École nationale
de la statistique
et de l'administration
économique

PROJET DE MODÉLISATION STATISTIQUE

Bioinformatique :

Segmentation bi-dimensionnelle pour l'analyse des données HiC

Auteurs :
Louisa CAMADINI
Quentin MENNER

Encadrants Professionnels :
Emilie LEBARBIER
Cristina BUTUCEA

MOTS CLÉS : *Segmentation, HiC, 1D, 2D, breakpoints, vraisemblance, paramètres*

Table des matières

1 Introduction

En biologie, l'étude de la conformation spatiale des chromosomes est essentielle à la compréhension de l'expression des gènes. La proximité des *loci* génomiques peut conduire à une interactions des gènes. Ces interactions ou "plis" au sein des chromosomes peuvent entraver, favoriser ou réguler l'expression de ceux-ci.

Différentes méthodes existent pour étudier l'organisation spatiale des chromosomes et l'interaction des *locis*. La technologie **HiC** est la plus récente de ces méthodes. En mesurant la fréquence à laquelle deux fragments d'ADN se lient au sein d'une cellule, celle-ci permet d'obtenir une matrice représentant ces interactions sur l'ensemble d'un chromosome.

On observe que le plus souvent, les interactions avec la plus grande intensité se forment dans des régions établies et homogènes (*"cis-interacting regions"*), ce qui se traduit de manière matricielle par l'observation de "blocs" autour de la diagonale.

L'objectif de l'étude est de déterminer une méthode pour mettre en évidence ces blocs diagonaux. Différentes méthodes ont déjà été envisagées pour étudier ces derniers, notamment à l'aide du modèle de Markov caché (*"Hidden Markov Model"*).

L'article *Two-dimensional segmentation for analysing HiC data* de *C. Lévy-Leduc, M. Delattre, T. Mary-Huard et S. Robin* propose ici une solution passant par la transformation du problème de segmentation 2D en un problème de segmentation 1D en s'appuyant sur les algorithmes de programmation dynamique pour déterminer ces régions par maximum de vraisemblance.

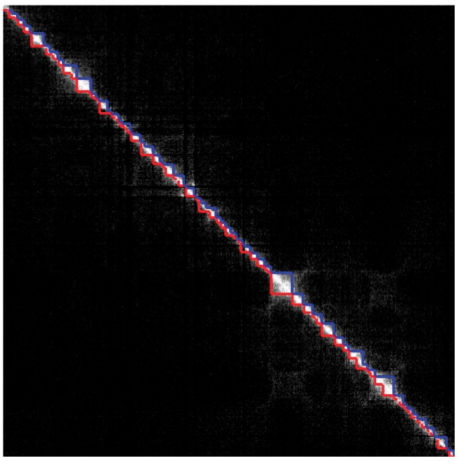


FIGURE 1 – Représentation matricielle de la méthode HiC. Les blocs correspondent aux *"cis-interacting regions"*.

Dans une première section, nous reviendrons sur la segmentation en une dimension dans un cadre paramétrique et *off-line* avec la méthode du maximum de vraisemblance ainsi que l'utilisation de la programmation dynamique pour déterminer les points de rupture optimaux.

Dans une deuxième section, nous verrons comment depuis notre problématique de segmentation de blocs en deux dimensions, nous pouvons en fait nous ramener à un problème de segmentation en une dimension et répondre ainsi à la problématique de l'article.

Enfin, dans une dernière partie, nous utiliserons le package R **HiCseg** mis à disposition par les auteurs de l'article afin de mettre la méthode en application sur un dataset, et tenter d'observer comment les résultats évoluent selon les paramètres.

2 Segmentation dans le cadre uni-dimensionnel

Tout d'abord, il faut définir la méthode de segmentation dans un cadre uni-dimensionnel. On recherche ici dans ces séries les points de rupture (« breakpoints ») qui sépareraient la série en plusieurs segments partageant un ou plusieurs paramètres communs en tenant compte du bruit présent dans le dataset. Le cadre supposé ici repose, comme dans l'article, sur une série *off-line* paramétrique et reposant sur de la segmentation où les paramètres sont spécifiques aux segments.

2.1 Présentation du cadre

La séquence est de la forme $y = y_1, \ldots, y_n$, modélisée par le processus aléatoire $Y = Y_1, \ldots, Y_n$ avec $Y_t \sim G(\theta_t, \phi)$ où ϕ est fixé. Le paramètre θ est différent dans chacun des K segments, séparés par les $K-1$ points de rupture $t_0 \leq t_1 ... \leq t_K$.
K prend ses valeurs dans $[\![1, n]\!]$.

