Cassiopée

Pradel Guillaume, Aidan Alex, Lasbordes Maxence

Table des matières

1 Introduction

L'algorithme ICE (Iterative Conditional Estimator) est un algorithme itératif permettant d'estimer les paramètres des lois avec lesquels ont été généré des données. Le but principal de ce travail est de réestimer les paramètres dans le cas où les sources sont des chaines de Markov cachées. Mais avant, nous allons étudier le cas où les sources sont indépendantes et identiquement distribuées.

2 Algorithme ICE dans le cas d'une mixture de gaussiennes

2.1 Modélisation mathématique

Le principe général va être d'estimer les paramètres $\mu_{\{0,1\}}$. Dans le cas d'une mixture de Gaussienne, les sources $R_i \in \{0,1\}$ sont choisies de manière indépendantes et identiquement distribuées. Les observations générées sont ici définies par les variables aléatoires $X_i \in \mathbb{R}$, et ce, pour tout $i \in \{1,..,n\}$. La loi de X conditionnellement à $R = 0$ est une loi normale de moyenne μ_0 et d'écart type σ_0 :

$$\sim \mathcal{N}(\mu_0, \sigma_0)$$

De même, la loi de X conditionnellement à $R = 1$ est une loi normale de moyenne μ_1 et d'écart type σ_1 :

$$\sim \mathcal{N}(\mu_1, \sigma_1)$$

On note également $\theta = (\alpha, \boldsymbol{\mu_0}, \boldsymbol{\mu_1}, \boldsymbol{\sigma_0}, \boldsymbol{\sigma_1})$, où $\alpha = P(R_i = 0)$. Ces paramètres définissent entièrement la loi de $(R, X) = ((R_1, ..., R_n), (X_1, ..., X_n))$. On cherche ici à estimer le paramètre θ à partir de X.

Pour initialiser l'algorithme ICE, nous tirons $\forall i \in \{1, ..., n\}, R_i$ selon une loi de Bernoulli de paramètre 0.5 car en effet, l'algorithme ICE suppose d'abord l'accès à une estimation des R_i.
Notons

$$U = \mathbb{1}_{[R_1=0]} + ... + \mathbb{1}_{[R_n=0]}$$

et

$$V = X_1\mathbb{1}_{[R_1=0]} + ... + X_n\mathbb{1}_{[R_n=0]}$$

On pose pour la suite :

$$\hat{\alpha} = \frac{U}{n}$$

$$\hat{\mu}_0 = \frac{V}{U}$$

$$\hat{\sigma}_0 = \sqrt{\frac{1}{U}\sum_{i=1}^{n}(X_i\mathbb{1}_{[R_i=0]} - \hat{\mu}_0)^2}$$

qui représentent respectivement la fréquence, la moyenne et la variance empirique. On initialise alors $\boldsymbol{\theta_0} = (\hat{\alpha}, \hat{\boldsymbol{\mu}}_{\mathbf{0}}, \hat{\boldsymbol{\mu}}_{\mathbf{1}}, \hat{\boldsymbol{\sigma}}_{\mathbf{0}}, \hat{\boldsymbol{\sigma}}_{\mathbf{1}})$

Supposons d'abord que nous avons accès à une estimation de $R = (R_1, ..., R_n)$. On se place à la k-ième itération, nous avons donc $\boldsymbol{\theta_k} = (\alpha_{\mathbf{k}}, \boldsymbol{\mu}_{\mathbf{0,k}}, \boldsymbol{\mu}_{\mathbf{1,k}}, \boldsymbol{\sigma}_{\mathbf{0,k}}, \boldsymbol{\sigma}_{\mathbf{1,k}})$.

Nous allons calculer $\forall i \in \{1, .., n\}, P(X_i = x_i | R_i = 0)$, et comme X suit une loi gaussienne conditionnellement à $R = 0$:

$$P(X_i = x_i | R_i = 0) = \frac{1}{2\pi\sigma_{0,k}^2} \exp\left(-\frac{(x_i - \mu_{0,k})^2}{2\sigma_{0,k}^2}\right)$$

On peut alors maintenant calculer, avec la formule de Bayes :

$$P(X_i = x_i \cap R_i = 0) = P(X_i = x_i | R_i = 0)P(R_i = 0)$$

Pour rappel, nous avons bien accès à $P(R_i = 0)$ car il vaut α_k. Ainsi, nous calculons :

$$P(R_i = 0 | X_i = x_i) = \frac{P(X_i = x_i \cap R = 0)}{P(X_i = x_i \cap R_i = 0) + P(X = x_i \cap R_i = 1)}$$

Avec maintenant la connaissance des probabilités à postériori, nous pouvons estimer α :

$$\alpha_{k+1} = \frac{1}{n}\sum_{i=1}^{n} P(R_i = 0|X_i = x_i)$$

On simule maintenant n réalisation $r^1, ..., r^n$ du vecteur $R = (R_1, ..., R_n)$ selon sa loi conditionnelle à X calculé précédemment par les probabilités à posteriori. Ainsi, on tire les r_i^j selon une loi de Bernoulli de paramètre $1 - P(R_i = 0|X_i = x_i)$. On calcule ensuite U et V pour ces réalisations r^j, puis on calcule $\hat{\mu}_0$ et $\hat{\sigma}_0$.

On réestime ainsi la valeur de μ_0 :

$$\mu_{0,k+1} = \frac{1}{n}(\hat{\mu}_{0,k}^1 + ... + \hat{\mu}_{0,k}^n)$$

Et de $\hat{\sigma}_0$:

$$\sigma_{0,k+1} = \frac{1}{n}(\sigma_{0,k}^1 + ... + \sigma_{0,k}^n)$$

Il reste maintenant à réestimer μ_1 et σ_1, que l'on trouve de manière analogue en changeant U et V par :

$$U = \mathbb{1}_{[R_1=1]} + ... + \mathbb{1}_{[R_n=1]}$$

$$V = X_1\mathbb{1}_{[R_1=1]} + ... + X_n\mathbb{1}_{[R_n=1]}$$

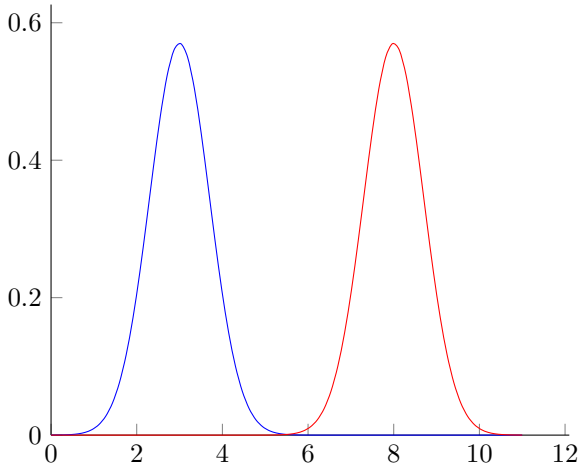
Nous avons alors la valeur suivante $\theta_{\mathbf{k+1}}$

2.2 Modélisation informatique

2.2.1 Quelques simulations

Pour les 3 premières simulations, il s'agira de faire varier les nombres d'itérations de l'algorithme ICE pour des moyennes et variances fixées. On fixe aussi le nombre d'observations à $n = 50$. L'algorithme ICE va donc renvoyer une estimation de la moyenne et de la variance, mais surtout une estimation des R_i, ce que nous allons étudier dans les prochaines simulations.

Commençons d'abord avec le cas le plus simple, c'est à dire avec μ_0 et μ_1 très distincts et avec σ_0 et σ_1 assez faible.



Les observations générées par ces 2 gaussiennes seront beaucoup plus facilement classifiable relativement à $R = 0$ ou $R = 1$.

