



STATISTIQUE
SCIENCE DES DONNÉES BIOSTATS
UNIVERSITÉ DE MONTPELLIER

Olivier CÔME

Nicolas PRALON

N^o étudiant Olivier CÔME: 22110708

N^o étudiant Nicolas PRALON: 22110681

olivier.come@etu.umontpellier.fr

nicolas.pralon@etu.umontpellier.fr

Master 2

Statistique et Science des Données

Université de Montpellier

Projet

HAX006X : Modèles à variables latentes

Rédigé le 24 mars 2023 en L^AT_EX

Sommaire

1	Introduction	2
2	L'algorithme EM	3
2.1	Implémentation de la fonction simulation	3
2.2	Implémentation de la fonction EM	4
2.3	Étude sur des données simulées	5
2.3.1	Simulation d'un mélange de deux gaussiennes	5
2.3.2	Simulation d'un mélange à quatre gaussiennes	6
2.4	Comparaison de la fonction <i>EM</i> avec <i>mixtools</i> sur des données réelles	7
3	Conclusion	11
4	Annexes	12
4.1	Script de la fonction <i>simulation</i>	12
4.2	Script de la fonction <i>EM</i>	13
4.3	Script de la fonction <i>plot_distrib</i>	14
5	Bibliographie	15

1 Introduction

Le présent projet s'inscrit dans le cadre de l'unité d'enseignement "HAX006X Modèles à variables latentes". L'objectif est ici d'appliquer sur des données réelles, plusieurs méthodes étudiées durant ce cours. Nous nous intéresserons à deux d'entre elles à savoir la méthode par thèmes et l'algorithme EM (Expectation-Maximization) conçue par Dempster et al., dont l'article est disponible via la source [1].

2 L'algorithme EM

Dans cette section nous allons nous intéresser à l'algorithme EM (Expectation-Maximization) afin d'estimer les paramètres α , μ et σ d'un mélange gaussien. Nous avons implémenté cet algorithme via la fonction *EM* (présente dans notre script *R*) en nous aidant de la source [2]. Afin de tester l'efficacité de notre implémentation, nous allons dans un premier temps l'exécuter sur des données simulées. En effet, dans notre script, nous avons implémenté une autre fonction que nous avons nommée *simulation*. Cette dernière nous permettra de générer de manière aléatoire, un échantillon issu d'un mélange gaussien. Nous décrirons plus en détail cette fonction, en aval. Dans un second temps, nous exécuterons notre algorithme sur de vraies données afin de voir s'il est robuste. Pour cela nous utiliserons le jeu de données galaxies de la librairie MASS et nous le décrirons ultérieurement.

2.1 Implémentation de la fonction simulation

Comme il a été mentionné précédemment nous allons réaliser dans un premier temps une étude sur des données simulées à partir de la fonction *simulation* (le code de son implémentation est disponible en annexes). Décrivons cette dernière, elle prend en argument :

- **dt_param** : Le dataframe contenant les paramètres α , μ et σ
- **n** : La taille de l'échantillon

Elle retourne un vecteur de taille n qui sera l'échantillon du mélange gaussien.

Regardons un plus en détail comment a été conçue cette fonction.

La partie la plus importante et la plus subtile de ce script est celle dans laquelle nous distribuons aléatoirement les $(X_i)_{i \in 1, \dots, n}$ de l'échantillon de sorte à avoir un bon mélange gaussien.

Afin de simplifier les choses, rien de mieux que de prendre un exemple. Dans celui-ci, l'objectif sera de générer un mélange de quatre gaussiennes, ayant pour paramètres respectifs $\theta_1 = (\alpha_1, \mu_1, \sigma_1)$, $\theta_2 = (\alpha_2, \mu_2, \sigma_2)$, $\theta_3 = (\alpha_3, \mu_3, \sigma_3)$ et $\theta_4 = (\alpha_4, \mu_4, \sigma_4)$.