On s'intéresse au cas où $Y_t \sim \mathcal{N}(\mu_k, \sigma^2)$ si $t \in r_k = [\![t_{k-1} + 1, t_k]\!]$. Le paramètre θ à déterminer pour chaque segment est $(\mu_k)_k$ si $t \in r_k = [\![t_{k-1} + 1, t_k]\!]$ pour $k = 1,...,K$.

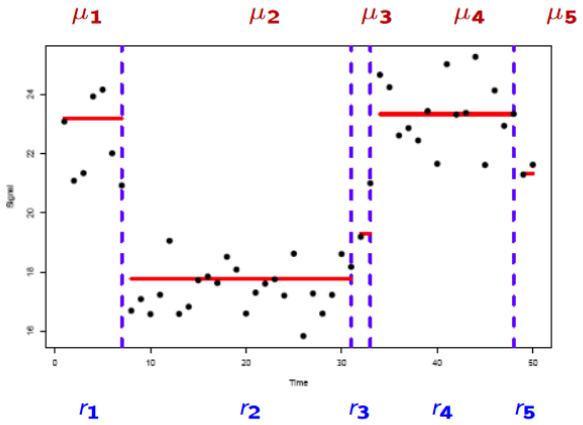


FIGURE 2 – Exemple d'application de la méthode de segmentation à une série temporelle

On doit donc déterminer 3 paramètres dans notre modèle :
- K, le nombre de segments
- $T = (t_1, t_2, ..., t_{K-1})$, les points de rupture
- $\mu = (\mu_1, ..., \mu_K)$, les paramètres de la distribution

Le but est donc d'estimer la valeur de μ_k pour chaque segment de la série.

La méthode de segmentation fonctionne donc en trois temps :

1. En supposant K et T connus, on estime μ.
2. En supposant K connu, on estime T.
3. On choisit K.

2.2 Méthode d'estimation

Pour estimer les paramètres de (T, μ) on utilise la méthode du maximum de vraisemblance sur les données observées.

$$\begin{aligned} p(y|K,T,\mu) &= \prod_{t=1}^{n} f(y_t;\mu,T) \\ &= \prod_{k=1}^{K} \prod_{t\in r_k} f(y_t;\mu_k) \end{aligned}$$

En 1, On estime μ_k par le maximum de vraisemblance : $\hat{\mu}_k = \underset{\mu_k\in\mathbb{R}}{\arg\min} \log p(y|K,T,\mu)$

En 2, On détermine l'ensemble T des points de rupture qui maximise la log-vraisemblance calculée à son maximum pour μ :

$$\hat{T} = \underset{T\in M_{K,n}}{\arg\max} p(y|K,T,\hat{\mu})$$

où $M_{K,n}$ est l'ensemble des segmentations possibles sur la grille $[\![1,n]\!]$ en K segments.

Pour trouver le meilleur ensemble T, nous utilisons la méthode de la programmation dynamique pour régler le problème du plus court chemin. Soit $C_k(i,j)$ est le coût du meilleur chemin connectant i à j en K segments. Le problème d'optimisation est donc :

$$C_{K,n} = C_K(1,n) = \min_{t_1,\ldots,t_{K-1}} \sum_{k=1}^{K} C_1(t_{k-1}+1, t_k)$$

Avec pour condition que les différents segments soient indépendants.

En 3, Grâce à l'étape précédente, on obtient les meilleures segmentations pour différentes valeurs de $K = 1, ..., n$. Cependant, le risque dans notre modèle uni-dimensionnel est d'avoir un K trop élevé donc un trop grand nombre de segments, réduisant ainsi l'intérêt du modèle de segmentation.

Il est donc nécessaire d'appliquer une forme de pénalité sur la valeur de K.

Le modèle est alors de la forme :

$$\hat{K} = \underset{K}{\arg\min}\{-logp(y|K,\hat{T},\hat{\mu}) + pen(K)\}$$

Les deux principaux modèles de pénalité sont :

- AIC : $pen(K) = K$
- BIC : $\frac{K\log(n)}{2}$

3 D'une à deux dimensions

Nous cherchons ici à étudier la conformation spatiale des chromosomes. La motivation repose sur le fait qu'un gène ne pourra être exprimé que si l'ADN qui le code est déplié. Par conséquent, les régions dans lesquelles les gènes ne sont pas exprimés sont quant à elles repliées et forment ce qu'on appelle les domaines topologiques. La méthode **HiC** (High Chromosome Contact) permet d'étudier ces domaines et leurs interactions. En effet, la technologie **HiC** permet d'évaluer la proximité entre toute paire de *loci* le long du génome. Elle aboutit à une matrice de données où apparaissent des blocs correspondant aux régions en interaction (et auto-interaction). La délimitation de ces blocs permet de comprendre l'organisation spatiale de la chromatine. Il en résulte donc un problème computationnel en 2D, que nous allons présenter avant de spécifier les principales différences avec la segmentation 1D présentée précédemment.