Pour la *Figure 1*, on observe une convergence très rapide vers un taux de correspondance de 1.0 après seulement une dizaine d'itérations. On a les centres des gaussiennes qui sont suffisamment éloignés l'un de l'autre de telle sorte à permettre une séparation claire et nette des clusters. L'algorithme ICE parvient de ce fait à identifier les clusters rapidement et de manière précise.

Nous pouvons également tester les limites de notre algorithme dans des cas plus compliqués. Pour cela, nous pouvons déjà faire quelques simulations dans le cas où les moyennes sont assez proches, et où les gaussiennes se chevauchent.

Pour $\mu_0 = 4$, $\mu_1 = 6$, $\sigma_0 = \sigma_1 = 0.7$:

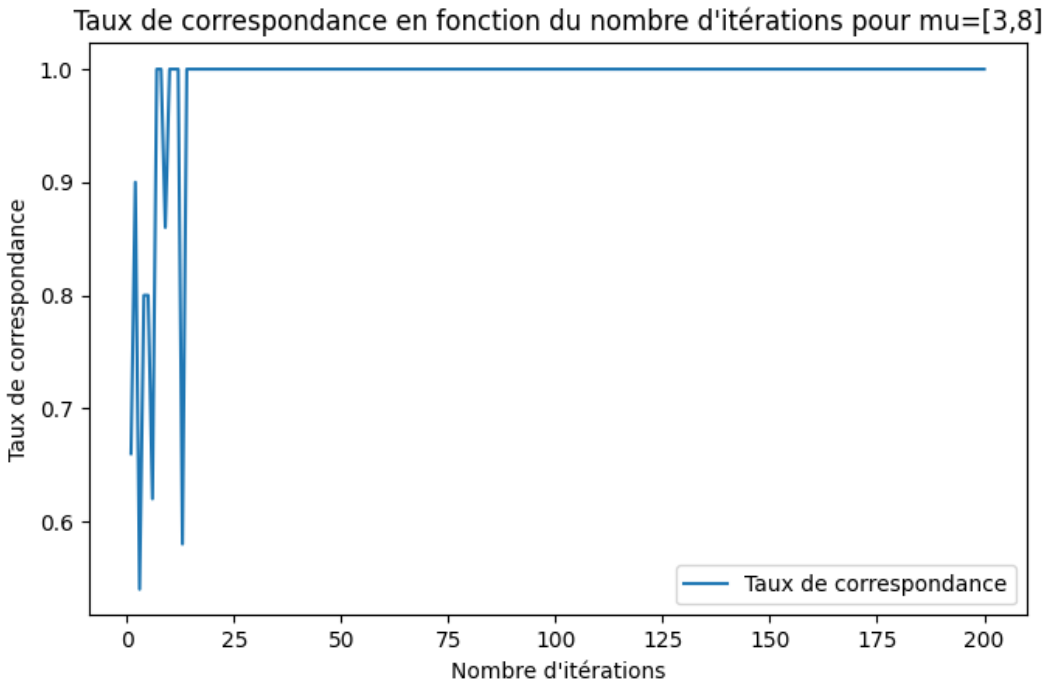
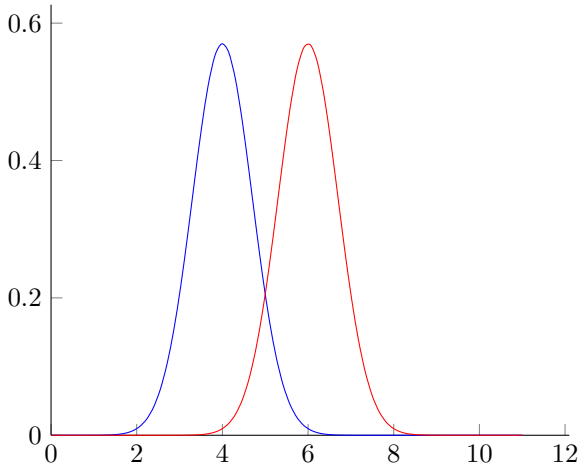


FIGURE 1 – Cas des Gaussiennes avec des moyennes distinctes



Les gaussiennes se chevauchent légèrement, on s'attend à ce que la classification soit un peu moins bonne que pour le cas précédent.

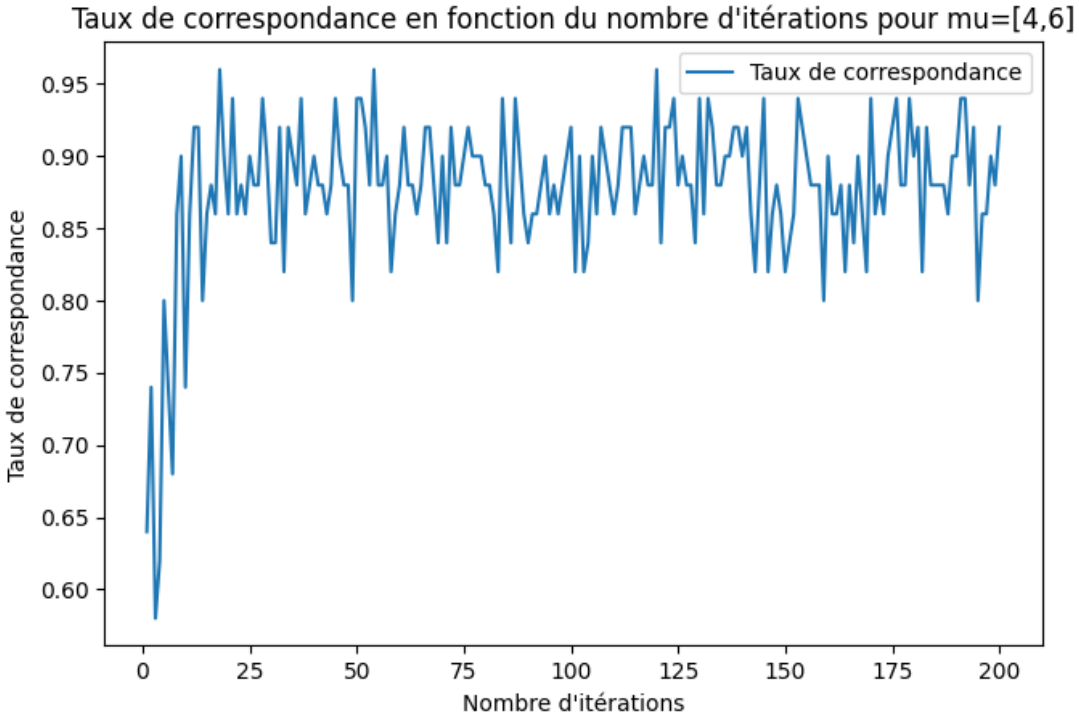
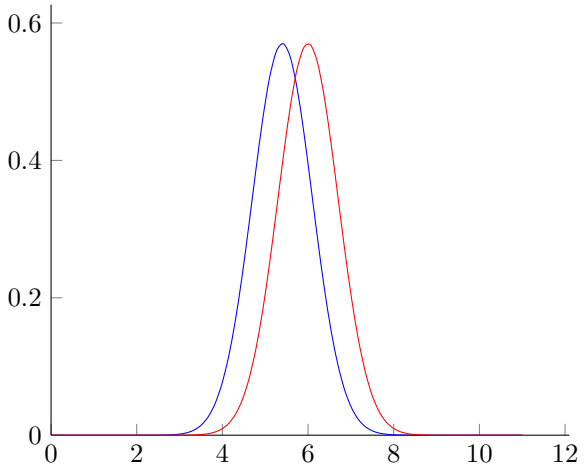


FIGURE 2 – Cas des Gaussiennes avec des moyennes rapprochées

La *Figure 2* montre une importante variation du taux de correspondance au fil des itérations. La correspondance fluctue principalement entre 0.8 et 0.95 avec un bruit très élevé. Les centres des gaussiennes étant relativement proches (moyennes à 3 et 8) comme évoqué précédemment, cela rend la séparation des clusters plus difficile pour l'algorithme ICE. La forte variation témoigne des difficultés à converger vers une solution stable. Cela est due à une confusion entre les deux clusters du fait de leur proximité.

