Introduction à l'optimisation continue
Travaux pratiques, séance du 18 décembre 2018
Implémentation d'algorithmes

1 Introduction

Nous allons tester différents algorithmes pour minimiser des problèmes élémentaires de traitement/reconstruction d'image.

Nous proposons d'étudier :

1. Un problème de débruitage :

$$\min_u \lambda |Du|_{2,1} + \frac{|u-g|_2^2}{2} \tag{1}$$

   où g représente une image "bruitée" avec un bruit Gaussien et u la reconstruction ;
2. Le même avec un bruit impulsionnel et $|u-g|_2^2$ remplacé par $|u-g|_1$;
3. Un problème de déconvolution ou suréchantillonnage :

$$\min_u \lambda |Du|_{2,1} + \frac{1}{2}|Au-g|_2^2 \tag{2}$$

   où A représente différents opérateurs linéaires.

Ici pour $u = (u_{i,j})$ matrice de taille réelle (pour simplifier on ne considère que des images en niveaux de gris) $N \times M$, l'opérateur D est un gradient discret :

$$D : u \mapsto \begin{pmatrix} D^1 u \\ D^2 u \end{pmatrix}$$

où

$$D^1 u = \begin{cases} u_{i+1,j} - u_{i,j} & (i < N) \\ 0 & (i = N) \end{cases}$$

$$D^2 u = \begin{cases} u_{i,j+1} - u_{i,j} & (j < M) \\ 0 & (j = N) \end{cases}$$

Ensuite,

$$|Du|_{2,1} = \sum_{i,j} \sqrt{(D^1 u)_{i,j}^2 + (D^2 u)_{i,j}^2}.$$

La norme $|\cdot|_2$ est la norme Euclidienne standard.

Pour construire une telle matrice efficacement en Matlab, le plus efficace (d'autant qu'on aura aussi besoin de l'adjoint) est de construire à l'aide de la commande sparse une matrice D qui agira sur un vecteur colonne $u(:)$

Pour lire une image, au besoin la convertir en niveaux de gris, et récupérer sa taille :

```
>> u = imread('nom');
>> u = mean(double(u),3);
>> [M N] = size(u);
>> MN = M*N;
```

M est alors le nombre de lignes et N le nombre de colonnes. Si par ailleurs l'image est trop grande la commande `imresize()` permet de la remettre à une échelle raisonnable. Le vecteur colonne `v=u(:)` est alors formé des N colonnes mises bout à bout (la taille totale est MN). L'image est reconstituée par la commande `u=reshape(v,M,N)`.

Pour créer la matrice D, on doit d'abord former une matrice des indices de chaque pixel : `I=reshape([1:M*N],M,N)`. `I(m,n)` contient alors l'indice correspondant au pixel (m, n) dans le vecteur v. On récupère les indices des pixels immédiatement à droite et en dessous en construisant alors les tableaux :

```
>> east=[I(:,2:end), I(:,end)];
>> north=[I(2:end,:); I(end,:)];
>> D1 = sparse(I,east,1,MN,MN) -speye(MN,MN);
>> D2 = sparse(I,north,1,MN,MN) -speye(MN,MN);
>> D = [D1 ; D2];
```

(le fait d'avoir recopié dans `east` et `north` la dernière colonne ou la dernière ligne est juste une astuce pour récupérer un zéro dans la matrice finale par soustraction avec les éléments correspondants dans `speye(MN,MN)`).

On admet que $\|D' * D\| \leq 8$ (la norme de l'opérateur). Vous pouvez le vérifier d'ailleurs avec la commande `normest()`.

On peut maintenant calculer le "gradient" d'une image par `gra=D*u(:)`. Les MN premiers éléments correspondent aux différences horizontales, les MN suivants aux directions verticales, le tout arrangé en colonne.