Le problème général de la segmentation est le suivant : il s'agit de minimiser la log-vraisemblance sur trois paramètres à hiérarchiser :

$$\arg\min_{K,T,\mu}\{-logp(y|K,T,\mu) + pen(K)\} \tag{1}$$

Où le terme $pen(K)$ est utile au choix du nombre de segments. On cherche donc la meilleure segmentation en K segments.

Ici, il s'agit de segmenter une matrice $(Y_{i,j})_{1\leq i\leq j\leq n}$, où $Y_{i,j}$ est l'intensité de l'intéraction entre les positions i et j sur le chromosome.

$$Y_{i,j} \sim p(.;\mu_{i,j}) \tag{2}$$

$$\mu_{i,j} = \mathbb{E}(Y_{i,j}) \tag{3}$$

Ainsi, les termes diagonaux sont les plus élevés (auto-interaction), ainsi que des blocs concentrés globalement autour de la diagonale (des replis causent l'interaction de régions proches). Du fait de la symétrie de l'interaction entre les positions (i,j) et (j,i), on considère une matrice triangulaire supérieure. Par exemple :

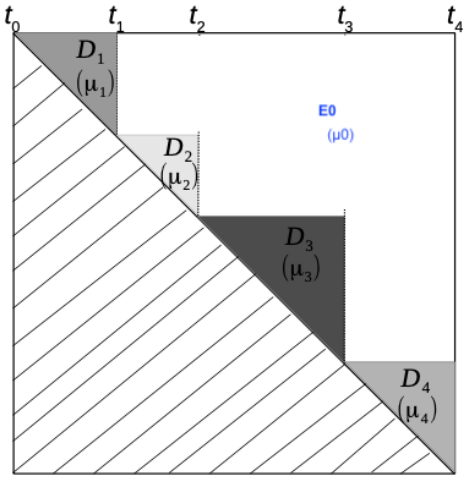


FIGURE 3 – Exemple de matrice triangulaire à blocs diagonaux

On définit D_k les (demi-)blocs diagonaux pour $k = 1, ..., K$, les t_k sont les frontières des blocs, et K le nombre de blocs. On définit également E_0 l'ensemble des positions dans la zone blanche, en dehors des blocs diagonaux. Cette zone est moins importante, la fréquence des interactions est faible du fait des positions éloignées sur le chromosome.

Les paramètres $\mu_{i,j}$ sont supposés constants au sein des blocs :

$$\mu_{i,j} = \mu_k \text{ pour tout } (i,j) \in D_k \tag{4}$$

$$\mu_{i,j} = \mu_0 \text{ pour tout } (i,j) \in E_0 \tag{5}$$

La méthodologie utilisée est la même qu'en une dimension (estimer μ, puis T, puis K) en séparant les paramètres en fonction des zones : les différents blocs D_k, et E_0, comme ci-dessous avec la log-vraisemblance :

$$l(Y) = \sum_{1 \leq i \leq j \leq n} l(Y_{i,j}) \tag{6}$$

$$= \sum_{k=1}^{K} \sum_{(i,j) \in D_k} logp(Y_{i,j}, \mu_k) + \sum_{(i,j) \in E_0} logp(Y_{i,j}, \mu_0) \tag{7}$$

3.1 Problème de la segmentation 2D

Lorsque l'on cherche à déterminer $T = (t_1, t_2, ..., t_{k-1})$, c'est à dire les $k-1$ « breakpoints », on souhaiterait utiliser la programmation dynamique (DP) comme en dimension 1, mais il n'est pas évident que cela soit possible. En effet, **l'utilisation du DP est possible si et seulement si la quantité à optimiser est additive par rapport aux segments**. La quantité à optimiser est :

— soit une fonction de coût à minimiser ;
— soit une fonction de gain à maximiser.

C'est-à-dire que quand on fait la somme sur k (sur les différents « segments ») la quantité sommée ne doit dépendre que de ce qu'il se passe dans **ce** segment, en termes de paramètres. En d'autres mots, le k^{ieme} terme de la somme ne doit pas dépendre de paramètres non-présents dans ce D_k. On se rend bien compte dans l'équation (7) que la présence de μ_0 ne respecte pas cette hypothèse et rend possiblement la programmation dynamique inutilisable dans notre cas de segmentation en deux dimensions.

