Optimisation et programmation dynamique

Ch. Dossal

2008

1 Introduction

Si on sait qu'un signal est uniformément régulier ou s'il est stationnaire , on utilisera facilement une transformée de Fourier pour le traiter. S'il est régulier par morceaux on sait qu'une transformée en ondelettes sera plus efficace. Si une image est géométriquement régulière on peut espérer que les curvelets fourniront un outils intéressant d'analyse. Dans nombre de situations les a priori que nous avons sur le signal ne permettent pas de dégager une représentation unique orthogonale ou non qui assure une représentation efficace du signal. En revanche il existe parfois des dictionnaires de bases ou d'une manière générales des représentations plus adaptatives que ne l'est la représentation dans une unique base, qui permettent de traiter le signal efficacement. C'est le cas des paquets d'ondelettes. Les paquets d'ondelettes sont des familles de BO construites à partir ondelettes. Pour une base d'ondelettes sur N points il existe de l'ordre de 2^N bases de paquets d'ondelettes. Pourtant, il existe un algorithme rapide qui permet de déterminer *un paquet bien adapté à un signal donné*. Plus précisément il existe un algorithme rapide déterminant la base de paquet d'ondelettes minimisant une certaine entropie. L'existence d'un algorithme rapide de minimisation globale de l'entropie malgré le nombre colossal de bases est rendu possible par la structure bien particulière des bases de paquets d'ondelettes. Ces bases sont chacune définie par un arbre binaire où chaque feuille correspond à un ensemble de fonctions. Ainsi chaque paquets d'ondelettes est associé à une feuille et appartient ainsi à un grand nombre de bases. Ce nombre limité de paquets d'ondelettes associé à la structure arborescente permet un algorithme de minimisation global et rapide.
L'algorithme consiste à calculer toutes les minimisations locales possibles dans un premier temps et à ensuite effectuer une minimisation globale en utilisant la structure arborescente de la famille de bases. Plus précisément : **Pour minimiser une fonctionnelle additive sur toutes les subdivisions dyadiques d'un segment on procède en deux étapes:**

1. **Initialisation :** on calcule la fonctionnelle sur chacun des segments dyadiques.

2. **Otpimisation** : pour chaque segment, en commençant par les plus petits et en terminant par le plus grand, on compare le coût de la meilleure approximation globale et le coût de la juxtaposition des deux meilleures approximations locales.

Cette technique de minimisation se transpose à beaucoup d'autres situations, en une et deux dimensions. Nous allons ici en étudier 2 en particulier.
La première consiste à déterminer la meilleure approximation dyadique et constante par morceaux d'un vecteur. Cette méthode qui fonctionne quelque soit la métrique choisie s'étend naturellement en 2D.
La seconde consiste à déterminer la meilleure approximation constante par morceaux d'un vecteur. Cette fois ci on enlève la contrainte d'être constant sur des intervalles dyadiques.

Cette méthode de minimisation est à la base de la décomposition en **bandelettes**, ces bases géométriques construites sur les ondelettes et qui s'adaptent à la fonction considérée. Cette même méthode est aussi à la base des edglets (ou weglets) développées par D. Donoho qui fournissent également des représentations adaptatives des images géométriques. Cette méthode utilise ce qu'on appelle la programmation dynamique qui permet également de s'attaquer à des problèmes d'optimisation plus généraux, comme la recherche de stratégie optimale (en finance par exemple).

2 Meilleure approximation dyadique constante par morceaux

Le but de cette section est de rédiger un programme qui prend en entrée un vecteur S et renvoie le meilleure approximation de ce vecteur, constante sur des segments dyadique avec N discontinuités.
Plus précisément, on minimisera la fonctionelle suivante :

$$\min_X d(X,S) + C \times (N(X)) \tag{1}$$

où d est une fonctionelle additive, une norme ℓ_1 ou un carré de norme ℓ_2, où $N(X)$ est le nombre de discontinuités de X. Tout minimiseur X_0 d'une telle fonctionnelle est la meilleure approximation de S avec moins $N(X_0)$ discontinuités. En faisant varier C qui est un paramètre de coût de chaque discontinuité, on obtient des approximations plus ou moins précises. Si C est choisi grand, chaque discontinuité coûte plus cher et donc X_0 admet peu de discontinuités. Si C est petit au contraire, X_0 sera plus proche de S.