Pour $\mu_0 = 5.4$, $\mu_1 = 6$, $\sigma_0 = \sigma_1 = 0.7$:



On arrive à un cas très délicat, les gaussiennes se confondent presque, on s'attend alors à avoir un taux de bonne classification assez mauvais.

En effet, dans le cas de la *Figure 3*, le taux de correspondance oscille entre 0.5 et 0.75, montrant une variabilité importante tout au long des itérations mais surtout un taux de correspondance très bas comparé aux autres cas de figure. On a des centres de gaussienne très proches, ce qui rend très difficile la distinction des clusters. L'algorithme a du mal à converger car les clusters sont presque indiscernables.

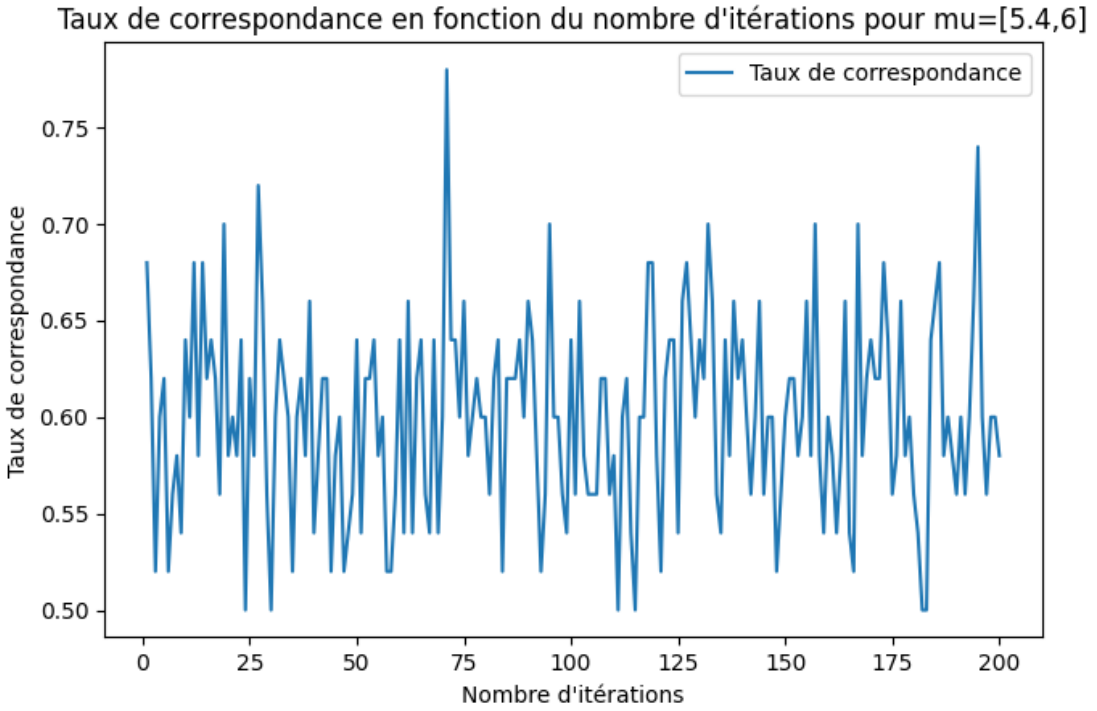


FIGURE 3 – Cas des Gaussiennes avec des moyennes très proches

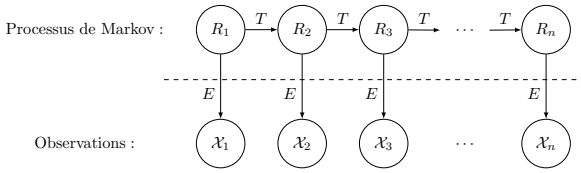
Comme le graphique est assez difficile à lire car il ne converge pas vraiment, nous allons utiliser la méthode de Monte Carlo sur 200 échantillons à 150 itérations pour mieux estimer les performances de l'algorithme. Le taux de correspondance moyen est donc de 0.60 qui est assez faible, cela revient presque à tirer à pile ou face les $\hat{R}_i$.

3 Algorithme ICE dans le cas d'une chaine de Markov Cachée

3.1 Modélisation mathématiques

3.1.1 Définition d'une chaine de Markov

Partons tout d'abord d'une représentation graphique d'une chaîne de Markov cachée pour pouvoir définir les notations qui nous seront utiles tout au long de ce travail.



Un modèle est Markovien si et seulement si son état à un instant i dépend uniquement de son état à l'instant $i-1$. En particulier, on parle de modèle de Markov caché *(HMM)* si à tout instant, on ne peut observer l'état dans lequel se trouve le modèle.

Nous effectuons $n \in \mathbb{N}^*$ observations. Ces observations sont les $\{X_i\}_{i\in[\![1;n]\!]}$, elles sont "tirées" selon une loi normale, conditionnellement aux états cachés du modèle HMM que nous notons $\{R_i\}_{i\in[\![1;n]\!]}$. Dans la suite, on considère une chaine de Markov avec 2 états cachés R possibles : $R = 0$ et $R = 1$. Les R_i sont eux tirés conditionnellement à R_{i-1} avec des probabilités constantes comme le montre ce graphe d'une chaine de Markov (ici les sources sont les X et on s'intéresse aux probabilités $a_{i,j}$) :

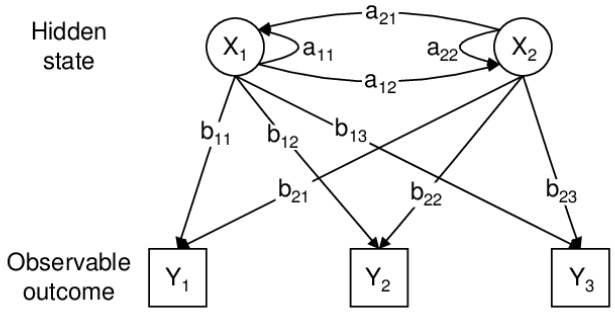


FIGURE 4 – Autre réprésentation d'une chaine de Markov

Lors de l'application de l'algorithme ICE dans le cas d'observations x_i générées par une chaine de Markov cachée, le problème principal est le calcul des probabilités à postériori $P(r_i|x_i)$ car en effet, les R_i ne sont plus indépendants comme dans le cas précédent. Ainsi, il nous faut un outil permettant de calculer ces probabilités et cet outil est l'algorithme forward backward, que nous allons détailler dans la prochaine partie.

3.1.2 Algorithme Forward Backward

Dans cette section, nous introduisons l'algorithme forward-backward normalisé que nous allons utiliser plus tard afin de calculer les probabilités à postériori dans le cas des chaines de Markov cachées. L'algorithme comprend deux étapes principales : l'étape forward et l'étape backward. Nous utilisons ici une version normalisé de l'algorithme pour éviter les valeurs très petites qui pourraient être moins bien approximées par python.