Les $(\alpha_j)_{j \in \{1, \dots, 4\}}$ étant ici des probabilités, nous avons que :

$$\sum_{j=1}^4 \alpha_j = 1$$

La démarche est la suivante :

- On tire $Z \sim \mathcal{U}(0, 1)$
- Si $Z < \alpha_1$ alors $X \sim \mathcal{N}(\mu_1, \sigma_1)$
- Sinon si $\alpha_1 < Z < \alpha_1 + \alpha_2$ alors $X \sim \mathcal{N}(\mu_2, \sigma_2)$
- Sinon si $\alpha_1 + \alpha_2 < Z < \alpha_1 + \alpha_2 + \alpha_3$ alors $X \sim \mathcal{N}(\mu_3, \sigma_3)$
- Sinon si $\alpha_1 + \alpha_2 + \alpha_3 < Z < \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4$ alors $X \sim \mathcal{N}(\mu_4, \sigma_4)$

Notez que notre implémentation marche dans le cas général d'un mélange de J gaussiennes avec $J \in \mathbb{N}^*$.

2.2 Implémentation de la fonction EM

La fonction *EM* est sans aucun doute celle la plus importante de cette section, il est donc primordial de la décrire (le code de son implémentation est disponible en annexes).

Tout d'abord, pour implémenter cette dernière, nous nous sommes fortement aidés du pseudo-code suivant :

Algorithm 1 L'algorithme EM (Dempster et al., 1977).

Entrée(s) : $N \in \mathbb{N}$, $\hat{\theta}_0 \in \Theta$, un jeu de données $x_1 \dots x_n$;

Initialisation ;

- 1: $k := 1$;
 - 2: **Tant que** $K < N + 1$ **faire**
 - 3: **ETAPE E :** Calculer la fonction $Q(\theta; \hat{\theta}_{k-1}) = \frac{1}{n} \sum_{i=1}^n \mathbb{E}_{\hat{\theta}_{k-1}} [\log f(X_i, Z_i, \theta) | X_i = x_i]$;
 - 4: **ETAPE M :** $\hat{\theta}_k = \operatorname{argmax} Q(\theta; \hat{\theta}_{k-1})$;
 - 5: $k \leftarrow k + 1$;
 - 6: **fin du Tant que ;**
 - 7: **retourner** $\hat{\theta}_N$;
-

La fonction *EM* prend en argument :

- *dt_init*, le dataframe contenant les paramètres initiaux (α_{init} , μ_{init} , σ_{init})
- *X*, les données (réelles ou simulées) issues d'un mélange gaussien
- *K* le nombre d'itérations souhaitées pour l'algorithme

Elle retourne un dataframe contenant les valeurs des paramètres estimées par l'algorithme, à savoir α , μ et σ .

Les formules que nous avons utilisées pour calculer l'étape E et M et qui sont présentées ci dessous sont issues de la source [2].

- Lors de l'étape E nous déterminons la probabilité $\mathbb{P}_{\hat{\theta}}(Z = j | X = X_i)$ via la formule suivante :

$$\mathbb{P}_{\hat{\theta}}(Z = j | X = X_i) = \frac{\alpha_j \times \gamma_{\mu_j, \hat{\sigma}_j}}{\sum_{k=1}^J \alpha_k \times \gamma_{\mu_k, \hat{\sigma}_k}}$$

- Lors de l'étape M, nous déterminons les estimations des estimateurs du maximum de vraisemblance ($\hat{\alpha}_j$, $\hat{\mu}_j$, $\hat{\sigma}_j$) via les formules suivantes :

$$\begin{aligned} \hat{\alpha}_j &= \frac{1}{n} \sum_{i=1}^n (Z = j | X = X_i) \\ \hat{\mu}_j &= \frac{\sum_{i=1}^n X_i (Z = j | X = X_i)}{\sum_{i=1}^n (Z = j | X = X_i)} \\ \hat{\sigma}_j &= \frac{\sum_{i=1}^n (X_i - \hat{\mu}_j)^2 (Z = j | X = X_i)}{\sum_{i=1}^n (Z = j | X = X_i)} \end{aligned}$$

Comme en témoigne la section 3.3 (Preuve de la croissance de la vraisemblance d'une itération à l'autre) de la source [2], il est théoriquement prouvé que l'algorithme permet de faire croître la log-vraisemblance des observations en les paramètres itérativement créés. Cependant, il est important de notifier le fait qu'il n'existe pas de convergence de la suite de paramètres établie par l'algorithme EM. En effet, ces derniers peuvent rester bloqués dans des extremas locaux. On comprend donc qu'il est primordial de choisir de bons paramètres initiaux afin de ne pas être confronté à ce problème. Dans la section consacrée à l'étude sur de vraies données, nous expliciterons la procédure qui a été mise en place pour choisir ces paramètres initiaux.