3.2 Solution

Il s'agit en quelques sorte de se ramener à faire de la segmentation en une dimension, en le sens où l'on ne considère plus

— Les K blocs diagonaux $(D_k)_{k=1,...,K}$
— E_0, « le reste », que l'on « moyennise » dans le paramètre μ_0

Mais plutôt des zones verticales en considérant un rectangle au-dessus de chaque D_k, que l'on nommera R_k.

Visuellement, on considère donc une matrice de la sorte :

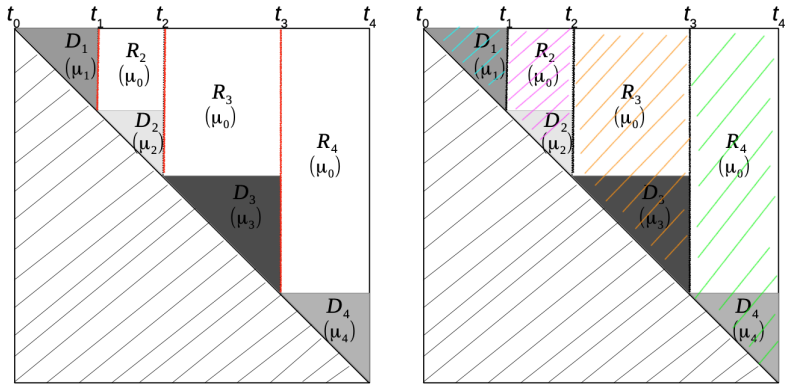


FIGURE 4 – Exemple de segmentation en $K = 4$ blocs

Ainsi, on se ramène en quelque sorte à de la segmentation 1D en considérant chaque $(D_k \cup R_k)_{k=1...K}$ comme étant un « segment ». La moyenne sur ce segment est μ_k dans le demi-bloc D_k, et μ_0 dans le rectangle R_k.

De cette façon, l'équation (7) peut se réécrire :

$$l(Y) = \sum_{k=1}^{K} \left(\sum_{(i,j)\in D_k} logp(Y_{i,j}, \mu_k) + \sum_{(i,j)\in R_k} logp(Y_{i,j}, \mu_0) \right) \tag{8}$$

Visuellement, on se rend compte que **la fonction objectif est maintenant additive par rapport aux segments**, ce qui rend possible l'utilisation de la programmation dynamique et donc la détermination de T.

3.3 Pénalisation sur K

Les problèmes liés aux paramètres μ et $T = (t_1, t_2, ..., t_{k-1})$ étant maintenant discutés et résolus, on cherche à optimiser la fonction objectif sur le nombre de segments K. Dans la première partie, en 1 dimension, on cherchait à trouver K qui soit l'$\arg\min_K\{-l(y|K, \hat{T}, \hat{\mu}) + pen(K)\}$ c'est à dire réaliser un arbitrage entre coller le plus justement aux données du modèle et ne pas considérer trop de segments K (ce qui impliquerait trop de paramètres à estimer). En général, plus le nombre de segments est grand, mieux on s'ajuste aux données. Nous allons voir qu'ici, en 2 dimensions, cela ne fonctionne pas tout à fait pareil.

Dans notre problème, K peut être estimé par $\hat{K} = \arg\min_K\{-l(y|K, \hat{T}, \hat{\mu})\}$, les paramètres $\hat{T}$ et $\hat{\mu}$ ayant été préalablement estimé. On remarque donc qu'en 2D, la pénalisation n'est pas nécessaire dans l'estimation du nombre de segments K optimal. En effet, dans l'équation (8), quelque chose joue naturellement le rôle de pénalité...

Voyons géométriquement ce qu'implique une augmentation de K (une segmentation plus précise) sur la matrice précédente :

