Université Aix-Marseille 2018-2019
Master Mathématiques appliquées et statistiques 2ème année

Représentation parcimonieuse des signaux

TP : Parcimonie et algorithme de Matching-Pursuit.

L'objectif de ce TP est la programmation de l'algorithme de Matching-Pursuit et la vérification de la pertinence du théorème 2 de la fiche de cours sur le Matching-Pursuit. Nous prendrons pour tous nos tests un dictionnaire particulier qui est composé de la réunion de la base canonique et de Fourier.

Notations

— On note $\langle .,. \rangle$ le produit scalaire usuel sur $\mathbb{C}^N$ tel que $\langle x, y \rangle = \sum_{n=0}^{N-1} x_n \overline{y_n}$ pour $x = (x_i)_{i=0,\dots N-1} \in \mathbb{C}^N$ et $y = (y_i)_{i=0,\dots N-1} \in \mathbb{C}^N$. On note $|x|_2^2 = \sum_{n=0}^{N-1} |x_n|^2$ pour $x \in \mathbb{C}^N$.

— On note δ^ℓ pour $\ell \in \{0, \dots N-1\}$ le ℓ-ième vecteur de la base canonique de $\mathbb{C}^N$. Il s'agit du vecteur de $\mathbb{C}^N$ tel que pour $n \in \{0, \dots N-1\}$ on a $\delta_n^\ell = 1$ si $n = \ell$ et 0 sinon. La collection de vecteurs $\{\delta^\ell, \ell \in \{0, \dots, N-1\}\}$ constitue la base canonique de $\mathbb{C}^N$.

— la famille de vecteurs $\widetilde{\mathcal{E}} = \{\frac{1}{\sqrt{N}} e^\ell \in \mathbb{C}^N, \ell \in \mathbb{Z}\}$ tels que pour $\ell \in \mathbb{Z}$ et pour $n \in \{0, \dots N-1\}$ la n-ième coordonnée du vecteur e^ℓ s'écrive

$$e_n^\ell = e^{\frac{2i\pi\ell n}{N}}$$

est appelée base de Fourier orthonormalisée discrète.

1 Algorithme de Matching-Pursuit

1.1 Rappel de l'algorithme

L'algorithme que nous souhaitons étudier est en effet le suivant.

Algorithme 1 : Matching Pursuit.

Data : dictionnaire $D = [\phi^0, \dots, \phi^{K-1}] \in \mathbb{C}^{N,K}$, $x \in \mathbb{R}^N$, $\varepsilon > 0$, $M_{max} \in \mathbb{N}$
initialisation :
$r \leftarrow x$ [Initialisation du résidu];
$\alpha \leftarrow 0_K$ [Initialisation de α];
$m \leftarrow 0_K$ [Initialisation des itérations];
while $m \leq M_{max}$ *et* $|r|_2 > \varepsilon$ **do**
  $m \leftarrow m + 1$;
  $\forall m, c_m \leftarrow \langle r, \phi^m \rangle$;
  $k_m \leftarrow \arg \max_{m \leq K-1} |\langle r, \phi^m \rangle|$;
  $\alpha(k_m) \leftarrow \alpha(k_m) + c_{k_m}$;
  $r \leftarrow r - c_{k_m} \phi^{k_m}$
Result : $\alpha \in \mathbb{C}^K$, M nombre d'itérations.

1.2 Travaux pratiques

Exercice 1 (*Programmation de l'algorithme*)

1. *Programmer l'algorithme de Matching Pursuit.*

   On cherchera en particulier à en programmer une version qui permet de vérifier qu'à chaque itération la norme du résidu est bien décroissante. Cela nous servira dans la phase de test du programme pour vérifier que la programmation de l'algorithme est correcte. Dans un deuxième temps on pourra s'en passer et garder une version qui ne renvoie pas la norme du résidu.

2. *Le tester en utilisant le dictionnaire $\mathcal{D}$ dont la matrice représentative dans la base canonique est produit par le code* `creaDictBaseCanFourier.py` *fourni sur Ametice et appliquant l'algorithme sur le vecteur $x = (x_0, \ldots, x_{N-1})$ tel que pour tout $n = 0, \ldots N-1$ avec $N = 1024$ $x_n = 4\delta_n^{256} + 0.25\delta_n^{700} + \frac{16}{\sqrt{N}} \cos\left(2\pi * 25 * \frac{n}{N}\right)$.*

   L'algorithme fournit-il la décomposition parcimonieuse recherchée ?