3.1.3 Partie Forward

Avec n états cachés, on définit $\alpha_i(u)$ qui représente la probabilité normalisée d'être dans l'état u à l'instant i et d'observer la séquence $x_1, x_2, ..., x_i$ jusqu'à l'instant i, ie. $\alpha_i(u) = P(R_i = u|x_{1,..,i})$. Les expressions des $\alpha_i(u)$ normalisés sont :

$$\alpha_1(u) = \frac{P(R_1 = u)P(X_1 = x_1|R_1 = u)}{P(R_1 = 1)P(X_1 = x_1|R_1 = 1) + P(R_1 = 0)P(X_1 = x_1|R_1 = 0)}$$

$$\alpha_i(u) = \frac{P(X_i = x_i|R_i = u)(\alpha_{i-1}(1)P(R_i = u|R_{i-1} = 1) + \alpha_{i-1}(0)P(R_i = u|R_{i-1} = 0))}{\sum_{k=0}^{1}\sum_{l=0}^{1} P(X_i = x_i|R_i = l)\alpha_{i-1}(k)P(R_i = l)|R_{i-1} = k)}, \quad 1 < i \leq n$$

3.1.4 Partie Backward

Avec N états cachés, on définit $\beta_i(u)$ qui représente la probabilité normalisée d'observer la séquence $x_{i+1}, x_{i+2}, ..., x_n$ à partir de l'état u à l'instant i, ie. $\beta_i(u) = \frac{P(x_{i+1,..,n}|R_i = u)}{P(x_{i+1,..,n}|x_{1,..,i})}$. Les expressions des $\beta_i(u)$ normalisés sont :

$$\beta_n(u) = 1$$

$$\beta_i(u) = \frac{\sum_{j=0}^{1} P(R_{i+1} = j|R_i = u)P(X_{i+1} = x_{i+1}|R_{i+1} = j)\beta_{i+1}(u)}{\sum_{k=0}^{1}\sum_{l=0}^{1} P(X_{i+1} = x_{i+1}|R_{i+1} = l)\alpha_i(k)P(R_{i+1} = l|R_i = k)}, \quad 1 \leq i < n$$

3.1.5 Calcul des probabilités à postériori

Une fois que les passes forward et backward sont terminées, les probabilités à postériori des états cachés peuvent être calculées en utilisant les quantités $\alpha_i(u)$ et $\beta_i(u)$. Les probabilités à postériori sont données par $\gamma_i(u)$ et $\psi_i(u,v)$ tels que :

$$\gamma_i(u) = \alpha_i(u)\beta_i(u)$$

$$\psi_i(u,v) = \frac{\alpha_i(u)P(R_{i+1}=v|R_i=u)P(X_{i+1}=x_{i+1}|R_{i+1}=v)\beta_{i+1}(v)}{\sum_{k=0}^{1}\sum_{l=0}^{1}\alpha_i(k)P(R_{i+1}=l|R_i=k)P(X_{i+1}=x_{i+1}|R_{i+1}=l)\beta_{i+1}(l)}$$

avec :
$\gamma_i(u) = P(R_i = u|x_{1,..,n})$

$\psi_i(u,v) = P(R_{i+1} = u, R_i = v|x_{1,..,n})$

3.1.6 Algorithme ICE

Dans le cas où les observations sont générées par une chaine de Markov cachée, nous avons besoin d'un nouveau θ car en effet, l'algorithme forward backward demande l'accés aux probabilités $P(R_{i+1}|R_i)$ qui seront dans ce modèle estimés. On note alors P la matrice de transition définie telle que Pour $(u,v) \in \{0,1\}^2, P_{u,v} = P(R_{i+1} = u|R_i = v)$.[1] On aura besoin également des probabilités d'être, au départ à $i = 0$, dans l'état $R = 0$ ou l'état $R = 1$. On note alors π le vecteur de probabilité de départ : $\pi = P(R_0 = 0)$. On a alors $\theta = (\pi, \mathbf{P}, \boldsymbol{\mu_0}, \boldsymbol{\mu_1}, \boldsymbol{\sigma_0}, \boldsymbol{\sigma_1})$

L'initialisation de θ_0 pour les composantes μ et σ se fait comme dans le cas iid. Pour π et P, on initialise toutes les probabilités à 0.5

Supposons maintenant que nous avons accès à une estimation de $R = (R_1, ..., R_n)$. On se place à la k-ième itération, nous avons donc $\theta_{\mathbf{k}} = (\mathbf{P_k}, \pi_{\mathbf{k}}, \boldsymbol{\mu_{0,k}}, \boldsymbol{\mu_{1,k}}, \boldsymbol{\sigma_{0,k}}, \boldsymbol{\sigma_{1,k}})$.

Nous allons calculer $\forall i \in \{1,..,n\}, P(X_i = x_i|R_i = 0)$, et comme X suit une loi gaussienne conditionnellement à $R = 0$:

$$P(X_i = x_i|R_i = 0) = \frac{1}{2\pi\sigma_{0,k}^2}\exp\left(-\frac{(x_i-\mu_{0,k})^2}{2\sigma_{0,k}^2}\right)$$

On effectue un calcul similaire pour trouver également $\forall i \in \{1,..,n\}, P(X_i = x_i|R_i = 1)$. Avec ces résultats nous pouvons alors utiliser l'algorithme forward backward détaillé dans les sections précédentes, l'utilisation de cet algorithme nécessite alors P_k, π_k et les vraisemblances de x sachant r calculé juste avant. On utilise alors les résultats vu dans la partie 3.1.5. pour avoir accès à $P(r_{n+1}, r_n|x_{1,..,n})$ et à $P(r_n|x_{1,..,n})$

On peut alors estimer de manière empirique la matrice de transition selon cette formule :

$$P(R_{i+1} = 0|R_i = v) = \frac{\sum_{k=1}^{n} P(R_{k+1} = 0, R_k = v|x_{1,..,n})}{\sum_{k=1}^{n} P(R_k = v|x_{1,..,n})}$$

et

$$P(R_{i+1} = 1|R_i = v) = 1 - P(R_{i+1} = 0|R_i = v)$$

En faisant le calcul pour $v = 0$ et $v = 1$, on arrive à estimer entièrement P_{k+1}

On estime ensuite la valeur de π_{k+1} par :

$$\pi_{k+1} = \gamma_0(0) = P(R_0 = 0|x_{1,..,n})$$

Nous avons donc déterminé entièrement les paramètres markovien P_{k+1} et π_{k+1}.

Contrairement au cas iid de la section 2. On simule les n réalisations du vecteur $R = (R_1, ..., R_n)$ selon sa loi conditionnelle à tous les $x_i, 0 < i \leq n$. Ainsi, on tire les r_i^j selon une loi de Bernoulli de paramètre $1 - P(R_i = 0|x_{1,..,n}) = 1 - \gamma_i(0)$. Puis on calcule μ et σ comme dans le cas iid.

Nous avons donc bien $\theta_{\mathbf{k+1}}$

1. NB. ces probabilités ne dépendent pas en fait pas de i comme vu dans la partie 3.1.1

3.2 Modélisation informatique

3.2.1 Notations

Nous allons définir quelques notations qui nous serviront pour le modèle informatique.

r : Ce sont les sources, ici r est un vecteur et r_i vaut 0 ou 1.

x : Ce sont les observations générés par les sources r_i.
La loi de X conditionnellement à $r = 0$ est une loi normale de moyenne μ_0 and d'écart type σ_0 :

$$\sim \mathcal{N}(\mu_0, \sigma_0^2)$$

De même, la loi de X conditionnellement à $r = 1$ est une loi normale de moyenne μ_1 and d'écart type σ_1 :

$$\sim \mathcal{N}(\mu_1, \sigma_1^2)$$

mu_hat ($\hat{\mu}$) : C'est l'estimation de μ.

sigma_hat ($\hat{\sigma}$) : C'est l'estimation de σ.