2.3 Étude sur des données simulées

Cette sous-section sera consacrée à l'étude menée sur des données simulées à partir de notre fonction *simulation*.

2.3.1 Simulation d'un mélange de deux gaussiennes

Nous avons ici décidé de générer un échantillon de taille 100 issu d'un mélange de deux gaussiennes de lois respectives $\mathcal{N}(\mu_1, \sigma_1) = \mathcal{N}(50, 11)$ et $\mathcal{N}(\mu_2, \sigma_2) = \mathcal{N}(220, 50)$. La densité associée à cet échantillon a été estimée de manière non paramétrique à partir d'une méthode à noyau et elle a été tracée sur la figure 1.

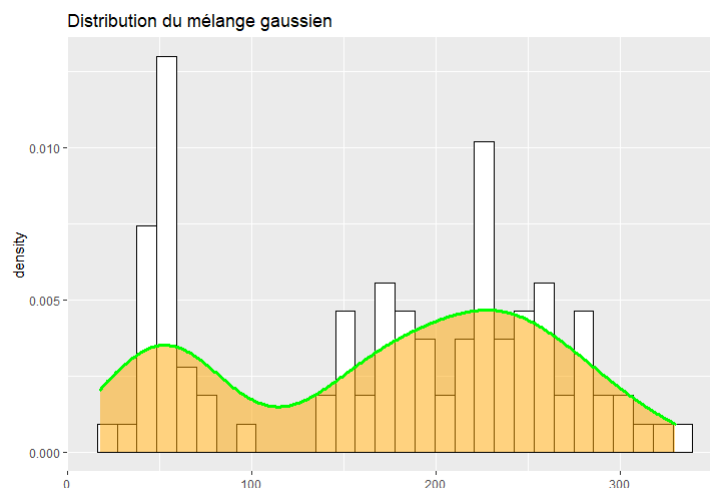


FIGURE 1 – Densité d'un mélange à 2 gaussiennes estimée par une méthode à noyau

Nous avons ensuite appliqué notre algorithme (fonction EM) sur cet échantillon. Le tableau 1 contient les valeurs des vrais paramètres de ce mélange simulé. Comme il a été mentionné précédemment, le choix des paramètres initiaux est crucial si l'on veut que l'algorithme estime correctement les paramètres du mélange. Le tableau 2 contient les valeurs des paramètres initiaux utilisés. Comme vous pouvez le voir en observant ces deux tableaux, nous avons choisi des paramètres initiaux assez proches des vrais paramètres (sauf pour la variance du 2ème mélange) de manière à ne pas être bloqué dans des extremas locaux. Les résultats obtenus par notre algorithme sont affichés sur la capture d'écran de la figure 2.

Comme on peut le voir sur la figure 2, les valeurs estimées par notre implémentation sont proches de celles des vrais paramètres (voir tableau 1). Ces résultats nous montrent donc que notre fonction *EM* fonctionne correctement, ce qui est rassurant.

	α	μ	σ
Paramètres du 1er mélange	0.4	50	11
Paramètres du 2ème mélange	0.6	220	50

TABLE 1 – Vrais paramètres du mélange

	α_{init}	μ_{init}	σ_{init}
Paramètres du 1er mélange	0.2	30	21
Paramètres du 2ème mélange	0.8	280	160

TABLE 2 – Paramètres initiaux

	mixtureParameters	alpha	mu	sigma
1	parameters of Mixture1	0.4584696	49.08584	9.983414
2	parameters of Mixture2	0.5415304	227.40612	55.429273

FIGURE 2 – Paramètres du mélange gaussien estimés par notre fonction *EM*

2.3.2 Simulation d'un mélange à quatre gaussiennes

Nous avons ici décidé de générer un échantillon de taille 1000 issu d'un mélange de quatre gaussiennes de lois respectives :

- $\mathcal{N}(\mu_1, \sigma_1) = \mathcal{N}(35, 11)$
- $\mathcal{N}(\mu_2, \sigma_2) = \mathcal{N}(350, 22)$
- $\mathcal{N}(\mu_3, \sigma_3) = \mathcal{N}(720, 32)$
- $\mathcal{N}(\mu_4, \sigma_4) = \mathcal{N}(1198, 55)$

La densité associée à cet échantillon a été estimée de manière non paramétrique à partir d'une méthode à noyau et elle a été tracée sur la figure 3.

