Introduction à l'optimisation continue
Travaux pratiques, séance du 19 décembre 2023
Implémentation d'algorithmes et accélération

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.

En Python, on peut programmer D et D^* de deux façons. La plus élégante est de construire une matrice "sparse" `D` en utilisant `scipy.sparse`. On utilisera alors `D.transpose()` pour calculer D^*. Ou bien, on peut programmer efficacement, sans se tromper, $u \mapsto Du$ et $z \mapsto D^* z$. Notons que dans le premier cas tous les vecteurs (u, g) doivent être convertis en colonne pour pouvoir ensuite

effectuer des opérations de type `D.dot(u)`, alors que dans le second cas on peut avoir u de taille $M \times N$ et Du de taille $M \times N \times 2$.

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

```
>> import matplotlib.pyplot as plt
>> u = plt.imread('nom');
>> # attention: si png -> entre 0 et 1, si couleur -> calculer np.mean(u,axis=2)
>> [M,N] = u.shape
```

M est alors le nombre de lignes et N le nombre de colonnes.

Pour créer la matrice D, on doit d'abord former une matrice des indices de chaque pixel :

```
>> MN = M*N
>> I = rs(np.arange(MN),(M,N))
```

Ici `I[m,n]` contient l'indice $(m + n * M)$ du point (m, n) dans une image mise sous forme de colonne.

On récupère les indices des pixels immédiatement à droite et en dessous en construisant alors les tableaux :

```
>> east = rs(np.hstack((I[:,1:], I[:,-1:])),MN)
>> # numpy compte à partir de 0, -1 correspond ici en fait à
>> # la dernière colonne de la matrice I, qu'on répète
>> north = rs(np.vstack((I[1:,:], I[-1:,:])),MN)
>> I = rs(I,MN) # ici on remet I sous forme de colonne
>> # Dérivée horizontale
>> D1 = ssp.csr_matrix((np.ones(MN),(I, east)),shape=(MN,MN)) - ssp.eye(MN)
>> # Dérivée verticale
>> D2 = ssp.csr_matrix((np.ones(MN),(I,north)),shape=(MN,MN)) - ssp.eye(MN)
>> # gradient complet: on assemble verticalement les deux matrices
>> D = ssp.vstack((D1, D2))
```

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

On peut maintenant calculer le "gradient" d'une image par `gra=D.dot(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 = rs(np.hypot(gra[:MN],gra[MN:]),(M,N))
>> plt.imshow(no)
```

Si on n'est pas à l'aise avec la création de D, on peut aussi directement calculer le gradient d'une matrice $M \times N$ comme une matrice $M \times N \times 2$ en calculant les différences finies :

```
>> du = np.zeros((M,N,2))
>> du[1:end-1,:,0] = u[2:end,:] - u[1:end-1,:]
>> du[:,1:end-1,1] = u[:,2:end] - u[:,1:end-1]
```

(on peut remplacer `end` par `M` dans la première ligne, `N` dans la seconde, ou bien par *rien* ci-dessus).

Dans ce cas, il faut aussi implémenter D^* sans se tromper (voir le notebook jupyter pour une façon simple de ne pas se tromper...)

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

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

On résout donc

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

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

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 = np.sum((D_transpose().dot(p))**2)/2 - np.sum((D.transpose().dot(p))*g);
Energies.append(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= (np.random.rand(M,N))>.2);
>> g = u;
>> g[mask] = np.random.rand(M,N)[mask]
```

(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 $u = (u_{i,j})$, $1 \leq i \leq M = rm$, $1 \leq j \leq N = rn$, $r \geq 2$, alors $(Au)_{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq n$ 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

```
>> from scipy.signal import convolve2d as conv2d
>> h = np.array([ 1, 4, 6, 4, 1 ]);
>> h = h[:,None]*h[None,:] # fabrique un noyau 2D 5x5
>> h = h/sum(sum(h));
>> h = conv2d(h,h,mode='full'); # fabrique un filtre 9x9
```

(la dernière ligne est facultative et vise à construire un noyau plus grand). On peut observer l'effet en affichant `plt.imshow(conv2d(u,hh),cmap='gray')`.

Pour faire l'adjoint de la convolution, en Python, ce n'est pas si simple qu'en Matlab où la commande `filter2` est "adjointe" de `conv2`.

Il faut introdure le filtre retourné `hr = np.flip(h,axis=(0,1))`, puis calculer `conv2d(u,hr,mode='full')` pour obtenir l'adjoint de `conv2d(u,h,mode='valid')`.

On suppose que g est une version bruitée (*cf* partie précédente) de `conv2d(u,h,'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*(conv2d(u,h,mode='valid')-g)**2).sum()
```

est donc donnée par l'expression

```
conv2d(conv2d(u,h,mode='valid')-g,hr,mode='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). Au choix :

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.

Pour la première option, il faut écrire le problème :

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

où au préalable g est obtenu en convolant une image avec le filtre h et en ajoutant un bruit. On écrit alors un problème de point-selle :

$$\min_u \sup_{|p|_{2,\infty} \leq \lambda, v} p \cdot (Du) + v \cdot (h * u - g) - \frac{1}{2}|v|^2$$

où v a la taille d'une image floutée. Une itération de l'algorithme consiste alors à calculer :

$$u^{k+1} = u^k - \tau(D^* p^k + \check{h} * v^k)$$

puis, pour $\bar{u} = 2u^{k+1} - u^k$,

$$\begin{cases} p^{k+1} = \Pi_{\{|\cdot|_{2,\infty} \leq \lambda\}}(p^k + \sigma_p D\bar{u}) \\ \\ v^{k+1} = \dfrac{v^k + \sigma_v(h * \bar{u} - g)}{1 + \sigma_v} \end{cases}$$

où les paramètres sont choisis de sorte que

$$\sup \left\{ p \cdot (Du) + v \cdot (h * u) : \frac{|p|_{2,2}^2}{\sigma_p} + \frac{|v|_2^2}{\sigma_v} \leq 1, \quad \frac{|u|_2^2}{\tau} \leq 1 \right\} \leq 1,$$

soit

$$\tau(8\sigma_p + \sigma_v) \leq 1.$$