On obtient donc par exemple la norme du gradient (et on l'affiche) par

```
>> no = reshape(hypot(gra(1:MN),gra(MN+1:end)),M,N);
>> imagesc(no);
```

2 Débruitage par FISTA

2.1 Forward-Backward vs FISTA

Pour ajouter un bruit Gaussien sur une image : `g= u+sigma*randn(size(u));` où `sigma` est la valeur de l'écart-type. On veut maintenant résoudre (1). On écrit

le problème dual

$$\begin{aligned}\min_u \lambda|Du|_{2,1} + \frac{1}{2}|u-g|_2^2 & \\ &= \min_u \sup_{|p|_{2,\infty}\leq\lambda} p\cdot(Du) + \frac{1}{2}|u-g|_2^2 \\ &= \sup_{|p|_{2,\infty}\leq\lambda} \inf_u (D'p)\cdot u + \frac{1}{2}|u-g|_2^2 \\ &= \sup_{|p|_{2,\infty}\leq\lambda} (D'p)\cdot g - \frac{1}{2}|D'p|^2.\end{aligned}$$

On résout donc

$$\min_{|p|_{2,\infty}\leq\lambda} \frac{1}{2}|D'p|^2 - (D'p)\cdot g$$

où la contrainte signifie que $p=(p^1,p^2)$ est tel que $\sqrt{(p^1_{i,j})^2+(p^2_{i,j})^2}\leq\lambda$ pour tous i,j. Le terme quadratique a un gradient L-Lipschitz avec $L=\|D*D'\|\leq 8$.

L'image u est retrouvée en calculant $u=g-D'p$. On calculera alors le gap

$$\lambda|Du|_{2,1} + \frac{1}{2}|u-g|_2^2 - (D'p)\cdot g + \frac{1}{2}|D'p|^2 = \lambda|Du|_{2,1} - (D'p)\cdot u$$

(dont on peut montrer qu'il majore $\|u-\bar{u}\|_2^2$ où $\bar{u}$ est la solution exacte du problème) pour s'en servir comme critère d'arrêt.

1. Écrire l'algorithme "forward-backward" pour résoudre ce problème. Le tester.
2. Écrire l'algorithm FISTA, le tester. Comparer la décroissance de l'énergie.

L'énergie se calcule facilement par (supposant que `p` est une colonne de longueur $2MN$)

```
Energie = sum ((D'*p).^2)/2 - sum((D'*p).*g);
Energies = [Energies , Energie];
```

à condition d'avoir initialisé `Energies = [];`

La partie la plus délicates est d'implémenter correctement et efficacement la projection de p sur la contrainte $|p|_{2,\infty}\leq\lambda$.

2.2 FISTA fortement convexe

On pourra essayer la chose suivante : on régularise légèrement le problème dual en ajoutant un terme $\varepsilon|p|_2^2/2$ où $\varepsilon>0$. Dans ce cas, le problème devient fortement convexe et il faut utiliser (pour calculer la suite "t_k" intervenant dans la sur-relaxation) les formules adaptées : on a

$$y^k = x^k + \beta_k(x^k - x^{k-1})$$

où

$$\beta_k = \frac{t_k - 1}{t_{k+1}}(1 - (t_{k+1} - 1)\varepsilon\tau),$$

$$t_{k+1} = \frac{1 - qt_k^2 + \sqrt{(1 - qt_k^2)^2 + 4t_k^2}}{2}$$

avec $\tau = 1/8$ et

$$q = \frac{\tau\varepsilon}{1 + \tau\varepsilon} = \frac{\varepsilon}{\varepsilon + 8}.$$

1. Implémenter la méthode. Que faut-il changer (en plus de la surrelaxation) par rapport au programme précédent ?
2. Comparer les vitesses pour différentes valeurs de ε. Comparer aussi le résultat.
3. Quel est le problème primal correspondant ?

2.3 Bruit impulsionnel

On suppose maintenant qu'on a non plus un bruit Gaussien mais un bruit impulsionnel. Etant donné u l'image non bruitée, on obtient g par (par exemple, pour altérer ici 20% des pixels) :

```
>> mask= (rand(size(u))>.2);
>> g = u;
>> g(mask) = rand(size(u));
```

(on peut remplacer la dernière ligne par `g(mask)=0`).

Dans ce cas, la norme 2 doit être remplacée, dans le problème de reconstruction, par une norme 1 :

$$|g - u|_1 = \sum_{i,j} |g_{i,j} - u_{i,j}|.$$

1. Écrire l'algorithme "primal-dual" pour résoudre

$$\min_u \lambda |Du|_{2,1} + |u - g|_1.$$

2. Peut-on facilement accélérer ?

3 Reconstruction d'image

On s'intéresse maintenant au problème (2).