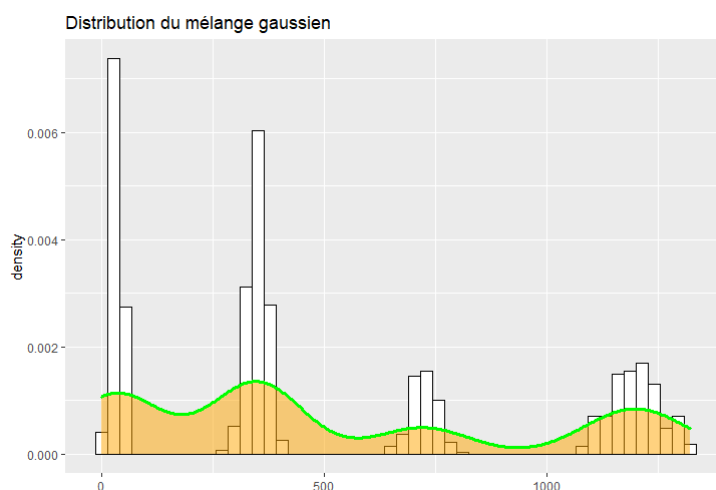


FIGURE 3 – Densité d'un mélange à 4 gaussiennes estimée par une méthode à noyau

Nous avons ensuite appliqué notre algorithme (fonction *EM*) sur cet échantillon. Le tableau 3 contient les valeurs des vrais paramètres de ce mélange simulé. Comme il a été mentionné précédemment, le choix des paramètres initiaux est crucial si l'on veut que l'algorithme estime correctement les paramètres du mélange. Le tableau 4 contient les valeurs des paramètres initiaux utilisés. Comme vous pouvez le voir en observant ces deux tableaux, nous avons choisi des paramètres initiaux assez proches des vrais de manière à ne pas être bloqué dans des extremas locaux. Les résultats obtenus par notre algorithme sont affichés sur la capture d'écran de la figure 4.

Comme on peut le voir sur la figure 4, les valeurs estimées par notre implémentation sont proches de celles des vrais paramètres (voir tableau 3). Ces résultats nous confortent une fois de plus dans l'idée que notre fonction *EM* fonctionne correctement.

	α	μ	σ
Paramètres du 1er mélange	0.30	35	11
Paramètres du 2ème mélange	0.33	350	22
Paramètres du 3ème mélange	0.15	720	32
Paramètres du 4ème mélange	0.22	1198	55

TABLE 3 – Vrais paramètres du mélange

	α	μ	σ
Paramètres du 1er mélange	0.33	30	10
Paramètres du 2ème mélange	0.30	370	25
Paramètres du 3ème mélange	0.17	717	30
Paramètres du 4ème mélange	0.20	1238	57

TABLE 4 – Paramètres initiaux choisis

	mixtureParameters	alpha	mu	sigma
1	parameters of Mixture1	0.296	35.40873	11.22125
2	parameters of Mixture2	0.318	350.93412	21.71632
3	parameters of Mixture3	0.141	722.17459	32.45836
4	parameters of Mixture4	0.245	1199.41438	49.72699

