Obtentions des données présentées au BeBeC Mi-mars 2018

Ce rapport présente les détails techniques ayant permis l'obtention des figures (avec Matlab) pour l'article du BeBeC 2018 sur le débruitage (notament sur le réglage des hyperparamètres de chaque méthode).

I. Structure générale

L'étude des paramètre SNR, nombre de sources et nombre de snapshots se fait respectivement dans les fichiers `influence_SNR.m`, `influence_rang.m` et `influence_Mw.m`. Dans ces fichiers, les CSM signal et bruit sont générées avec la fonction `generate_Spp_signal.m` et les algorithmes de débruitage sont ensuite appliqué à la CSM bruitée.

Les codes EM et MCMC de Jérôme sont conservés en local uniquement dans `/Debruitage_Jerome_VersionMars2018`

II. Génération des CSM

Le dossier `Generate_Spectra` contient la fonction de génération des CSM `generate_Spp_signal.m`, ainsi que les coordonnées de l'antenne utilisée (dans `spiral_array_coord.mat`, provenant du benchmark de P. Sijstma) et la graine pour générer tout le temps le même spectre `s.mat`.
Remarque : il se peut que les données avec bruits hétérogène aient été obtenue sans utiliser la graine.

III. Types de bruit étudiés

Le dossier `Bruit_Decorrele` contient l'étude d'un bruit parfaitement décorrélé

```
Sy = Sp+diag(diag((Sn)));
```

Cette étude se veut être proche de celle qui était faite par Hald. Elle donne des reconstructions de diagonale parfaites.

Le dossier `Bruit_Heterogene` contient l'étude d'un bruit corrélé par les termes croisés obtenus lors du calcul de la CSM à partir des spectres. Ce bruit est homogène sur la plupart des micros, sauf sur 10 d'entre eux, où il est 10dB plus fort.
Le dossier `Bruit_Homogene` contient l'étude d'un bruit corrélé par les termes croisés obtenus lors du calcul de la CSM à partir des spectres. Ce bruit est homogène sur tous les micros.

IV. Reconstruction de diagonale

4.1. Optimisation convexe (Hald) : `CSMRecHald`

Réglage unique : `cvx_precision('high')`

4.2. Alternating projection : `recdiag`

Code de Quentin Leclère. Réglages :
— `ro=1` : loop gain
— `nmax=1000` nombre d'itération max
— `tol=1e-7` l'algo s'arrête lorsque toutes les valeurs propres sont supérieures à `-tol`
— `timetol=30` l'algo s'arrête au bout de 30 secondes de calcul max

4.3. Optimisation linéaire (Dougherty) : `recdiagd.m`

Cet algo étant plus long, le calculs sont effectués sur 2 fois moins de paramètres.
Réglages :

- `nit=500` Nombre d'itérations max
- `timtol=60` l'algo s'arrête au bout de 60 sec max

V. RPCA

Code : `proximal_gradient_rpca.m` développé par Arvind Ganesh, 2009.
Réglages :

- λ paramètre de régularisation : le débruitage est effectué pour tous les λ allant de 0 à 1 par pas de 0,1.
- `maxIter=300` nombre d'itérations max
- `tol=1e-7` tolérance pour le critère d'arrêt
- `lineSearchFlag=-1` la direction de descente est calculée à chaque itération
- `continuationFlag=-1` à sa valeur par défaut : 1. Pour prendre en compte `mu`
- `eta=-1` paramètre pour la recherche de descente à sa valeur par défaut : 0.9
- `mu=-1` paramètre de relaxation à sa valeur par défaut : $1e-3$
- `outputFileName` Non renseigné

VI. EM : `EM_CSM_Fit.m` développé par Jérôme Antoni

Le bruit `Syc` est initialisé à `1e-16`.
La matrice de facteurs `L` est aussi initialisée à `1e-16`.
Le nombre de facteurs choisi est

```
if Nsnap(j)<Nmic
     k=Nsnap(j)-1;
else
     k=92;
end
```

J'ai testé différentes valeurs pour le nombre d'itérations et le seuil pour le critère d'arrêt et il semble difficile d'arrêter l'algo pour obtenir une erreur optimale étant donné que l'erreur diminue puis réaugmente (cf Rapport du 22 décembre 2017). Selon le temps de calcul disponible il faut donc fiwer le seuil bas et le nombre d'itération grand.

VII. MCMC : `MCMC_AnaFac_Quad_Sparse3.m` développé par Jérôme Antoni

7.1. Choix du nombre de facteurs

C'est la version sparse qui est utilisée pour le BeBeC, qui donne de bons résultats même si le nombre de facteur est grand. Le nombre de facteurs `k` est donc fixé à

```
k=Nsnap(i)+5;
if k>Nmic
        k=Nmic;
end
```

7.2. Initialisation

Pour le bruit : `noise` est la moyenne de la diagonale de la CSM bruitée.
la valeur moyenne pour les `alpha` sont les valeurs propres de la CSM bruitée, normalisée par la plus grande :

```
alpha2_mean = abs(real(sort(eig(Sy)/max(eig(Sy)),'descend')));
```

L'initialisation pour les `alpha` est

```
for k = 1:K_est
              a.alpha(k) = 1/alpha2_mean(k); %hyper-paramètre
end
Ini.alpha(1,:) = 1./a.alpha(:)';
```

Pour le paramètre `beta` :

```
[a.beta2,b.beta2] = Convert_InvGamma(mean(noise.^2),10*mean(noise.^2)); %hyper-paramètre
Ini.beta2(1,:) = b.beta2/a.beta2*ones(M,1);
```

avec `Convert_InvGamma` un code de Jérôme qui calcul les coefficients d'une loi Gamma à partir d'une moyenne et d'un écart-type.
Pour le paramètre `gamma` :

```
gamma_mean = (real(trace(Sy))/M)/mean(alpha2_mean); %pas de bruit retiré a priori
[a.gamma2,b.gamma2] = Convert_InvGamma(gamma_mean,10*gamma_mean); %hyper-paramètre
Ini.gamma2(1) = b.gamma2/a.gamma2;
```

Pour `Lambda` :

```
Ini.Lambda(1,:,:) = (randn(M,K_est) + 1i*randn(M,K_est))/sqrt(2);
```

7.3. Nombre d'itération de la chaîne

Certains résultats sont obtenus pour 500 itérations, par manque de temps de calcul. Dans ce cas, la moyenne des diagonales est effectuée sur les 200 dernières itérations.
D'autres résultats sont obtenus pour 1000 itérations. Dans ce cas, la moyenne des diagonales est obtenue à partir des 500 dernières itération. Dans ce cas, double le nombre d'itérations améliore le résultat d'environ 1-2 dB.

7.4. Autres remarques

Ce code est parallélisé avec `parfor` ce qui explique l'utilisation de nombreuses variables temporaires.

VIII. Affichage

Les figures sont tracées en utilisant les scripts nommés `plot_XX` avec `XX` le nom du paramètre concerné. Ces figures sont ensuite exportées en utilisant l'outil `matlab2tikz`.