Algorithmes de gradient à pas fixe
suite

J.Ph. Chancelier*

23 octobre 2003

Le but de cette première séance est double, se refamiliariser avec la programmation en Scilab, commencer à voir la mise en oeuvre d'algorithmes d'optimisation.

1 Minimisation de fonctionnelles quadratique

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

$$\min_{x\in\mathbb{R}^N} \frac{1}{2}\langle x, Ax\rangle + \langle b, x\rangle \tag{1}$$

Écrire une première fonction qui tire de façon aléatoire un matrice `N x N` définie positive dont on peut contrôler la plus petite valeur propre (assurer qu'elle est plus grande qu'une valeur donnée) (`rand`, `diag`, `inv`)

```
function A=defpos(n,lmin,lmax)
  // tirage aléatoire d'une matrice définie positive
  // dont le spectre aléatoire est réel uniforme sur [lmin,lmax]
  A=diag( (lmax-lmin)*rand(N,1)+lmin);
  P=rand(N,N);
  A=P*A*inv(P);
endfunction

function A=defpos\_sym(n,lmin,lmax)
  // tirage aléatoire d'une matrice définie positive et symetrique
  // dont le spectre aléatoire est réel uniforme sur [lmin,lmax]
  A=diag((lmax-lmin)*rand(N,1)+lmin);
  [U,H]=hess(rand(N,N));
  A=U*A*U';
endfunction
```

*Cermics, École Nationale des Ponts et Chaussées, 6 et 8 avenue Blaise Pascal, 77455, Marne la Vallée, Cedex 2

A=defpos_sym(N,2,3) ; b=rand(N,1) ;

Une remarque : génériquement une matrice aléatoire (ici avec loi uniforme) est inversible :

```
ne=100;
for i=1:ne
  A1=rand(N,N);
  if rank(A1)<>N then pause;
  end;
end
```

Écrire une fonction Scilab qui calcule le critère (1) et une fonction qui calcule le gradient du critère.

```
function y=f(x,A,b)
  y = (1/2)* x'*A*x  + b'*x
endfunction

function y=df(x,A,b)
  y = (A+A')*x/2  + b;
endfunction
```

Programmer un algorithme de gradient à pas constant sur le problème quadratique. Il faudra évaluer numériquement la borne maximale du pas de gradient et choisir un critère d'arrêt. On pourra aussi tracer au cours des itérations l'évolution de la fonction coût et évaluer les temps de calculs (`timer`)

```
function rho_max=rmax(A)
  alpha = min(abs(spec((A+A')/2)))
  // valeur de Lipschitz du gradient
  C= norm((A+A')/2,2)
  // contrainte sur le pas de gradient
  rho_max= 2*alpha/C^2
endfunction

function [c,xn,fxn]=gradient(x0,f,df,rho,stop,nmax,A,b)
  // a la fin de l'algoithme c contiendra les valeurs f(xi,...)
  // xn et fxn sont les valeurs finales obtenues
  // rho : pas de gradient
  // stop : valeur numérique du critère d'arret
  // nmax : nombre d'itérations maximales
endfunction
```

Rendre le programme plus générique

```
function [c,xn,fxn]=gradient(x0,f,df,rho,stop,nmax,varargin)
  // a la fin de l'algoithme c contiendra les valeurs f(xi,...)
  // xn et fxn sont les valeurs finales obtenues
  // rho : pas de gradient
```

```
  // stop : valeur numérique du critère d'arret
  // nmax : nombre d'itérations maximales
  .... f(x,varargin(:))....
endfunction
```

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

1.1 Rajout d'une contrainte

On rajoute maintenant au problème d'optimisation une contrainte linéaire de la forme $Tx = 0$. Pour que l'ensemble admissible ne soit pas réduit à $x = 0$, il faut bien s'assurer que la matrice T à un noyau de dimension non nulle.

Écrire une fonction qui tire de façon aléatoire un matrice `N x N` dont le rang est par exemple `N/2` (indication : si `M` est une matrice `n x m` aléatoire quel est le rang générique de `M*M'` ?).

Réprendre la première partie avec un nouveau critère d'optimisation ou on pénalise la contrainte $Tx = 0$ en rajoutant au critère un terme $(1/(2\epsilon)) * \|Tx\|$:

$$\min_{x\in\mathbb{R}^N} \frac{1}{2}\langle x, Ax\rangle + \langle b, x\rangle + \frac{1}{2\epsilon}\langle Tx, Tx\rangle \tag{2}$$

1.2 Un peu d'algèbre linéaire

En utilisant la fonction `colcomp` (compression de colonnes) On peut obtenir une base du noyau de l'opérateur T. Cela permet d'écrire les vecteurs x admissibles du problème avec contraintes $Tx = 0$ sous la forme $x = Wy$. On se ramène ainsi à un problème d'optimisation sans contraintes en y. Le programmer et comparer avec la section précédente.

Que se passe-t-il si le rang de la matrice T n'est pas égale au nombre de lignes de T ? (c'est d'ailleurs ce qui se passe avec la méthode que nous avons utilisé pour obtenir `T`).