FIGURE 4 – Paramètres du mélange gaussien estimés par notre fonction *EM*

2.4 Comparaison de la fonction *EM* avec *mixtools* sur des données réelles

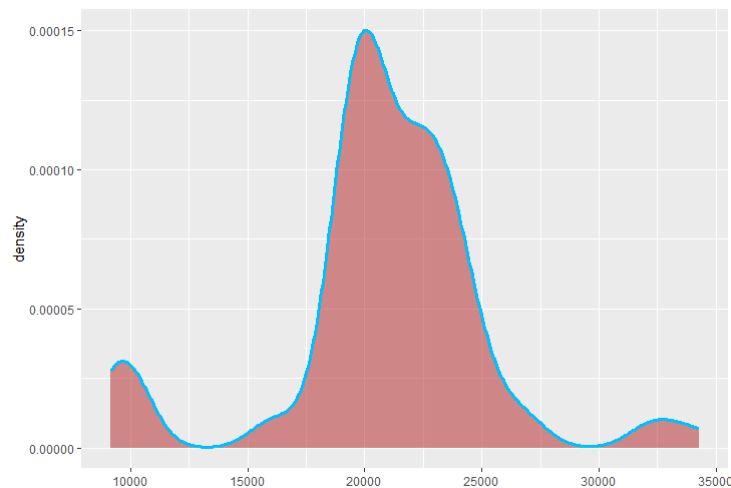
Dans cette section, l'étude sera menée sur des données réelles. Nous utiliserons le jeu de données *galaxies* provenant de la librairie *MASS* de *R*. Dans un premier temps, nous allons estimer les paramètres du mélange associés à ce dataset via l'utilisation de notre implémentation de l'algorithme EM (la fonction *EM*). Nous estimerons ensuite une seconde fois les paramètres de ce même mélange mais cette fois-ci, en utilisant la fonction *normalmixEM* prédéfinie de *R* qui est disponible via le package *mixtools*. Nous avons choisi d'utiliser cette librairie car l'algorithme EM y est implémenté et il est utilisé par la fonction *normalmixEM* pour estimer les paramètres d'un mélange. Le fait de comparer les résultats de notre fonction avec ceux obtenus par celle prédéfinie de *R* nous permettra d'évaluer la performance et la robustesse de notre implémentation sur de vraies données.

Le jeu de données *Galaxies* est un vecteur numérique qui représente les vitesses en km/s (kilomètre par seconde) de 82 galaxies. La figure 5 est un extrait de ce jeu de données.

```
> galaxies
[1] 9172 9350 9483 9558 9775 10227 10406 16084 16170 18419 18552 18600 18927
[14] 19052 19070 19330 19343 19349 19440 19473 19529 19541 19547 19663 19846 19856
[27] 19863 19914 19918 19973 19989 20166 20175 20179 20196 20215 20221 20415 20629
[40] 20795 20821 20846 20875 20986 21137 21492 21701 21814 21921 21960 22185 22209
[53] 22242 22249 22314 22374 22495 22746 22747 22888 22914 23206 23241 23263 23484
[66] 23538 23542 23666 23706 23711 24129 24285 24289 24366 24717 24990 25633 26690
[79] 26995 32065 32789 34279
```

FIGURE 5 – Extrait du jeu de données *galaxies*

Comme il a été mentionné dans la section 2.2, si l'on veut obtenir de bonnes estimations pour les paramètres du mélange, il est primordial de sélectionner correctement les paramètres qui serviront de conditions initiales pour l'algorithme. Nous allons donc détailler la stratégie qui a été mise en place pour sélectionner ces derniers. Étant donné que nous ne connaissons pas la vraie densité associée à ces données, nous avons utilisé dans un premier temps une méthode d'estimation non paramétrique (à noyau) afin d'estimer cette dernière. La figure 6 représente la courbe de densité estimée par la méthode à noyau.

FIGURE 6 – Densité du dataset *galaxies* estimée par une méthode à noyau

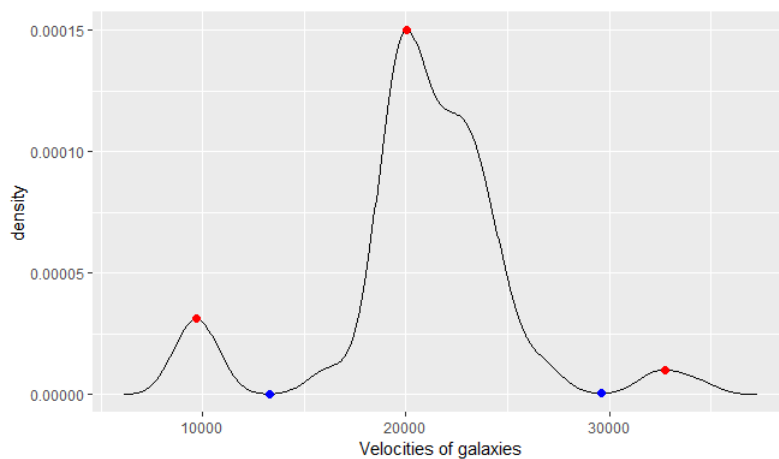
En observant la figure 6, on peut facilement distinguer 3 pics principaux. Un premier vers 9800 sur l'axe des abscisses, un deuxième vers 21000 et un 3ème vers 32000. Il semblerait aussi y avoir une sorte de pic vers 24000 mais celui-ci n'étant pas clairement visible nous ne le considérerons pas. On supposera donc pour la suite de l'étude que nous sommes dans le cas d'un mélange à 3 composantes.