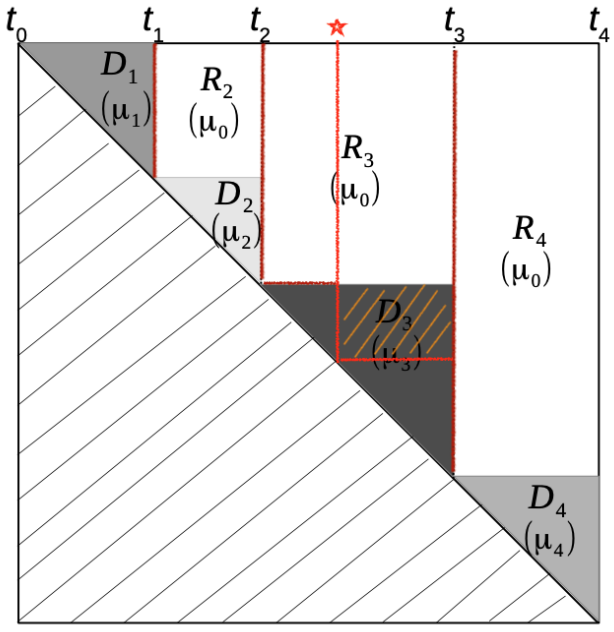


FIGURE 5 – Passage de K=4 à K=5 blocs de segmentation

On avait préalablement $K = 4$ blocs, on observe qu'en augmentant le nombre de « segments » (un breakpoint supplémentaire au niveau de l'étoile rouge), du fait de la structure en demi-blocs diagonaux, une partie de ce qui était auparavant un bloc diagonal (D_3 dans l'exemple) est désormais dans le E_0, c'est le cas de la partie hachurée en orange sur cette image. Ainsi, on se rend compte que plus on segmente (plus on augmente K), plus la partie « en blanc » $E_0 = \bigcup_k R_k$ augmente également.

Si l'on rappelle l'équation (8) :

$$l(Y) = \sum_{k=1}^{K} \left(\sum_{(i,j)\in D_k} logp(Y_{i,j}, \mu_k) + \sum_{(i,j)\in R_k} logp(Y_{i,j}, \mu_0) \right)$$

Nous avons en tête qu'il s'agit de maximiser cette fonction de vraisemblance. On voit donc qu'il y a toute une partie des positions (i, j), celles présentes dans la zone hachurée en orange, qui ne sont plus comptabilisées dans la première mais dans la deuxième somme. Or, la « zone E_0 » est une zone grossière qui colle globalement faiblement aux données, où la vraisemblance est donc plus faible.

L'idée est que lorsque K devient trop grand, $logp(Y_{i,j}, \mu_0)$ pour $(i, j) \in E_0$ est plus faible que $logp(Y_{i,j}, \mu_k)$ pour $(i, j) \in D_k$. Dans ce cas, considérer les données (i, j) comme étant dans un bloc diagonal permet d'avoir une vraisemblance plus élevée que si l'on « noie » ces données dans le grand bloc E_0 : c'est ce qui fera office de pénalité pour K dans la maximisation de la vraisemblance $l(Y)$. On appelle ce principe l'auto-pénalisation.

Cette arbitrage est en quelque sorte un arbitrage *biais/variance* : le biais étant la distance à la vérité, plus on utilise un « gros » modèle (K important), plus on s'approche des données réelles ; mais aussi plus on a de paramètres à estimer, donc d'erreurs (grande variance). C'est ce que nous allons tenter de visualiser en réalisant quelques simulations sur R.

4 Simulations sur R

Pour réaliser ces simulations, nous utilisons le package R *HiCseg* mis à disposition par les auteurs de l'article. Avec ce package est fourni une matrice de taille $200 * 200$, contenant des données gaussiennes et satisfaisant le modèle "D" c'est-à-dire, organisées en blocs diagonaux selon les points de rupture $(40, 80, 120, 160)$.

4.1 Le package HiCseg

La fonction *HiCseg_linkC_R* permet de segmenter des données bi-dimensionnelles, elle nécessite en entrée :

- *size_mat* : la taille de la matrice (pour le jeu de donnée fourni, 200) ;
- *distrib* : la distribution du jeu de données : "B" pour une distribution binomiale négative, "P" pour une distribution de Poisson et "G" pour une distribution gaussienne (dans notre cas, "G") ;
- *mat_data*, la matrice du jeu de données ;
- *model* : le type de modèle. On utilise "D" pour "block-diagonal".

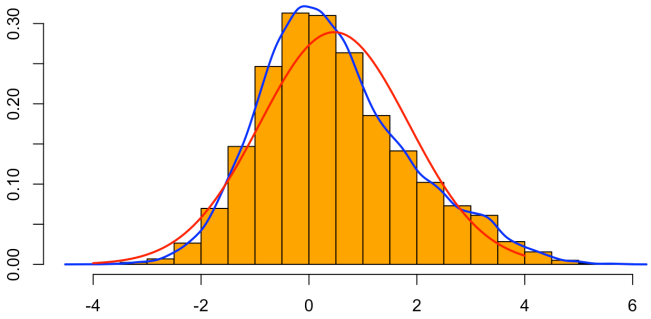
La fonction renvoie alors :

- *t_hat* : l'ensemble des points de rupture ;
- J : la liste des valeurs de log-vraisemblance pour les différentes valeurs de K possibles (avec $K \in [\![1, K_{max}]\!]$). La valeur la plus élevée de log-vraisemblance correspond donc au K choisi par le modèle et au nombre de points de rupture correspondant dans *t_hat*.
- *t_est_mat* : matrice représentant l'ensemble des points de rupture, chaque ligne correspondant à une valeur de K possible (avec $K \in [\![1, K_{max}]\!]$).