2 Limites du théorème de recouvrement des décompositions parcimonieuses

Notre objectif est ici de chercher à voir si le théorème 2 de la fiche 4 est un énoncé « optimal », au sens où on peut se demander par exemple si lorsque $card(\mathcal{L})\mu(\mathcal{D}) > \frac{1}{2}$ alors l'algorithme du Matching-Pursuit ne converge pas vers la décomposition parcimonieuse voulue.

2.1 Approfondissement théorique

Nous rappelons l'énoncé du théorème donné dans la fiche 4.

Théorème 1

Soit $\mathcal{D} = \{\phi^0, \ldots, \phi^{K-1}\}$ un dictionnaire de $\mathbb{C}^N$ et $x \in \mathbb{C}^N$ tel que

$$x = \sum_{\ell \in \mathcal{L}} \alpha_\ell \phi^\ell \tag{1}$$

où $\mathcal{L}$ est un sous-ensemble de $\{0, \ldots, K-1\}$ tel que $\{\phi^\ell, \ell \in \mathcal{L}\}$ forme un système libre.

Si $card(\mathcal{L})\mu(\mathcal{D}) < \frac{1}{2}$ alors l'algorithme de Matching-Pursuit sélectionne à chaque itération un vecteur ϕ^ℓ avec $\ell \in \mathcal{L}$ et converge vers la décomposition (1).

Une question que nous pouvons nous poser c'est de savoir si dans le théorème 1 les hypothèses sur le fait qu'un système de $card(\mathcal{L})$ vecteurs extraits du dictionnaire est libre sont restrictives ou non.

Nous avons la proposition suivante

Proposition 1

Soit $\mathcal{D} = \{\phi^0, \ldots, \phi^{K-1}\}$ un dictionnaire de $\mathbb{C}^N$ de cohérence $\mu(D)$ et $\mathcal{L}$ un sous-ensemble de $\{0, \ldots, K-1\}$.

Si $card(\mathcal{L})\mu(\mathcal{D}) < 1$ le système $\{\phi^j, j \in \mathcal{L}\}$ est un système libre.

Début de la démonstration de la proposition 1

La démonstration repose sur l'étude de la matrice de Gram G du système $\{\phi^j, j \in \mathcal{L}\}$, c'est à dire de la matrice dont les coefficients sont $\langle \phi^i, \phi^j \rangle$ pour $i \in \mathcal{L}$ et $j \in \mathcal{L}$. En effet on sait que G est inversible si et seulement si le système des $\{\phi^j, j \in \mathcal{L}\}$ est un système libre. Donc montrer que $\{\phi^j, j \in \mathcal{L}\}$ est un système libre revient à montrer que G est inversible. Montrer que G est inversible revient à montrer qu'elle n'a pas de valeurs propres nulles.

Exercice 2

En vous inspirant du théorème de Gershgorin (qui permet de borner les valeurs propres d'une matrice carrée) ou même de sa démonstration, compléter et finir la démonstration de la proposition 1.

2.2 Travaux pratiques

On pourra penser pour cette partie à utiliser le module `numpy.random`.

Exercice 3

1. *Écrire une fonction qui permet de générer aléatoirement un vecteur α_s de taille K ayant exactement s coordonnées non nulles ($s \leq K$), chacune de ces coordonnées étant la réalisation d'une variable aléatoire suivant une loi normale.*
2. *Soit $\mathcal{D} = \{\phi^0, \ldots, \phi^{K-1}\}$ le dictionnaire constitué de la réunion des vecteurs de la base canonique et de la base de Fourier orthonormalisée.*

   Calculer la décomposition sur les vecteurs du dictionnaire donnée par l'algorithme de Matching-Pursuit du vecteur x calculé à l'aide de $x = D\alpha_s$ où α_s est calculé par la fonction de la question précédente pour différentes valeurs de s. Y-a-t-il des cas où l'algorithme retrouve bien α_s alors que l'hypothèse sur le produit $card(\mathcal{L})\mu(\mathcal{D}) < \frac{1}{2}$ du théorème 1 n'est pas vérifiée ?