On pourra considérer pour A un sous-échantillonnage (problème du zooming) : si $u = (u_{i,j})$, $1 \le i \le M = rm$, $1 \le j \le N = rn$, $r \ge 2$, alors $(Au)_{i,j}$, $1 \le i \le m$, $1 \le j \le n$ est donné par

$$(Au)_{i,j} = \frac{1}{k^2} \sum_{k=1}^{r} \sum_{l=1}^{r} u_{r(i-1)+k, r(j-1)+l}.$$

Il faudra calculer alors son adjoint.

On pourra aussi considérer une convolution (qui "floute" l'image). On construit un (petit) noyau h, par exemple

```
>> h = [ 1 4 6 4 1 ];
>> h = h'*h;
>> h = h/sum(sum(h));
>> h = conv2(h,h);
```

(la dernière ligne est facultative et vise à construire un noyau plus grand). On peut observer l'effet en affichant `imagesc(conv2(hh,u));`.

La commande `filter2` est "adjointe" de `conv2`. On suppose que g est une version bruitée (*cf* partie précédente) de `filter2(h,u,'valid')` (`'valid'` ne retient que les pixels qui dont la convolution peut être calculée à partir de l'image u). La dérivée du terme de rappel quadratique

```
0.5*sum(sum( (filter2(h,u,'valid')-g).^2 ));
```

est *a priori* donnée par l'expression

```
conv2(h,filter2(h,u,'valid')-g,'full');
```

On suppose maintenant qu'on observe une donnée filtrée par u. Remarquons qu'ici la norme des opérateurs est ≤ 1 (par un résultat classique sur les convolutions).

1. Implémenter l'algorithme primal-dual (avec terme explicite) pour résoudre ce problème, et/ou bien :
2. Utiliser le "prox" de $|Du|_{1,2}$ calculé dans la section 2.1 pour implémenter plutôt "FISTA" pour résoudre ce problème. On pourra utiliser la version fortement convexe si on l'a implémentée.

4 Quelques indications supplémentaires

Exemple de fonction en matlab : faire un fichier "`nomdelafonction.m`" contenant

```
function [a b] = nomdelafonction(c,d,e)
a = c*d;
b = d*e;
```

La fonction `bsxfun` permet d'agir sur des lignes ou des colonnes. Exemple : projeter p sur la boule unité

```
no= max(1,hypot(p(1:MN),p(MN+1:end)));
p = bsxfun(@rdivide,reshape(p,MN,2),no);
p = p(:);
```

Le premier argument est l'opérateur qu'on veut appliquer (ici une division), le second un tableau $MN \times 2$ qu'on veut diviser terme à terme par le second qui est un tableau $MN \times 1$. Notons qu'ici pour ce calcul particulièrement simple,

```
p = p./[no;no];
```

fait la même chose (voir aussi la commande `repmat`). Exemple : projeter chaque composante (p^1, p^2) sur la boule de rayon $\lambda > 0$:

```
no= max(1,hypot(p(1:MN),p(MN+1:end))/lambda);
p = p./[no;no];
```

5 Appendice : les commandes équivalentes en Scilab

En scilab beaucoup de choses changent. Notamment, la syntaxe de `sparse` est différente et `reshape` devient `matrix`. Le mot clef `end` n'existe pas, ni la commande `hypot`. Voici une suggestion pour remplacer les commandes précédentes. Il faut d'abord charger un package de traitement d'images tel que "SIP" ou "SIVP".

```
%% scilab avec SIP/SIVP
u = double(imread('data/einsteinsmall.png'));
% éventuellement
u = sum(u,3)/3;

[M N] = size(u);
MN=M*N;
I=matrix([1:M*N],M,N);
east=[I(:,2:N), I(:,N)];
north=[I(2:M,:); I(M,:)];
D1 = sparse([I(:),east(:)],ones(MN,1),[MN,MN]) -speye(MN,MN);
D2 = sparse([I(:),north(:)],ones(MN,1),[MN,MN]) -speye(MN,MN);
D = [D1;D2];
%norm(D'*D) %% prend des heures ; retourne 7.9987664  pour [100 200]

gra = D*u(:);
no = matrix(sqrt((gra(1:MN).^2+gra(MN+1:2*MN).^2)),M,N);

%% exemple: projection sur la boule unité du gradient
no = max(1,sqrt((gra(1:MN).^2+gra(MN+1:2*MN).^2)));
gra = gra./[no;no];
```