Application au dataset fourni avec le package

Nous vérifions que les données de référence fournies avec le package sont gaussiennes, afin de spécifier la bonne loi pour notre modèle. L'histogramme suivant superpose relativement bien la densité d'une loi normale, ce qui nous conforte dans le choix de cette loi.

FIGURE 6 – Distribution des données



En appliquant la fonction *HiCseg_linkC_R* aux données, elle nous renvoie bien la matrice segmentée en blocs diagonaux. Ici, il semble évident que la matrice doit être optimalement découpée en 5 blocs ($K = 5$). C'est bien ce que fait la fonction en renvoyant 4 points de rupture, qui correspondent bien aux limites des blocs dans la représentation ci-dessous ($T = [39, 79, 119, 159]$).

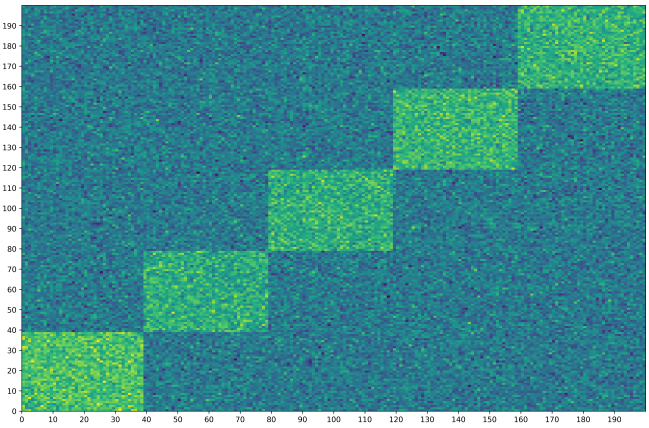


FIGURE 7 – Matrice segmentée en $K = 5$ blocs

Le package fonctionne donc bien avec le dataset fourni. Cependant, la matrice est de faible dimension ($n = 200$). Aussi comme on peut le voir sur la Figure 7, les données de référence sont très "propres", elles possèdent une faible variance ce qui facilite probablement leur segmentation. Nous allons donc jouer sur ce paramètre dans la suite en générant d'autres données.

4.2 Génération des données

Pour mettre en application la méthode décrite dans l'article, nous avons simulé des données sous la forme de matrices **HiC** (avec des blocs diagonaux). Nous simulons une matrice de dimension $n * n$ où chaque valeur suit une loi normale $\mathcal{N}(\mu_0, \sigma^2)$.

Puis nous définissons une valeur de K, les points de rupture $T = (t_1, t_2, ..., t_{K-1})$, et les moyennes correspondantes $(\mu_1, ..., \mu_K)$. Nous avons ainsi défini les blocs diagonaux.
Enfin, nous ajoutons à chaque bloc D_k la valeur $\mu_k - \mu_0$ correspondante, de sorte que les éléments du bloc diagonal D_k suivent donc la loi $\mathcal{N}(\mu_k, \sigma^2)$.

La simulation de différentes matrices (en faisant varier les paramètres de distribution $(\mu_k)_k$, la variance σ et le K_{max}) nous permettra de déterminer dans quelle mesure la méthode présentée dans l'article est capable de segmenter des données **HiC** pour différentes valeurs de paramètres.

Nous définissons les paramètres suivants :
- $\sigma = 2$
- $K = 6$
- $K_{max} = 15$
- $T = (100, 140, 230, 340, 420)$
- $\mu_0 = 0.05$
- $(\mu_k)_{1 \leq k \leq 6} = (3, 8, 5, 2, 4, 3)$

K_{max} représente le nombre maximal de blocs que l'on autorise le programme à faire sur les données.