2.3 Au delà du théorème 1.

Lorsque $\mathcal{D} = \{\phi^0, \ldots, \phi^{K-1}\}$ est un dictionnaire constitué de la réunion de deux bases orthonormées, on peut obtenir un résultat légèrement plus favorable que celui du théorème 1 en écrivant la partition $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$ telle que $\{\phi^\ell, \ell \in \mathcal{L}_1\}$ est un sous-ensemble de la première base alors que $\{\phi^\ell, \ell \in \mathcal{L}_2\}$ est un sous-ensemble de la deuxième.

On peut ainsi montrer sans trop d'efforts supplémentaires le théorème suivant

Théorème 2

Soit $\mathcal{D} = \{\phi^0, \ldots, \phi^{K-1}\}$ un dictionnaire de $\mathbb{C}^N$ qui est la réunion d'une base orthonormée $\mathcal{B}_1$ et d'une base orthonormée $\mathcal{B}_2$. Soit $x \in \mathbb{C}^N$ tel que

$$x = \sum_{\ell \in \mathcal{L}} \alpha_\ell \phi^\ell \tag{2}$$

où $\mathcal{L}$ est un sous-ensemble de $\{0, \ldots, K-1\}$ tel que $\{\phi^\ell, \ell \in \mathcal{L}\}$ forme un système libre.

On pose $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$ avec $\mathcal{L}_1$ et $\mathcal{L}_2$ tels que $\{\phi^\ell, \ell \in \mathcal{L}_1\} \subset \mathcal{B}_1$ alors que $\{\phi^\ell, \ell \in \mathcal{L}_2\} \subset \mathcal{B}_2$.

Si $\max(card(\mathcal{L}_1), card(\mathcal{L}_2)) \times \mu(\mathcal{D}) < \frac{1}{2}$ alors l'algorithme de Matching-Pursuit sélectionne à chaque itération un vecteur ϕ^ℓ avec $\ell \in \mathcal{L}$ et converge vers la décomposition (1).

On peut de même adapter l'énoncé de la proposition 1 sous les hypothèses que $\mathcal{D}$ est la réunion de deux bases orthonormées.

3 Remarques pour la programmation

Ici on fait quelques remarques pour faciliter la programmation et constater que certaines opérations qu'on cherche à faire dans l'algorithme peuvent se faire à l'aide d'opérations d'algèbre linéaire.

Soit x un vecteur colonne de $\mathbb{C}^N$ et D la matrice dont les vecteurs colonnes sont les vecteurs ϕ^k pour $k = 0, \dots K-1$ du dictionnaire représentés dans la base canonique. Chaque vecteur colonne de D s'écrit donc $\phi^k = \begin{pmatrix} \phi_0^k \\ \phi_1^k \\ \vdots \\ \phi_{N-1}^k \end{pmatrix}$

On suppose ici que $x = \sum_{k=0}^{K-1} \alpha_k \phi^k$ avec $\alpha_i \in \mathbb{C}$.

3.1 Synthèse d'un vecteur x à partir des coefficients de décomposition de ce vecteur dans le dictionnaire.

Supposons que α est le vecteur colonne $\alpha = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{K-1} \end{pmatrix}$

tel que l'on a $x = \sum_{k=0}^{K-1} \alpha_k \phi^k$. Alors cela correspond à l'opération matricielle

$$x = D \times \alpha$$

ou encore

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{pmatrix} = \begin{pmatrix} \phi_0^0 & \phi_0^1 & \dots & \phi_0^{K-1} \\ \phi_1^0 & \phi_1^1 & \dots & \phi_1^{K-1} \\ \phi_2^0 & \phi_2^1 & \dots & \phi_2^{K-1} \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{N-1}^0 & \phi_{N-1}^1 & \dots & \phi_{N-1}^{K-1} \end{pmatrix} \times \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{K-1} \end{pmatrix}$$

3.2 Calcul des produits scalaires

Nous aurons besoin de calculer des produits scalaires $c = \begin{pmatrix} \langle x, \phi^0 \rangle \\ \langle x, \phi^1 \rangle \\ \langle x, \phi^2 \rangle \\ \vdots \\ \langle x, \phi^{N-1} \rangle \end{pmatrix}$.