Phat ($\hat{P}$) : C'est l'estimation de la matrice de transition :

$$P_{transition} = \begin{bmatrix} P(\mathrm{R_{i+1}} = 0 | R_i = 0) & P(\mathrm{R_{i+1}} = 0 | R_i = 1) \\ P(\mathrm{R_{i+1}} = 1 | R_i = 0) & P(\mathrm{R_{i+1}} = 1 | R_i = 1) \end{bmatrix}$$

Pxr ($P(\mathrm{x}|r)$) : La probabilité de x conditionnellement à r.

pihat ($\hat{\pi}$) : C'est un vecteur représentant les probabilités d'être, au départ, dans l'état $r = 0$ ou l'état $r = 1$. On a donc : $\hat{\pi}_0 = P(r_0 = 0)$ et $\hat{\pi}_1 = P(r_0 = 1)$

3.2.2 Générer les données pour le cas Markov

Pour pouvoir utiliser cette algorithme, il nous faut donc d'abord des données d'entrée. Cette partie détaille le processus de génération des données dans le cas Markovien. Il nous faut donc générer des sources r_i selon un modèle de Markov, puis tirer des observations x_i selon les sources générées précédemment. Pour cela, nous allons initialiser une liste avec comme premier élèment r_0, on considère que : $P(R_0 = 0) = \hat{\pi}_0$ et $P(R_0 = 1) = \hat{\pi}_1$. Ensuite, on traite 2 cas :

- Si $R_i = 0$, alors on tire r_{i+1} avec les probabilités suivants : $P(R_{i+1} = 0) = \hat{P}_{0,0}$ et $P(R_{i+1} = 1) = \hat{P}_{1,0}$
- Sinon, $R_i = 1$, et on tire R_{i+1} avec les probabilités suivants : $P(R_{i+1} = 0) = \hat{P}_{0,1}$ et $P(R_{i+1} = 1) = \hat{P}_{1,1}$

Avec ce procédé, nous avons donc nos sources. Maintenant pour générer les observations, on parcourt la liste contenant les sources. Pour une observation générée par la source $r_i = j$, on tire x_i selon $\sim \mathcal{N}(\mu_j, \sigma_j^2)$. Nous avons donc maintenant toutes nos sources ainsi que nos observations.

Le code en python pour la génération des données est donc le suivant :

```
1  def generer_donnees_Markov(n,mu,sigma,pi,P_hat) :
2
3      r = np.random.rand(n)
4      r[0] = r[0] > pi
5      x = np.zeros(n)
6      x[0] = rd.normalvariate(mu[int(r[0])],sigma[int(r[0])])
7      for i in range(1,n) : #on met r à 0 ou à 1 en fonction de P et on tire les observations
8          r[i] = r[i] > P_hat[0][int(r[i-1])]
9          x[i] = rd.normalvariate(mu[int(r[i])],sigma[int(r[i])])
10     return r, x
```

3.2.3 Quelques simulations

On fixe aussi le nombre d'observations à $n = 50$ et $\sigma_1 = \sigma_2 = 0.7$

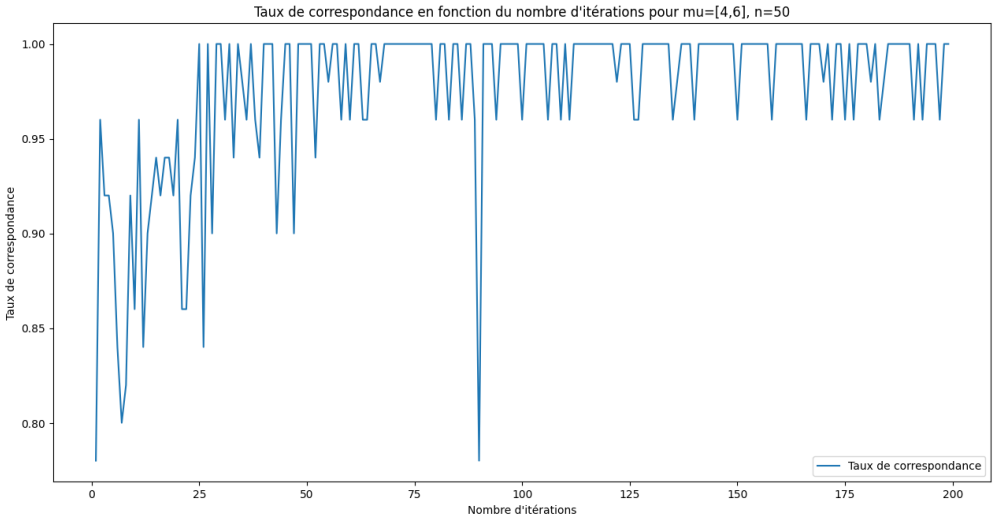


FIGURE 5 – Gaussiennes rapprochées, Markov

La *Figure 5* représente le taux de correspondance en fonction du nombre d'itérations de l'algorithme ICE pour le cas Markov. On remarque qu'après environ 25 itérations, le taux de correspondance commence à se stabiliser autour de 0.95, avec des fluctuations mineures. On a donc un algorithme qui est très efficace pour trouver une correspondance élevée pour des chaînes de Markov cachées après un nombre suffisant d'itérations. On observe quand même des chutes notables à quelques reprises ce qui est pourrait s'expliquer par une tentative d'exploration nécessaire des paramètres de l'algorithme. Mais on a globalement une convergence vers des solutions optimales.

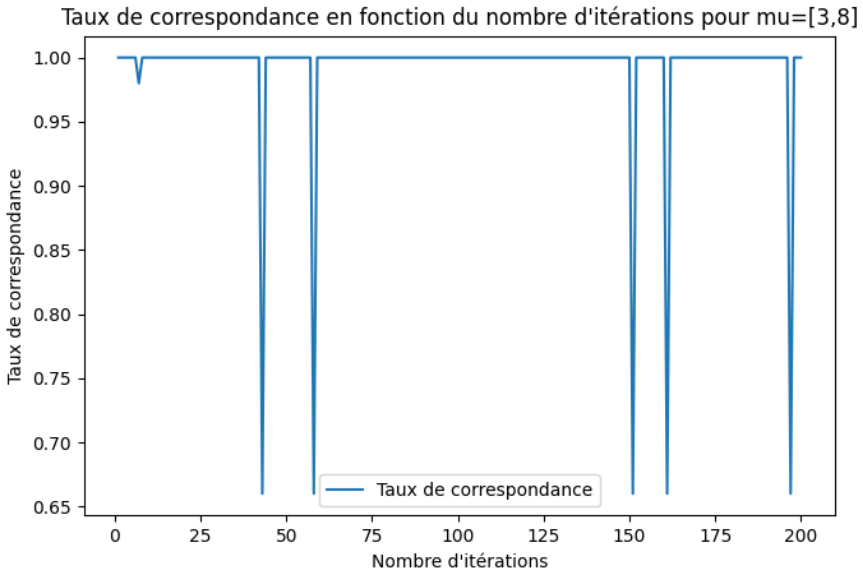


FIGURE 6 – Gaussiennes éloignées, Markov

La *Figure 6* modélise le cas ou les gaussiennes sont très éloignées pour le cas Markov. Dans ce cas, le taux de correspondance est très stable et rapide autour de 1.0, on a donc bien convergence or pour

certaines valeures particulières où le programme effectue des divisions par 0 car σ converge vers 0. On s'attendait à un tel résultat car les gaussiennes sont éloignées. On a cependant, des chutes brusques aux environs de 10, 50, 150 et 200 itérations qui peuvent être dues à des ajustement majeurs ou des recalculs dans les paramètres de l'algorithme, puis celui-ci revient ensuite à la solution optimale.