Il nous faudra donc déterminer les paramètres :

- α_1, α_2 et α_3 qui sont les proportions associées aux 3 gaussiennes
- μ_1, μ_2 et μ_3 qui sont les moyennes des 3 gaussiennes
- σ_1, σ_2 et σ_3 qui sont les écarts-types des 3 gaussiennes

Pour commencer, nous dirons que les paramètres $(\alpha_j)_{j \in \{1, \dots, 3\}}$ qui seront utilisés dans les conditions initiales seront tous égaux, c'est à dire que $(\alpha_1)_{init} = (\alpha_2)_{init} = (\alpha_3)_{init} = \frac{1}{3}$ (car il faut que $\sum_{i=1}^3 \alpha_j = 1$).

Pour la recherche des paramètres $(\mu_j)_{j \in \{1, \dots, 3\}}$ et $(\sigma_j)_{j \in \{1, \dots, 3\}}$ initiaux, nous allons nous aider de la figure 7.

FIGURE 7 – Extremums locaux de la densité estimée du dataset *galaxies*

À l'aide de la fonction *find_peaks()* du package *ggpmisc* et de la fonction *local.min.max()* du package *spatialEco*, nous avons pu déterminer les extremums locaux de la courbe de densité estimée des données *galaxies*. Les maximums locaux sont représentés par les points rouges et les minimums locaux par les points bleus sur la figure 7.

Les paramètres $(\mu_j)_{j \in \{1, \dots, 3\}}$ qui seront utilisés dans les conditions initiales seront donc les abscisses respectives de ces 3 maximums locaux. Nous aurons donc :

- $(\mu_1)_{init} = 9698.471$
- $(\mu_2)_{init} = 20050.850$
- $(\mu_3)_{init} = 32717.292$

Les $(\alpha_j)_{j \in \{1, \dots, 3\}}$ et les $(\mu_j)_{j \in \{1, \dots, 3\}}$ initiaux ayant été déterminés, il ne nous reste plus qu'à trouver les $(\sigma_j)_{j \in \{1, \dots, 3\}}$ initiaux. Nous allons les déterminer en calculant les écarts-types de chacun des 3 pics. Tout d'abord, nous scindons nos données en 3 intervalles car il y a 3 pics. C'est à ce moment là qu'interviennent les minimas locaux. En effet, c'est eux qui vont justement nous permettre de délimiter notre jeu de données en 3 intervalles.

Le premier intervalle sera composé des abscisses allant du début du jeu de données jusqu'à l'abscisse du premier minimum local. Le deuxième intervalle quant à lui sera composé des valeurs des abscisses allant du premier minimum local jusqu'au deuxième minimum local. Enfin le dernier intervalle contiendra les abscisses allant du deuxième minimum local jusqu'à la fin du jeu de données. Dans la liste à puces ci-dessous nous avons expliciter les bornes *inf* et *sup* de chacun des trois intervalles :

- **Première intervalle** : [6166.482; 13291.36]
- **Deuxième intervalle** : [13291.36; 29611.58]
- **Troisième intervalle** : [29611.58; 37284.52]

En appliquant la fonction $sd()$ sur le premier intervalle nous obtiendrons $(\sigma_1)_{init}$. En appliquant la fonction $sd()$ sur le deuxième intervalle nous obtiendrons $(\sigma_2)_{init}$. Nous procéderons de même pour avoir $(\sigma_3)_{init}$. Au final, nous obtenons que :

- $(\sigma_1)_{init} = 2083.124$
- $(\sigma_2)_{init} = 4737.603$
- $(\sigma_3)_{init} = 2241.339$

L'ensemble des paramètres initiaux qui ont été déterminés précédemment sont résumés dans le tableau 5.