Cela se fait à l'aide de l'opération matricielle $c = D^* \times x$ où D^* est la matrice qui a pour ligne d'indice k les conjugués des coordonnées de ϕ^k en effet

$$\begin{pmatrix} \langle x, \phi^0 \rangle \\ \langle x, \phi^1 \rangle \\ \langle x, \phi^2 \rangle \\ \vdots \\ \langle x, \phi^{N-1} \rangle \end{pmatrix} == \begin{pmatrix} \sum_{n=0}^{N-1} x_n \overline{\phi_n^0} \\ \sum_{n=0}^{N-1} x_n \overline{\phi_n^1} \\ \sum_{n=0}^{N-1} x_n \overline{\phi_n^2} \\ \vdots \\ \sum_{n=0}^{N-1} x_n \overline{\phi_n^{K-1}} \end{pmatrix} = \begin{pmatrix} \overline{\phi_0^0} & \overline{\phi_1^0} & \dots & \overline{\phi_{N-1}^0} \\ \overline{\phi_0^1} & \overline{\phi_1^1} & \dots & \overline{\phi_{N-1}^1} \\ \overline{\phi_0^2} & \overline{\phi_1^2} & \dots & \overline{\phi_{N-1}^2} \\ \vdots & \vdots & \vdots & \vdots \\ \overline{\phi_0^{K-1}} & \overline{\phi_1^{K-1}} & \dots & \overline{\phi_{N-1}^{K-1}} \end{pmatrix} \times \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{pmatrix}$$

3.3 Programmation avec Python

L'autrice de ce sujet ayant passé du temps sur la programmation de l'algorithme elle vous donne quelques conseils issus de son expérience concrète...

— Vecteurs : prendre le format `np.array` sans chercher à en faire des vecteurs colonnes c'est à dire **sans** utiliser la commande `reshape(x,(N,1))` par exemple. Ainsi on définit le vecteur $x = a_0\delta^{k_0} + a_1\delta^{k_1}$ de la manière suivante

```
import numpy as np

N=1024
x=np.zeros(N)

a0=4
k0=2*128
x[k0]=a0

a1=0.25
k1=2*350
x[k1]=a1
```

— Par contre on utilisera comme commande pour le produit matriciel `@`, ce qui donne par exemple pour la première itération du programme

```
import numpy as np

r=np.copy(x)
D1=np.conj(D.T)

c=D1@r
```

— on pourra utiliser la bibliothèque `numpy.linalg` qui contient une fonction norme afin de calculer la norme euclidienne du résidu r.

```
import numpy as np
import numpy.linalg as npl

nr=npl.norm(r)
```

— On peut interrompre une boucle si une certaine condition est remplie à l'aide la commande `break`, ce qui donne par exemple

```
import numpy as np
eps=10**(-6)

def MPursuit(D,x,eps,M):
        n=0...
        while n<M
                ..
                if nr<eps
                        break
        return alpha,n,NR
```

— Pour ajouter des éléments à un vecteur numpy on peut utiliser la fonction `np.append`

4 Compte-rendu du TP

Le compte-rendu est à rendre individuellement.

Le compte-rendu du TP consiste en une archive contenant
- un fichier .pdf écrit en Latex ou un fichier Note_book contenant le texte (commentaires, démonstrations mathématiques) et les figures du compte-rendu
- les fichiers `.py` qui ont servi pendant la programmation, qui doivent être commentés : on doit dès les premières lignes et tout au cours du programme savoir ce qu'ils font et ce qu'ils calculent ou illustrent.

Si le texte est rendu en .pdf il est indispensable que parmi les fichiers `.py` on trouve pour chaque partie pratique un fichier dont le titre commence par `demo` et indexé par le numéro de l'exercice qu'il suffit d'executer pour illustrer informatiquement ce qui est indiqué dans le compte-rendu.

