Comprendre le code, résultats savoir manipuler pour pouvoir comprendre

Identification du signal

Main = RV\_analysis

Etude du HD189733

Récupération des données : On dispose de 63 mesures dont 55 sur un temps continu, pour chacune on obtient 39 estimations différents de RV correspondants aux différents ordres. Les ordres 30 et 38 sont « fausses données » dont on les enlève. On conserve seulement 55 temps (on a enlevé les instants 3 et 17 car l’ordre 0 était faux).

* On normalise la matrice en la centrant
* On réalise une PCA

L’idée est de construire un « proxy » (en gros une première approximation), soit en considérant pour chaque temps le RV obtenu en moyennant les ordres (la + naïve) soit en projetant sur l’axe d’inertie le plus fort de la PCA.

On obtient un meilleur SNR pour AVG que PCA1 mais relativement équivalent

Comme pour chaque instant on a 37 variables explicatives, réaliser une ICA sur les 37 variables de la PCA est impossible et n’aurait d’ailleurs pas beaucoup d’intérêts car certains axes n’expliquent rien. Pour réduire le problème on cherche donc une solution avec un nombre d’axes limités par exemple les 5 meilleurs de la PCA qui soit la combinaison linéaire de ces 5 axes qui maximise le SNR avec le signal théorique.

On note cette CL HopeLim

On obtient des HopeLim bien inférieures en réduisant la dimension mais nettement meilleurs qu’une PCA. On observe que le SNR de la PCA ne varie pas trop en réalisant une PCA après réduction de dimension, c’est logique car la quasi-totalité est expliquée par les 5 premières composantes. (En réalité il est même meilleur si l’on réduit à 1 dimension, on observe aussi que le SNR de la PCA oscille jusqu’à chuter complétement en dim 19 pour remonter progressivement à la valeur obtenue précédemment lorsqu’on a pas de réduction).

Fichiers :

Heuristics.py 🡺 Fichier qui contient les différentes méthodes d’ICA donc la méthode implémentée par Pablo V\_global\_optim

Ces fonctions renvoient à partir de X, les matrices A et S telles que X = A\*S

Benchmark.py 🡺 Code implémenté pour comparer les différentes méthodes ICA (donc pas très utile pour nous)

Utils.py 🡺 Fichier contenant toutes les fonctions de calculs et process et d’affichage, notamment les fonctions whiten qui crée X\_w à partir de X, standardize, SNR, mutual\_information ou encore plot\_2D\_distrib

Penal.py 🡺

Core.py 🡺 Fichier qui contient les classes CustomICA et PostProcess

Fonctions : (T est l’opérateur transposé)

SNR (package utils) 🡺 prend en argument le vrai signal (1er arg) et l’estimation et renvoie le rapport de la variance du premier signal avec la variance de la différence du signal et de l’estimation (ie le bruit) (Objectif : avoir un bon SNR car le signal du dénominateur correspond au bruit)

Whiten(package utils) 🡺 prend en argument la matrice des RV, X (en assumant qu’elle est centrée ie mean(X)=0 et renvoie X\_w, Whiten\_mat,Whiten\_mat\_inv et eigs\_RY\_2 tel

que

X\_w \*T(X\_w) = Id, Whiten\_mat\*X = X\_w et eigs\_RY\_2 liste des vp matrice de cov X\*T(X)

Rappel : X\_w = (D^(-1/2)\*T(E))\*X (PCA 🡺 X\*T(X) = E\*D\*T(E))

Standardize (package utils) 🡺 prend un vecteur u et renvoie u/norm(u) où norm est la norme euclidienne

CustomICA (package core)🡺