Se ramener à une structure de contrainte ou `T` est de rang plein (`rowcomp`).

Résoudre avec Scilab le système linéaire :

$$\begin{pmatrix} A & T' \\ T & 0 \end{pmatrix}\begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} -b \\ 0 \end{pmatrix} \tag{3}$$

2 Un problème de fil pesant (d'après Ciarlet, Joly. Ensta)

On se propose maintenant de trouver la position d'équilibre d'un fil pesant fixé à ses deux extrémités en résolvant un problème de minimisation. le fil pesant est caractérisé par son élévation $v(x)$. Il est fixé a la position $x = 0$ à l'élévation $v(0) = a$ et à la position $x = 1$ à l'élévation $v(1) = b$. La longueur du fil est fixée L. En minimisant l'énergie potentielle du fil on est conduit au problème de minimisation suivant :

$$\min_{\Theta(v)=0} \int_0^1 v(x)(1 + v'(x)^2)^{1/2} dx \tag{4}$$

où les contraintes $\Theta(v) = 0$ sont :

$$\begin{aligned} v(0) - a &= 0 \\ v(1) - b &= 0 \\ \int_0^1 (1 + v'(x)^2)^{1/2} dx - L &= 0 \end{aligned}$$

Pour résoudre ce problème numériquement nous allons tout d'abord éliminer la contrainte portant sur la longueur de la corde par *pénalisation*. On va remplacer le problème initial par :

$$\min_{v(0)=a, v(1)=b} \int_0^1 v(x)(1 + v'(x)^2)^{1/2} dx + \frac{1}{\epsilon}(\int_0^1 (1 + v'(x)^2)^{1/2} dx - L)^2 \tag{5}$$

Pour se ramener à un problème d'optimisation dans un espace de dimension finie, on va chercher v sous la forme d'une fonction affine par morceaux. Soient $(x_i)_{i=0,N}$ des points régulièrement espacés sur $[0, 1]$, $x_i = ih$ ($h = 1/N$). On cherche $v_h(x)$ affine par morceaux sur les intervalles $[x_i, x_{i+1}]$. Soit $(v_i)_{i=0,N}$ le vecteur des valeurs de v au points x_i, v_h s'écrit :

$$v_h(x) = v_i + (x - x_i)(v_{i+1} - v_i) \quad \text{sur} \quad [x_i, x_{i+1}] \tag{6}$$

Le problème d'optimisation sur $\mathbb{R}^{N+1}$ s'écrit :

$$\min_{v_0=a, v_N=b} J(v) \tag{7}$$

avec

$$\begin{aligned} J(v) &= h \sum_{i=0}^{N-1} \frac{(v_i + v_{i+1})}{2} \left(1 + \left(\frac{v_{i+1} - v_i}{h}\right)^2\right)^{1/2} + \frac{1}{\epsilon}(lc(v, h) - L)^2 \\ lc(v, h) &= h \sum_{i=0}^{N-1} \left(1 + \left(\frac{v_{i+1} - v_i}{h}\right)^2\right)^{1/2} - L \end{aligned}$$

2.1 Un algorithme de gradient

Pour résoudre numériquement ce problème on va utiliser une méthode de gradient. On notera que les deux contraintes qui restent $v_0 = a$ et $v_N = b$ ne seront pas traités comme des contraintes d'un problème d'optimisation. Ces deux valeurs de v étant connues le problème d'optimisation se pose sur $\mathbb{R}^{N-1}$.

L'algorithme est donc :

$$v^{k+1} = v^k - \rho \Delta J(v^k) \tag{8}$$

2.2 Résolution numérique avec Scilab

Comme on le voit sur l'équation (8) l'algorithme de gradient est un algorithme vectoriel, le vecteur v est mis à jour à chaque itération. On devra tenir compte de ce fait et programmer l'algorithme de gradient de façon vectorielle en Scilab. La seule boucle `for` du programme sera la boucle d'itération (8).

On pourra au cours de l'algorithme d'optimisation tracer la position de la corde et voir ce tracé évoluer au cours de l'algorithme.

On pourra programmer les fonctions suivantes :

– La fonction coût du problème pénalisé
```
function [j]=J(v,L,h,eps)
...
endfunction
```
– La fonction $(1 + ((x - y)/h).^2)^{1/2}$ pour x et y vectoriels.
```
function [y]=R(v1,v2,h)
...
endfunction
```
– La dérivée de R par rapport à x, à nouveau vectorielle.
```
function [y]=Dr(v1,v2,h)
...
endfunction
```
– la longueur de la corde :
```
function [l]=lc(v,h)
...
endfunction
```
– le gradient de J :
```
function [dj]= dJ(v,L,h,eps)
...
endfunction
```

`v(1:$-1)`	extraction d'un sous vecteur `$` désigne le dernier élément
`.*`	multiplication termes à termes
`.^`	exponentiation termes à termes
`sum`	somme des éléments d'un vecteur
`sqrt`	racine
`linspace`	points régulièrement espacés
`modulo`	fonction modulo
`plot2d`	tracé de courbes 2D

TAB. 1 – Quelques éléments