En appliquant la méthode de segmentation de l'article pour ces paramètres, nous obtenons la matrice suivante :

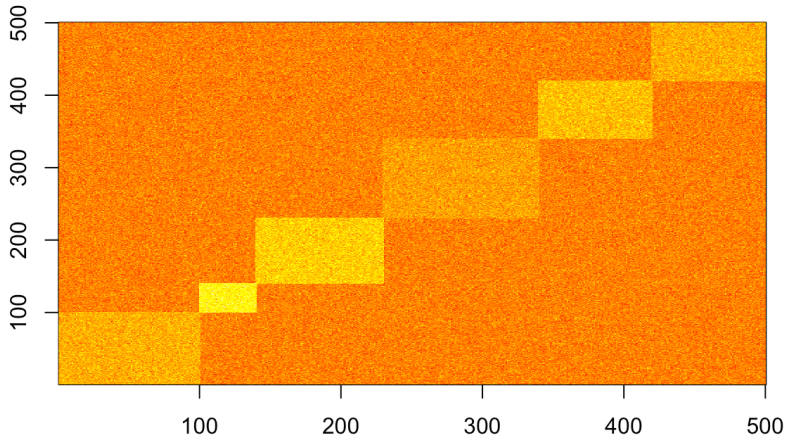


FIGURE 8 – Matrice simulée pour $\sigma = 2$, $K = 6$, $(\mu_k)_{1 \leq k \leq 6} = (3, 8, 5, 2, 4, 3)$

Comme pour le dataset fourni avec le package, la segmentation fonctionne ici très bien. La méthode a déterminé correctement que $K = 6$, ainsi que les différents points de rupture, quelque soit la valeur du μ_k correspondant. Encore une fois, la matrice simulée par nos soins est assez "propre", elle possède une faible variance ici. Nous allons jouer avec ses paramètres.

4.2.1 Moyennes égales entre deux blocs

Nous essayons dans un premier temps de changer μ_2 tel que $\mu_1 = \mu_2 = 2$. L'objectif est de déterminer si la segmentation échoue dans le cas où deux blocs contigus ont des moyennes proches. Dans ce cas, la segmentation pourrait les considérer comme un seul bloc au lieu de deux blocs distincts. Dans la figure ci-dessous, on observe que la méthode de segmentation fonctionne, certainement parce que les deux blocs diagonaux sont séparés par les R_k de moyenne μ_0.

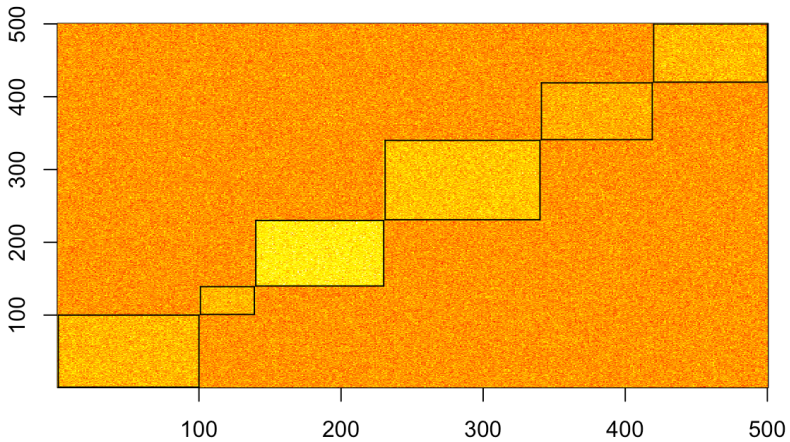


FIGURE 9 – Matrice simulée pour $\mu_1 = \mu_2 = 2$.

4.2.2 Moyenne d'un bloc proche de μ_0

On teste ci-dessous le cas où une valeur de μ_k est proche de μ_0. Ici $\mu_4 = 0.2$. Alors que la segmentation fonctionnait pour toute valeur supérieure de μ_4, on observe que la méthode de segmentation n'arrive plus à déterminer correctement les limites du bloc pour $\mu_4 = 0.2$.

Cela s'explique par le fait que la valeur de μ_4 est tellement proche de μ_0 que la segmentation ne fait plus de distinction entre le bloc diagonal et E_0, et identifie ainsi une multitude de petits blocs.

Le fait d'augmenter K_{max} de 15 à 30 ne fait qu'aggraver le problème : la segmentation, ne distinguant pas le bloc D_4 et E_0, redécoupe cette partie en de plus petits blocs encore. La segmentation est dans ce cas complètement faussée et le principe de l'auto-pénalisation décrit en 3.3 ne semble plus jouer son rôle.