Dans tous les cas dans le compte-rendu on doit trouver

1. Des commentaires et autant que possible des explications sur les simulations présentées et les phénomènes constatés. On pourra choisir d'approfondir numériquement certains points particuliers qui vous paraissent pertinents et intéressants et détailler les conclusions qu'on peut tirer de ces expériences numériques.
2. Une étude mathématique partielle ou exhaustive. Elle peut s'appuyer par exemple sur la démonstration des propositions indiquées dans le sujet, ou sur tout autre développement dans le cadre du sujet.
3. **Pour aller plus loin :** voici quelques pistes (non exhaustives)
   - On peut analyser la vitesse de convergence de l'algorithme.
   - on peut examiner les capacités de recouvrement de la décomposition parcimonieuse par le Matching-Pursuit d'un signal lorsqu'il a été bruité.
   - on peut s'intéresser au cas d'autres dictionnaires que celui de la réunion de la base canonique et de la base de Fourier orthonormaliée.
   - On peut étudier l'algorithme de Matching-Pursuit orthogonal présenté à la fin de la fiche de cours 4.

Grille de notation approximative

Ci-dessous quelques critères qui guideront la notation : quatre critères vont être pris en compte
- programmation : les codes ne doivent pas être buggés et comporter des commentaires qui expliquent et détaillent ce qu'ils font.
- commentaires des parties simulations : ils doivent aider à comprendre ce qui est simulé, et le point principal est leur clarté et leur précision. On attend aussi de chercher à comprendre ce que l'on fait et donc d'expliquer au lecteur le plus possible les raisons pour lesquelles on choisit de calculer telle ou telle quantité.
- explications théoriques : un ou plusieurs développements mathématiques qui vous intéressent ou vous semblent pertinents doivent être présentés, expliqués et articulés avec la partie simulation. Il est possible de choisir d'autres points que ceux qui sont suggérés par l'énoncé du TP et cela est a priori encouragé à l'avertissement près que le plagiat sera durement sanctionné.
- forme du document : le document doit être clair et correctement rédigé. Il doit comporter une introduction et un plan et être rédigé en français clair et autant que possible correct.

Pour vous donner une idée voilà à peu près ce qui vous attend selon le degré d'investissement dans le travail.
- Note entre 0 et 5/20 : les codes ne tournent pas en général. En particulier les fichiers `demo....py` ne sont pas présents ou sont buggés. La rédaction est très succinte voir inexistante. La forme du document laisse à désirer et le français est incorrect ou parfois même incompréhensible.
- Note entre 5 et 8/20 : certains codes tournent, mais pas tous. En particulier un ou deux fichiers `demo....py` ne tournent pas. Les commentaires sont généraux et peu précis et

reprennent le sujet du TP avec presque aucun apport personnel ni détails sur ce qui est fait, ou alors les commentaires sont juste descriptifs sans aucun effort d'analyse.
— Note entre 8/20 et 10/20 : les codes tournent mais les commentaires restent peu précis ou alors purement descriptifs, et il y a peu de réflexion personnelle dans le compte-rendu. On ne comprend pas bien pourquoi tel ou tel calcul est produit.
— Note entre 10/20 et 15/20 : les codes tournent. Une partie jusqu'à toutes les parties mathématiques sont développées et détaillées. Les simulations sont expliquées clairement. Il y a des efforts de rédaction pour articuler les commentaires des simulations et de la partie mathématique.
— Note entre 15/20 et 20/20 : les codes tournent, les commentaires de la partie simulation et de la partie mathématique s'articulent et expliquent ce qui est fait. On va même dans certains cas jusqu'à aller plus loin et proposer d'explorer une piste qui n'est pas présente stricto senso dans le sujet.

Annexe

On met en annexe le programme pour obtenir le dictionnaire Base canonique-Fourier orthonormalisé.

```
import numpy as np
import numpy.fft as npfft
import numpy.linalg as npl

def creatematrixCaFo(N):
    F=np.zeros((N,N),dtype='complex')
    for i in np.arange(0,N):
        u=np.zeros(N,dtype='complex')
        u[i]=np.sqrt(N)
        e=npfft.ifft(u)
        e=e/npl.norm(e)
        F[:,i]=e
    D=np.concatenate((np.eye(N),F),axis=1)
    return D
```