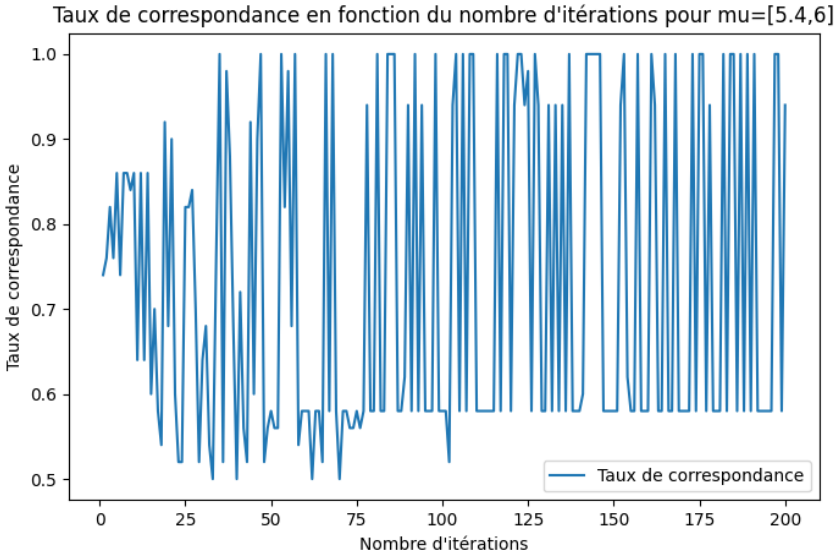


FIGURE 7 – Gaussiennes très rapprochées, Markov

La *Figure 7* modélise le cas ou les gaussiennes sont très proches, presque confondues pour le cas Markov. On a un taux très instable qui oscille entre 0.5 et 1.0. Les fluctuations suggèrent que l'algorithme explore l'espace de solutions sans trouver un résultats stable. Cela est dû au chevauchement des gaussiennes causé par les moyennes très proches.

Il est également possible d'utiliser la loi des grands nombres pour regarder le taux de correspondance moyen pour 150 itérations. On trouve alors 0.85933

4 Utilisation de l'algorithme ICE

4.1 Traitement d'image

L'algorithme ICE trouve un de ses cas d'utilisation principal en traitement d'image. Il y est souvent utilisé pour restaurer des images bruitées. B. BENMILOUD et W. PIECZYNSKI détaillent dans leur article de 1995 [1] cette utilisation de l'algorithme ICE dans le cas Markovien. Ils y montrent ses bonnes performances en comparaison à d'autres algorithmes tels que le Stochastic Estimation Maximisation ou le Expectation-Maximisation.

Les figures suivantes sont tirées de l'article de B. BENMILOUD et W. PIECZYNSKI.

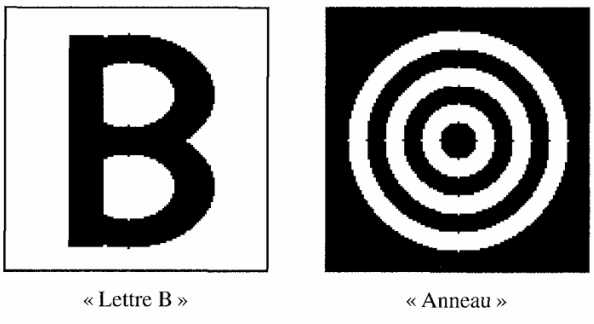


FIGURE 8 – Images d'origines

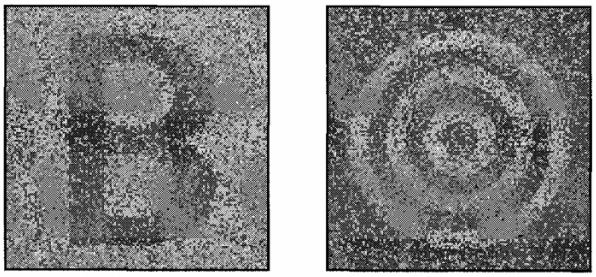


FIGURE 9 – Images bruitées

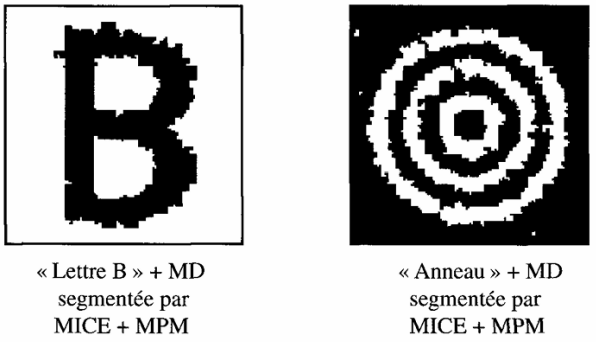


FIGURE 10 – Images restaurées par l'algorithme ICE

Ils développent aussi l'intérêt de modéliser une image par une chaîne de Markov. Pour ce faire ils introduisent le parcours de Hilbert-Peano.

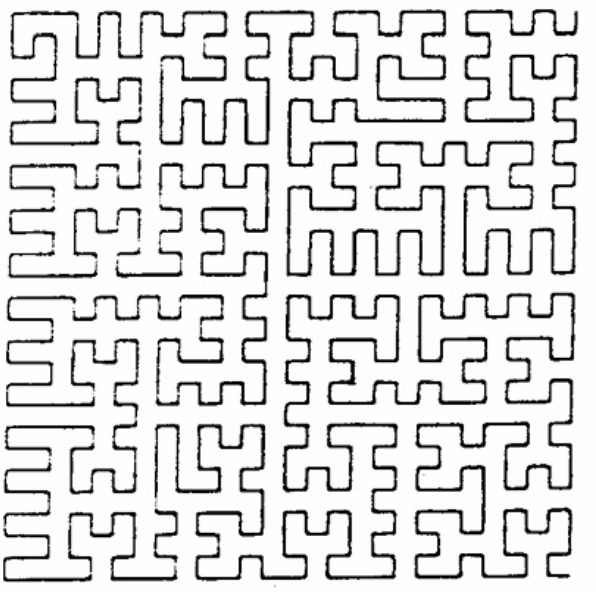


FIGURE 11 – Tracé du parcours Hilbert-Peano d'une image

Ce parcours particulier *(Figure 11)* de l'image permet de mieux rendre compte de la proximité des pixels dans leur modélisation par une chaîne. Un parcours classique ligne à ligne éloignerait démesurément 2 pixels l'un en-dessous de l'autre, par exemple. En adoptant celui de Hilbert-Peanon, il est donc possible de mieux rendre compte de l'agencement des pixels sur l'image et ainsi d'aboutir à une meilleure segmentation.

4.2 Traitement du signal

L'algorithme ICE peut aussi servir dans le domaine du traitement du signal. L'utilisation des nouvelles technologies de communication est conditionnées par notre capacité à garder "l'intégrité" des signaux envoyés. La dégradation des ondes, surtout dans un contexte mobile, empêche la bonne compréhension des canaux émetteurs par le récepteur. Un article de 2022 de Rabah Ouchikh et al. reprend un équivalent de l'algorithme ICE pour estimer les canaux dans une technologie utilisant l'OFTS (Orthogonal Time Frequency Space), le remplaçant de l'OFDM.

4.3 Note sur la convergence de ICE

Il est à noter que l'utilisation de l'algorithme ICE est limité par le manque de certitudes sur sa convergence. Peux de ressources sont en effet disponibles à ce sujet, empêchant probablement la mise en place effective de cet algorithme dans un contexte industriel.