1. Montrer que si X_0 est un minimiseur de (1) alors X_0 est le vecteur le plus proche de S parmi tous les vecteurs ayant moins de $N(X_0)$ discontinuités.

2. Si on veut faire une minimisation au sens ℓ_2, pourquoi doit-on utiliser carré de la norme ℓ_2 dans la fonctionnelle (1) et pas la norme ℓ_2 elle même.

3. Ecrire une fonction *Approximation*

   ```
   function [a,d]=Approximation(S1,par)
   ```

   qui calcule la meilleure approximation constante a de $S1$ pour la distance considérée et donne la distance d de $S1$ à cette approximation constante. La distance étant définie par le paramètre *par*.

   (a) Pour $par = 0$, d désigne la distance de Hamming, c'est à dire le nombre de valeurs où les vecteurs sont différents.

   (b) Pour $par = 1$, d désigne la norme ℓ_1.

   (c) Pour $par = 2$, d désigne **le carré** de la norme ℓ_2.

4. Tester la fonction précédente avec différents signaux simples et les différentes distances en affichant le signal et l'approximation.

5. Ecrire un programme qui calcule le minimiseur de (1)

   ```
   Srec=Optidyadique(S,C,par)
   ```

qui prend en entrée un vecteur S de taille N dyadique (i.e. $N = 2^d$), un réel positif C et un paramètre spécifiant la distance utilisée et qui calcule le vecteur dyadique (i.e. constant sur des intervalles dyadiques) minimiseur de la fonctionelle (1).
Pour cela on pourra utiliser

- Un Vecteur *Approx* de taille $2N-1$ qui constiendra toutes les meilleures approximations par des constantes. Chaque composante correspondant à un des $2N-1$ segments dyadiques.
- Un Vecteur *Cout* de taille $2N-1$ qui contiendra la valeur de la fonctionnelle (1) sur chacun de ces mêmes segments. Il sera initialisé grace à la fonction Approx et optimisé dans un second temps.
- Un vecteur *Segmentation* de taille $2N-1$ (en fait N) suffirait, à valeur booléenne. Chaque composante du vecteur est associé à un intervalle dyadique et vaut 1 quand il est optimal de le segmenter et 0 sinon.

Dans le programme on séparera bien les trois phases

(a) Initialisation : On calcule *Approx* et on initialise *Cout*.

(b) Optimisation : On optimise *Cout* et on calcule *Segmentation*. Cette partie peut se faire de manière itérative en commençant par les petits segments ou de manière récursive en lançant directement l'optimisation sur tout le vecteur en utilisant une fonction qui s'appelle elle même sur les deux segments fils. Cette deuxième approche est plus risquée.

(c) Reconstruction : A partir de *Segmentation* et *Approx* on reconstruit le minimiseur de (1).

Penser à tester les phases une par une sur des exemples simples avec des vecteurs de taille 4 ou 8.

6. Tester le code sur différents vecteurs, pour différentes distances et pour différentes valeurs de C. On pourra utiliser le vecteur *Blocks* construit par la fonction*MakeSignal*.

7. Est-il toujours possible en ajustant finement la valeur de C d'imposer un nombre choisi de discontinuités à l'approximation ?

8. Tester l'algorithme en présence de bruit.

9. Quel peut être l'intérêt de la distance de Hamming dans ce contexte ?

3 Meilleure approximation constante par morceaux

On souhaite maintenant s'affranchir de la segmentation dyadique pour réaliser l'optimisation sur toutes les segmentations.

10. Ecrire un programme *Opti*

```
Srec=Opti(S,C,par)
```

qui calcule le minimiseur de (1) sur toutes les segmentation possible. On pourra utiliser la même fonction *Approximation* et la même structure de programme avec 3 phases. La différence fondamentale réside ici dans le fait qu'on s'autorise non plus une segmentation par segment mais $n-1$ où n est la taille du segment. On pourra ainsi utiliser des matrices à la place de vecteurs pour *Approx, Cout et Segmentation* ou chaque ligne correspond à une taille de segment. On peut aussi choisir des structures de liste de vecteurs. La matrice *Segmentation* n'est ici plus booléenne car la segmentation peut se faire n'importe où mais prend maintenant des valeurs entières.
On optimisera la fonction de coût avec un algorithme itératif en commençant par les petits segments et en considérant les segments par taille croissante.