Université Bordeaux 1

Algorithme pour la Minimisation ℓ_1

Introduction

On reprend les notations de la feuille précédente.
Pour calculer le minimiseur de

$$\min_{x\in\mathbb{R}^N} \|x\|_1 \quad \text{sous la contrainte} \quad y = Ax \tag{1}$$

on effectue souvent une minimisation de la forme

$$\min_{x\in\mathbb{R}^N} \frac{1}{2}\|y-Ax\|_2^2 + \gamma\|x\|_1 \tag{2}$$

où on fait tendre γ vers 0. On peut montrer que si la solution du premier problème (1) est unique les solutions $x(\gamma)$ du second problème convergent vers cette solution quand γ tend vers 0. Ajoutons que dans la pratique, il y a toujours un bruit sur les observations et qu'il est souvent pertinent de résoudre (2) pour une petite valeur de γ que directement (1). On peut par ailleurs montrer que pour tout $\varepsilon > 0$ le problème

$$\min_{x\in\mathbb{R}^N} \|x\|_1 \quad \text{sous la contrainte} \quad \|y-Ax\|_2 \leqslant \varepsilon \tag{3}$$

peut s'écrire sous la forme (2) pour un γ adéquat. Le point délicat est que la bijection entre ε et γ n'est pas explicite.
Dans la suite on se concentrera sur la résolution de (2) en ayant en tête que la résolution d'un tel problème permet moyennant un jeu sur le paramètre γ de traiter les autres problèmes mentionnés ici.

1 Iterative Soft Thresholding

L'algorithme appelé Iterative Soft Thresholding (IST) a été explicité et démontré vers 2003. Il permet de résoudre par un algorithme itératif le problème (2).
L'algorithme peut être décrit de la manière suivante :

$$x_{n+\frac{1}{2}} = x_n + A^t(y - Ax_n)$$
$$x_{n+1} = ST(x_{n+\frac{1}{2}}, \frac{\gamma}{2})$$

où $ST(x,t)$ désigne le seuillage doux. Rappelons que

$$ST(x,t) = \begin{cases} x-t & \text{si } x > t \\ x+t & \text{si } x < t \\ 0 & \text{sinon.} \end{cases}$$

Le seuillage doux est ici effectué terme à terme sur chacun des composantes de x_n.
Il a été démontré que si l'opérateur A est contractant, quelque soit le point de départ x_0, la suite x_n converge vers un minimiseur de (2).
Pour mettre en place l'algorithme il faut donc savoir appliquer l'opérateur A et l'opérateur A^t qui l'opérateur transposé de A, comme pour le Matching Pursuit.

1. Ecrire une fonction Matlab

`function z=DCTDirac(x)`

qui prend en entrée un vecteur de taille N pair et qui renvoie un vecteur de taille $N/2$ dont l'image est la même que celle obtenue par la matrice formée de la concaténation d'une base de DCT et de la matrice identité. (C'est à dire la matrice utilisée pour le Matching Pursuit dans un TD précédent). On n'utilisera pas la matrice mais uniquement la dct de matlab.

2. Ecrire une fonction Matlab

   `function z=DCTDiracTransp(x)`

   Qui calcule la transposée de l'application précédente.

3. Ecrire une fonction

```
function x=DIST(y,OpDirect,OpTansp,gamma,N,x0)
A=str2func(OpDirect);
At=str2func(OpTranp);
if nargin<6
x0=At(y);
end
```

   qui renvoie le vecteur calculé par l'IST après N itérations, intialisé à $x0$ et qui prend en paramètres deux chaines de caractères donnant les opérateurs direct et transposé. On pourra faire un affichage toutes les 100 itérations.

4. Tester ce code comme vous l'avez fait pour le MP en utilisant la fonction *Genevect*. Vérifiez bien que vous avez fait suffisamment d'itérations.
5. Comparer pour $N = 256$ la parcimonie des vecteurs qu'on peut reconstruire à l'aide de cet algorithme et par MP.
   Application à l'inpainting .

6. Ecrire une fonction matlab

   `function D=DCTLocale(I)`

   qui prend en entrée une matrice carrée I et qui renvoie une matrice de même taille contenant les coefficients de DCT calculés sur des bloc 8×8 (utiliser la commande *dct2*).
   On pourra supposer que les dimensions de la matrice sont des multiples de 8.
7. Faites plusieurs tests sur des images en niveaux de gris.
8. Justifier que cette transformation est orthogonale.
9. Ecrire la fonction réciproque

   `function I=InvDCTLocale(D)`

   qui reconstruit l'image à partir de la DCT locale et la tester sur plusieurs images.
10. Quelle est l'application transposée de l'application *DCTlocale* ?
11. Soit I une image. On peut définir un opérateur de masquage M consistant à multiplier par 0 certains pixels de l'image. Quelle est l'application transposée au masquage ?
12. Simuler un masque aléatoire M sur une image en conservant aléatoirement environ 70 pourcents des pixels.
13. Utiliser le fait que la plupart des images naturelles sont parcimonieuses en *DCT locale* pour reconstruire l'image en utilisant l'*Iterative Soft Thresholding*. Il faudra pour cela construire les applications directes et transposées associées.