En 2008, W. PIECZYNSKI propose tout de même une preuve de la convergence de l'algorithme dans le cas d'estimation de proportions dans un mélange de lois multi-variées.

5 Bibliographie

- Champs de Markov cachés et estimation conditionnelle itérative, Wojciech PIECZYNSKI, 1994
- Pattern Recognition and Machine Learning, Christopher M. Bishop, 2006
- Estimation des paramètres dans les chaînes de Markov cachées et segmentation d'images, Btissam BENMILOUD et Wojciech PIECZYNSK, 1995
- Estimation de canal pour les systèmes OTFS, Rabah Ouchikh, Thierry Chonavel, Abdeldjalil Aissa El Bey, Mustapha Djeddou, 2022

6 Annexes

6.1 Démonstrations

6.1.1 Probabilités à Postériori de l'algorithme Forward-Backward Normalisé

Montrons que :

$$\gamma_i(u) = \alpha_i(u)\beta_i(u) = P(R_i = u|x_{1,..,n})$$

Tout d'abord,

$$\gamma_i(u) = \alpha_i(u)\beta_i(u) = \frac{P(x_{i+1,..,n}|R_i = u)}{P(x_{i+1,..,n}|x_{1,..,i})} P(R_i = u|x_{1,..,i})$$

D'après la formule de Bayes et par simplification :

$$\gamma_i(u) = \frac{P(x_{i+1,..,n}|R_i = u)P(x_{1,..,i}|R_i = u)P(R_i = u)}{P(x_{1,..,n})}$$

Par indépendance conditionnelle des $X_{i+1,..,n}$ et $X_{1,..,i}$ sachant R_i, on en déduit l'expression suivante :

$$\gamma_i(u) = \frac{P(x_{1,..,n}|R_i = u)P(R_i = u)}{P(x_{1,..,n})}$$

Ainsi,

$$\gamma_{\mathbf{i}}(\mathbf{u}) = \mathbf{P}(\mathbf{R_i} = \mathbf{u}|\mathbf{x_{1,..,n}})$$

6.2 Code en python

6.2.1 Algorithme ICE (cas iid)

```python
# Algorithme ICE dans le cas gaussien

'''
Attention les notations sont inversées par rapport aux articles de W. PIECZYNSKI.
Nous nous sommes basé sur les notations de M. CASTELLA.
Notamment R=(r1,...,rn) représente ici les états cachés tandis que X=(x1,...,xn)
correspond aux observations.
'''
import numpy as np
import random as rd

def generer_donnees(n,mu,sigma,alpha) :
    ''' Fonction de génération des données
    Parametres
    ----------
    n = taille de l'échantillon
    mu = matrice 1x2 composée de mu0 et mu1 les moyennes des lois conditionnelles
    à x=0 et x=1 resp.
    sigma = matrice 1x2 composée de sig0 et sig1 les ecarts-types des lois conditionnelles
    à x=0 et x=1 resp.
    alpha = proba que ri=0

    Return
    ------
    r = matrice de taille n des états cachés
    x = matrice de taille n des observations
    '''
    r = np.random.rand(n) > alpha

```

```
30      x = np.zeros(n)
31      for i in range(n) : #on tire les observations en fonction de r
32          x[i] = rd.normalvariate(mu[int(r[i])],sigma[int(r[i])])
33      return r, x
34
35  def tirage(n,prx,mu,sigma):
36      ''' Fonction tirant des sets de X, utilisée dans les itérations de l'algo ICE pour
37      estimer mu et sigma
38      Parametres
39      ----------
40      n = taille de l'échantillon
41      prx = matrice de taille n indiquant les probas ri=0 sachant x
42      mu = matrice 2x1 composée de mu0 et mu1 les moyennes des lois conditionnelles
43      à x=0 et x=1 resp.
44      sigma = matrice 2x1 composée de sig0 et sig1 les ecarts-types des lois conditionnelles
45      à x=0 et x=1 resp.
46
47      Return
48      ------
49      r = matrice de taille n des états cachés supposés
50      '''
51      r = np.random.rand(n) > prx
52      return r
53
54  def ice(n,X,q,N):
55      ''' Fonction de classification itérative
56      Parametres
57      ----------
58      n = taille de l'échantillon
59      X = matrice de taille n des observations
60      q = nombres d'itérations globales de l'algo ICE
61      N = nombre de tirages de X à faire à chaque itération
62
63      Return
64      ------
65      mu = matrice 2x1 composée de mu0 et mu1 les moyennes des lois conditionnelles
66      à x=0 et x=1 resp.
67      sigma = matrice 2x1 composée de sig0 et sig1 les ecarts-types des lois conditionnelles
68      à x=0 et x=1 resp.
69      '''
70      #initialisation
71      R_tirage = np.zeros((N,n)) # représente les N tirages qui seront fait à chaque itération
72      R_hat = np.random.binomial(1,0.5,n) #On tire un premier set d'états cachés
73
74      U0 = R_hat.sum() #U0 et les V sont calculés comme définis dans notre formalisation du pbm
75      V0 = ((1-R_hat) * X).sum()
76      V1 = ((R_hat) * X).sum()
77
78      alpha_hat = 0.5 #proba que Ri=0
79      mu_hat = np.array([V0/U0,V1/(n-U0)])
80
81      S0=0
82      S1=0
83      S0 = np.sum((1-R_hat)*(X - mu_hat[0]) ** 2)
84      S1 = np.sum((R_hat)*(X - mu_hat[1]) ** 2)
85
86      sigma_hat = np.array([np.sqrt(S0/U0),np.sqrt(S1/(n-U0))])
87
```

```
    Pxr = np.empty((2, n)) #proba de x sachant r
    Px_inter_r = np.empty((2, n)) #proba de x inter r
    Prx = np.empty((n,)) #proba de x sachant r

    for _ in range (q):
        print((2*np.pi*sigma_hat[0]**2))
        #On évalue les proba de xi sachant ri=0 et sachant ri=1
        Pxr[0,:]=(1/(2*np.pi*sigma_hat[0]**2))*np.exp(-(X-mu_hat[0])**2/(2*sigma_hat[0]**2))
        Pxr[1,:]=(1/(2*np.pi*sigma_hat[1]**2))*np.exp(-(X-mu_hat[1])**2/(2*sigma_hat[1]**2))

        #On évalue les proba de (xi inter ri=0) et (xi inter ri=1)
        Px_inter_r [0,:] = Pxr[0,:] * alpha_hat
        Px_inter_r [1,:] = Pxr[1,:] * alpha_hat

        #On évalue la proba que ri=0 sachant xi
        Prx = Px_inter_r[0,:]/(Px_inter_r[0,:]+Px_inter_r[1,:])

        #ré-estimation alpha
        alpha_hat = (Prx.sum())/n

        #on effectue N tirage de R
        for j in range(N) :
            R_tirage[j]= tirage(n,Prx,mu_hat,sigma_hat)


        #ce qui suit est très moche mais on optimisera plus tard...
        #ré-estimation de mu et sigma par tirage de N sets de R et calcul empirique
        U0=0
        V0=0
        V1=0
        #calcul de mu
        mu0_hat_tirage = np.zeros(N)
        mu1_hat_tirage = np.zeros(N)
        for j in range(N) :
            U0 = np.sum(1-R_tirage[j])
            V0 = np.sum((1-R_tirage[j])*X)
            V1 = np.sum(R_tirage[j]*X)
            mu0_hat_tirage[j] = V0/U0
            mu1_hat_tirage[j] = V1/(n-U0)

        mu_hat[0] = mu0_hat_tirage.sum()/N
        mu_hat[1] = mu1_hat_tirage.sum()/N

        S0 = 0
        S1 = 0
        #calcul de sigma
        sig0_hat_tirage = np.zeros(N)
        sig1_hat_tirage = np.zeros(N)
        for j in range(N):
            U0 = np.sum(1-R_tirage[j])
            S0 = np.sum((1-R_tirage[j])*(X - mu_hat[0]) ** 2)
            S1 = np.sum((R_tirage[j])*(X - mu_hat[1]) ** 2)
            sig0_hat_tirage[j] = np.sqrt(S0/(U0))
            sig1_hat_tirage[j] = np.sqrt(S1/(n-U0))


        sigma_hat[0] = sig0_hat_tirage.sum()/N
        sigma_hat[1] = sig1_hat_tirage.sum()/N
```

```
    return mu_hat,sigma_hat, alpha_hat, tirage(n,Prx,mu_hat, sigma_hat)


if __name__ == '__main__':

    #paramètres du modèles
    mu = [4,7]
    sigma = [0.3,4]
    alpha = 0.7

    #Generation des datas
    n = 20 #nombres d'observations
    r,x = generer_donnees(n,mu,sigma,alpha)

    #Utilisation de l'algo ICE
    q=100 #nombre d'itérations de l'algo
    N=5 #nombre de tirages effectués à chaque itération
    mu_hat,sigma_hat, alpha_hat, r_hat = ice(n,x,q,N)

    print("L'algo ICE renvoie comme paramètres")
    print(f"alpha : {alpha_hat}")
    print(f"mu : {mu_hat}")
    print(f"sigma : {sigma_hat}")
    print(f"taux de correspondance entre les séquences estimées et réelles: {(r_hat== r).sum()/n}")
```