	α_{init}	μ_{init}	σ_{init}
Paramètres du 1er mélange	$\frac{1}{3}$	9698.471	2083.124
Paramètres du 2ème mélange	$\frac{1}{3}$	20050.850	4737.603
Paramètres du 3ème mélange	$\frac{1}{3}$	32717.292	2241.339

TABLE 5 – Paramètres initiaux

Maintenant que nous avons toutes nos conditions initiales, nous sommes en mesure d'exécuter notre implémentation de l'algorithme *EM* sur les vraies données *galaxies*. Les paramètres du mélange estimés par notre fonction *EM* sont visibles sur la capture d'écran (figure 8). Nous les avons également résumé dans le tableau 6 pour une meilleure visibilité.



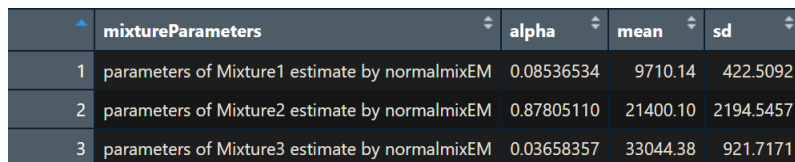
	mixtureParameters	alpha	mu	sigma
1	parameters of Mixture1	0.08536534	9710.14	422.5092
2	parameters of Mixture2	0.87805110	21400.10	2194.5457
3	parameters of Mixture3	0.03658357	33044.38	921.7171

FIGURE 8 – Paramètres estimés par notre fonction *EM*

	α	μ	σ
Paramètres du 1er mélange estimés par <i>EM</i>	0.08536534	9710.14	422.5092
Paramètres du 2ème mélange estimés par <i>EM</i>	0.87805110	21400.10	2194.5457
Paramètres du 3ème mélange estimés par <i>EM</i>	0.03658357	33044.38	921.7171

TABLE 6 – Paramètres estimés par notre fonction *EM*

Maintenant que nous avons les paramètres du mélange, estimés par notre implémentation, comparons les avec ceux obtenus en utilisant la fonction *normalmixEM* de *R* dans laquelle est implémenté l'algorithme EM. La figure 9 est une capture d'écran dans laquelle sont stockés tous les paramètres estimés par la fonction *normalmixEM* de *R*. Nous les avons également résumé dans le tableau 7 pour une meilleure visibilité.



	mixtureParameters	alpha	mean	sd
1	parameters of Mixture1 estimate by normalmixEM	0.08536534	9710.14	422.5092
2	parameters of Mixture2 estimate by normalmixEM	0.87805110	21400.10	2194.5457
3	parameters of Mixture3 estimate by normalmixEM	0.03658357	33044.38	921.7171

FIGURE 9 – Paramètres estimés par *normalmixEM*

	α	μ	σ
Paramètres du 1er mélange estimés par <i>normalmixEM</i>	0.08536534	9710.14	422.5092
Paramètres du 2ème mélange estimés par <i>normalmixEM</i>	0.87805110	21400.10	2194.5457
Paramètres du 3ème mélange estimés par <i>normalmixEM</i>	0.03658357	33044.38	921.7171

TABLE 7 – Paramètres estimés par notre fonction *EM*

En regardant les tableaux 6 et 7, on constate que les valeurs des paramètres estimées avec notre fonction *EM* et celles estimées avec la fonction *normalmixEM* sont exactement les mêmes. Cela nous montre donc que notre implémentation de l'algorithme EM est robuste et qu'elle peut être aussi utilisée sur des données réelles de mélanges. Notre fonction *EM* fournira donc de bonnes estimations aussi bien sur des données simulées que sur des données réelles à partir du moment où les conditions initiales sont correctement choisies.

