stop : valeur num\'erique du crit\'ere d'arret
  // nmax : nombre d'it\'erations maximales
  // type indique la r\'egle de recherche lin\'eaire
  // de votre choix ('G' pour Golstein et  'W' pour Wolfe)


endfunction
```

• Tester votre algorithme sur la fonction de Rastrigin suivante

$$f_2(x) = n + \sum_{i=1}^{N} \left(x_i^2 - \cos(2\pi x_i)\right), \text{ pour } x = (x_1, \ldots, x_n) \in \mathbb{R}^N$$

Cette fonction est très utile lorsqu'il s'agit de tester des algorithmes d'optimisation. Elle admet plusieurs minimum locaux et un minimum global en l'origine. Le script de la fonction de rastrigin et de son gradient son dans le fichier `BFGS.sci`

3 Une méthode de type quasi-Newton

L'algorithme de Newton en optimisation est une application directe de l'algorithme de Newton pour la résolution d'équations du type : $F(x) = 0$. En optimisation sans contrainte, i.e. pour résoudre le problème $x_* := \underset{x\in\mathbb{R}^n}{\operatorname{argmin}} f$, l'algorithme de Newton cherche les solutions de l'équation : $\nabla f(x) = 0$.

En supposant la fonction à minimiser f de classe $\mathscr{C}^2$ et la matrice Hessienne $H_f(x_k)$ inversible, une itération de l'algorithme de Newton s'écrit : $x_{k+1} = x_k - H_f(x_k)^{-1}\nabla f(x_k)$. La direction de descente est alors $d_k = -H_f(x_k)^{-1}\nabla f(x_k)$. Dans le cas où la matrice $H_f(x_k)$ est définie positive à chaque itération, la méthode de Newton est une méthode de descente à pas constant égal à 1. Le principal atout de cette méthode est sa convergence quadratique. Ses principaux inconvénients sont qu'elle nécessite le calcul des expressions analytiques des dérivées secondes (pas toujours possible), l'inversion d'une matrice et l'abscence de convergence si le premier itéré est trop loin de la solution (convergence locale). Cet algorithme ne fait pas la différence entre minima, maxima et points stationnaires.

3.1 Algorithme de Broyden, Fletcher, Goldfarb et Shanno (BFGS)

Pour forcer la convergence globale de l'algorithme de Newton on remplace les itérations par

$$x_{k+1} = x_k - \rho_k H_k^{-1} \nabla J(x_k) \ ,$$

où H_k et une approximation de la Hessienne de J évaluée en x_k, notée $H_J(x_k)$, et ρ_k est le pas calculé par recherche linéaire. Ces matrices sont construites de telles sorte qu'elles soient symétriques définies positives. On obtient ainsi l'un des algorithmes les plus performants: l'algorithme de Broyden, Fletcher, Goldfarb et Shanno.

$$\left\{ \begin{array}{l} \text{Données : } J,\ dJ,\ m_1,\ m_2 \\ \text{Initialisation : Choix de } x_0 \text{ et } H_0 \text{ définie positive quelconque,} \\ \text{Itération } k : x_k \text{ et } H_k \text{ étant connus,} \\ \qquad \text{calculer } d_k = -H_k^{-1} \nabla J(x_k), \\ \qquad \text{déterminer le pas } \rho_k \text{ selon la règle de votre choix,} \\ \qquad \text{puis } x_{k+1} = x_k + \rho_k d_k \\ \qquad \text{Calculer } s_k = x_{k+1} - x_k \text{ et } y_k = \nabla J(x_{k+1}) - \nabla J(x_k) \\ \qquad H_{k+1} = H_k + \dfrac{y_k y_k^T}{y_k^T s_k} - \dfrac{H_k s_k s_k^T H_k}{s_k^T H_k s_k} \ . \end{array} \right.$$

• Exécuter le fichier `BFGS.sci` . Utiliser vos algorithmes de recherche linéaire pour le calcul du pas ρ_k à chaque itération et modifier seulement la ligne 59 qui calcule les nouveaux itérés x_{k+1} et $J(x_{k+1})$ connaissant x_k, J, ∇J et d_k.

Attention : Ici la direction d_k n'est plus égale à $-\nabla J(x_k)$.

4 Travail à rendre

Le but de ce TP est de comparer les vitesses de convergence des algorithmes de gradient et de BFGS selon le choix du pas qu'il soit constant ou choisi selon la règle de Goldstein ou de Wolfe. Dans un court rapport de 4 pages maximum vous présenterez vos observations et critiques concernant les points suivants.

P1. Pour $N = 2$ et chacune des deux fonctions quadratique J et de Rastrigin, vous dresserez un tableau présentant le nombre d'itérations nécessaires pour atteindre différentes précisions (`stop`$= 0.1,\ 10^{-2}$, 10^{-4}, 10^{-6} ou 10^{-9}). Prendre un critère d'arrêt `nmax` très grand (20 000 par exemple).

Testez différents points initiaux. Conclure

P2. Pour $N = 2,\ 5,\ 10,\ 20,\ 40$, etc... résoudre le problème de minimisation avec la fonction f_2. Pour toutes les simulations reporter le nombre d'itérations, le temps de calcul nécessaire pour trouver la solution avec une précision à 10^{-3} près et tracer la courbe qui à N associe le temps de calcul. Faire de même pour la courbe qui à N associe le nombre d'itérations. Conclure.

P3. Tracer par exemple la suite des valeurs $f(x_k)$ en fonction de k pour comparer les vitesses de convergence.

P4. Dans le cas d'une fonction quadratique, comparer les vitesses de convergence des algorithmes de gradient avec pas de Wolfe et pas variable optimal. Conclure sur la performance de la règle de Wolfe.

P5. Faites varier les valeurs de m_1 et m_2. Conclure

P6. Comparer les performances de BFGS avec lalgorithme de gradient.

Ce compte-rendu (format NomsBinome.zip) avec les commentaires (saisis en Word ou LaTeX) et figures est à envoyer à l'adresse électronique `frederique.le-louer@utc.fr` avant le 18 juin minuit.