6.2.2 Algorithme ICE (cas Markov)

```
import numpy as np
import random as rd


def forward_backward(P_hat,pi_hat,Pxr,nEtats,n):
    #Initialisation
    alpha = np.zeros((nEtats, n))
    beta = np.zeros((nEtats, n))

    #Partie Foward
    alpha[:, 0] = pi_hat * Pxr[:, 0]
    alpha[:, 0] = alpha[:, 0] / np.sum(alpha[:, 0])
    for t in range(1, n):
        alpha[:, t] = (alpha[:, t-1] @ P_hat) * Pxr[:, t]
        alpha[:, t] = alpha[:, t] / np.sum(alpha[:, t])

    #Partie Backward
    beta[:, n-1] = [1, 1]
    for t in range(n-1, 0, -1):
        beta[:, t-1] = P_hat @ (beta[:, t] * Pxr[:, t])
        coeffrenorm = alpha[:, t-1].T @ P_hat @ Pxr[:, t]
        beta[:, t-1] = beta[:, t-1] / coeffrenorm

    # Autres coefficients
    gamma = alpha * beta  #

    psi = np.zeros((nEtats, nEtats, n-1))
    for t in range(n-1):
        psi[:, :, t] = (alpha[:, t].reshape(-1, 1) * (Pxr[:, t+1] * beta[:, t+1])) * P_hat
        coeffrenorm = np.sum((alpha[:, t].reshape(-1, 1) * (Pxr[:, t+1] * beta[:, t+1])) * P_hat)
```

```
31          psi[:, :, t] = psi[:, :, t] / coeffrenorm
32
33      return gamma, psi
34
35  def ice(n,x,q,N,nEtats=2):
36      ''' Algorithme ICE pour la segmentation de séquences
37      '''
38
39      pi_hat_Ini = np.ones(nEtats) / nEtats
40      P_hat_Ini = 0.5 * np.eye(nEtats) + (np.ones((nEtats, nEtats))
41      Phat_Ini = Phat_Ini - np.eye(nEtats)) * 1 / (2 * (nEtats - 1))
42
43      mu_Ini = np.zeros(nEtats)
44
45      if nEtats % 2 == 0:
46          for i in range((nEtats + 1) // 2):
47              mu_Ini[i] = np.mean(x) - (((nEtats // 2) - (i)) * (np.std(x) / 2))
48              mu_Ini[nEtats - (i + 1)] = np.mean(x) + (((nEtats // 2) - (i)) * (np.std(x) / 2))
49      else:
50          for i in range(nEtats):
51              mu_Ini[i] = np.mean(x) - (((nEtats // 2) - (i)) * (np.std(x) / 2))
52
53      sigma_Ini = np.std(x)
54
55      mu_hat = mu_Ini
56      sigma_hat = np.tile(sigma_Ini, nEtats)
57      pi_hat = pi_hat_Ini
58      P_hat = P_hat_Ini
59
60      for ii in range(q):
61          # Calcul de la proba conditionnelle à état
62          Pxr = np.zeros((nEtats, n))
63          Pxr[0, :] = 1 / (sigma_hat[0] * np.sqrt(2 * np.pi))
64          Pxr[0, :] = Pxr[0, :] * np.exp(- (x - mu_hat[0]) ** 2 / (2 * sigma_hat[0] ** 2))
65          Pxr[1, :] = 1 / (sigma_hat[1] * np.sqrt(2 * np.pi))
66          Pxr[1, :] = Pxr[1, :] * np.exp(- (x - mu_hat[1]) ** 2 / (2 * sigma_hat[1] ** 2))
67
68          #Appel de la fonction forward_backward
69          gamma, psi = forward_backward(P_hat,pi_hat,Pxr,nEtats,n)
70
71          # Réestimation des paramètres
72          pihat = gamma[0][0]
73          P_hat = np.sum(psi, axis=2) / np.sum(gamma[:, :n-1], axis=1).reshape(-1, 1)
74
75          # Simulation des variables cachées selon la loi a posteriori
76          r_ice = np.zeros((N, n), dtype=int)
77          mutmp = np.zeros((N, nEtats))
78          sigmatmp = np.zeros((N, nEtats))
79          for n_ice in range(N):
80              r_ice[n_ice, 0] = np.random.rand() > gamma[0, 0]
81              for t in range(n-1):
82                  U = np.random.rand()
83                  r_ice[n_ice,t+1] = (U > gamma[0,t+1])
84              mutmp[n_ice, 0] = np.mean(x[np.where(r_ice[n_ice, :] == 0)])
85              mutmp[n_ice, 1] = np.mean(x[np.where(r_ice[n_ice, :] == 1)])
86              sigmatmp[n_ice, 0] = np.std(x[np.where(r_ice[n_ice, :] == 0)])
87              sigmatmp[n_ice, 1] = np.std(x[np.where(r_ice[n_ice, :] == 1)])
88
```

```
89          mu_hat = np.mean(mutmp, axis=0)
90          sigma_hat = np.mean(sigmatmp, axis=0)
91
92
93      return mu_hat, sigma_hat, P_hat, gamma
94
95  if __name__ == '__main__':
96
97      #paramètres du modèles
98      mu = [0,2]
99      sigma = [1,1]
100     pi = 0.5
101     P_hat = np.array([[0.7, 0.3], [0.7, 0.3]])
102
103     #Generation des datas
104     n = 500 #nombres d'observations
105     r,x = generer_donnees_Markov(n,mu,sigma,pi, P_hat)
106
107     #Utilisation de l'algo ICE
108     q=500 #nombre d'itérations de l'algo
109     N=1 #nombre de tirages effectués à chaque itération
110     mu_hat,sigma_hat, P_hat, gamma = ice(n,x,q,N,nEtats=2)
111
112     r_segm = r_ice[-1,:]
113     r_segm = r_segm - 1
114
115     print("L'algo ICE renvoie comme paramètres")
116     print(f"P : {P_hat}")
117     print(f"mu : {mu_hat}")
118     print(f"sigma : {sigma_hat}")
119     print('Taux bonne segmentation: ', np.sum(r == r_segm) / n)
```