3 Conclusion

Au vu des résultats et des comparaisons faites entre notre fonction *EM* et celle *normalmixEM* du package *mixtools*, nous pouvons conclure que notre implémentation fournira de bonnes estimations aussi bien sur des données simulées que sur des données réelles de mélanges à condition bien évidemment de choisir correctement les conditions initiales. Une piste d'amélioration possible de notre fonction serait d'y ajouter une fonctionnalité de recherche automatique de paramètres initiaux. D'après la littérature, plusieurs techniques peuvent être envisagées. Parmi celles qui reviennent le plus souvent, on retrouve la méthode des kmeans.

4 Annexes

4.1 Script de la fonction *simulation*

Ci dessous, l'export de code de la fonction *simulation*.

```
simulation = function(dt_param, n=100){
  X = rep(NA,n) #echantillon
  vect_alpha = dt_param[,2]
  vect_mean = dt_param[,3]
  vect_sd = dt_param[,4]
  for(i in 1:n){
    Z = runif(1)
    if (Z <= vect_alpha[1]){
      X[i] = rnorm(1, vect_mean[1], vect_sd[1])
    }else{
      k = 1
      l = 2
      Bool = FALSE
      cumul_alpha = cumsum(vect_alpha)
      while(Bool == FALSE){
        if((cumul_alpha[k]<=Z) & (cumul_alpha[l]>=Z)){
          Bool = TRUE
          param_index = l
        }
        k = k+1
        l = l+1
      }
      X[i] = rnorm(1, vect_mean[param_index], vect_sd[param_index])
    }
  }
  return(X)
}
```

4.2 Script de la fonction *EM*

Ci dessous, l'export de code de la fonction *EM*.

```
EM = function(dt_init, X, K){
  J = dim(dt_init)[1]
  n = length(X)
  data_stateE = param_State_E(n, J)

  for(k in 1:K){
    vect_alpha = dt_init[,2] #de longueur J
    vect_mean = dt_init[,3]
    vect_sd = dt_init[,4]
    # vecteur contenant la somme des des numerateurs de P_thetat(j|X = X_i)
    # pour chaque valeur de l echantillon
    v = rep(0,n)

    # Etape E
    for(j in 1:J){
      # On remplit le tableau param_State_E contenant P_thetat(j|X=X_i)
      data_stateE[,j] = vect_alpha[j]*dnorm(X,vect_mean[j],vect_sd[j])
      v = v+data_stateE[,j]
    }
    for(j in 1:J){
      data_stateE[,j] = data_stateE[,j]/v
    }

    # Etape M
    H = data_stateE
    for(col in 2:4){ #on met a jour le dt_init
      for(ind in 1:J){

        # on met a jour les alpha
        if(col == 2){
          dt_init[,col][ind] = mean(H[,ind])
        }
        # on met a jour les mu
        if(col == 3){
          dt_init[,col][ind] = (sum(X*H[,ind]))/(sum(H[,ind]))
        }
        # on met a jour les sigma
        if(col == 4){
          dt_init[,col][ind] = sqrt((sum( (X-rep(dt_init[,col-1][ind],n))^2
                                           *H[,ind] ))/sum(H[,ind]))
        }
      }
    }
  }
  new_df = dt_init
  colnames(new_df) = c('mixtureParameters', 'alpha', 'mu', 'sigma')
  return(new_df)
}
```

4.3 Script de la fonction *plot_distrib*

Ci dessous, l'export de code de la fonction *plot_distrib*.

Cette fonction permet d'afficher l'histogramme des données ainsi que la courbe de densité estimée associée à ces données. Cette courbe de densité sera superposée à l'histogramme.

```
plot_distrib = function(df, X){  
  data_distrib = data.frame(gaussian_mixture = X)  
  ggplot(data_distrib, aes(x=data_distrib[, 'gaussian_mixture'])) +  
    geom_histogram(aes(y=..density..), colour="black", fill="white", bins = 50) +  
    geom_density(alpha=.5, color = "green", fill="orange", size=1.2) +  
    ggtitle("Distribution du melange gaussien") + xlab("")  
}
```

5 Bibliographie

- [1] Dempster A.P., Laird N. M., Rubin D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm, Journal of the Royal Statistical Society, Series B, Vol. 39, 1, 1-38
- [2] Frédéric Santos (2015). L'algorithme EM : une courte présentation <https://members.loria.fr/moberger/Enseignement/AVR/Exposes/algo-em.pdf>