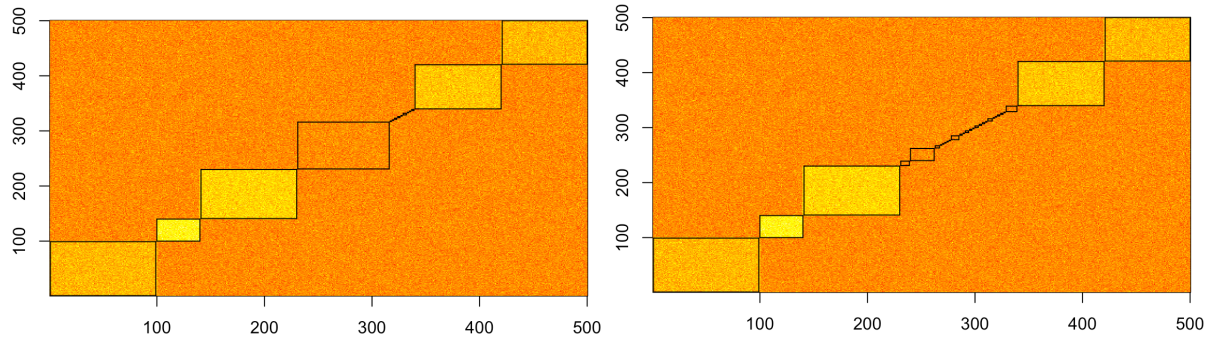


FIGURE 10 – Matrices simulées pour $\mu_4 = 0.2$ et $K_{max} = 15$ (Gauche) ou $K_{max} = 30$ (Droite)

4.2.3 Variation de σ

Nous décidons maintenant de faire varier σ, l'écart-type de la loi normale afin d'observer si la méthode de segmentation fonctionne encore pour des données à forte variance. Nous simulons les matrices suivantes pour $\sigma = 5$.

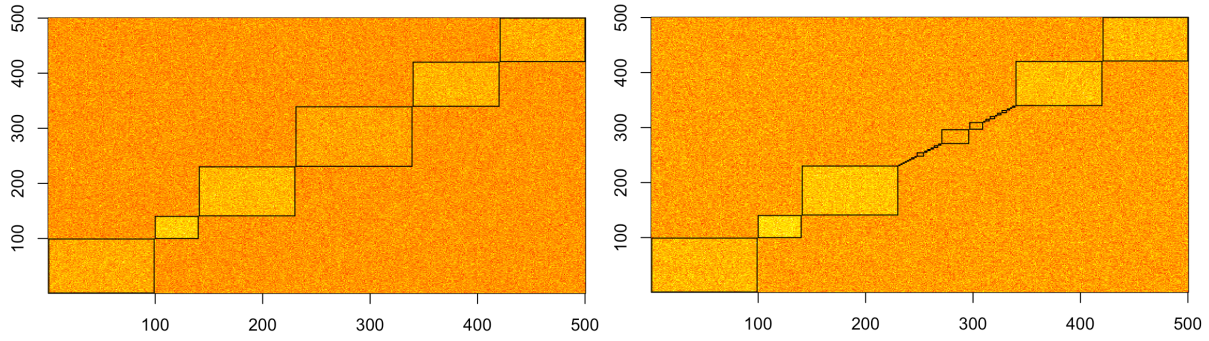


FIGURE 11 – Matrices simulées pour $\mu_4 = 2$ (Gauche) et pour $\mu_4 = 0.6$ (Droite)

Sur la matrice de gauche, on observe qu'avoir augmenté la valeur de σ n'empêche pas la segmentation de se faire correctement. Toutefois, on remarque aussi qu'avec l'augmentation de σ, les valeurs des blocs semblent beaucoup plus proches des valeurs de E_0, comparativement à la Figure 8.

En abaissant la valeur de μ_4 à 0.6 (toujours avec $\sigma = 5$), la segmentation ne fonctionne plus. Le programme ne distingue plus D_4 de E_0 et la matrice est segmentée en 30 blocs, car $K_{max} = 30$. Il convient de rappeler que dans le cas où $\mu_4 = 0.6$ mais $\sigma = 2$, la segmentation fonctionne correctement et nous n'avons pas ce problème.

En conclusion, pour les mêmes paramètres de distribution $(\mu_k)_k$, il apparaît que le fait d'augmenter σ peut conduire la méthode de segmentation à ne plus fonctionner.

De manière générale, ce sont les petites valeurs des μ_k qui sont mal détectées et posent problème dans la segmentation. En effet, plus la variance du modèle est élevée, moins les petites valeurs pour $(\mu_k)_k$ sont bien détectées par la méthode de segmentation. C'est ce que nous allons étudier en détails dans cette dernière section.

4.2.4 Moyennes $(\mu_k)_k$ faibles

Nous redéfinissons ici les paramètres suivants :

- $\sigma = 2$
- $K = 6$
- $K_{max} = 100$
- $T = (100, 140, 230, 340, 420)$
- $\mu_0 = 0.05$
- $(\mu_k)_{1 \leq k \leq 6} = (1, 0.7, 1.1, 0.9, 1.3, 0.8)$

La méthode de segmentation de l'article est utilisée pour ces nouvelles valeurs et les matrices suivantes sont obtenues :

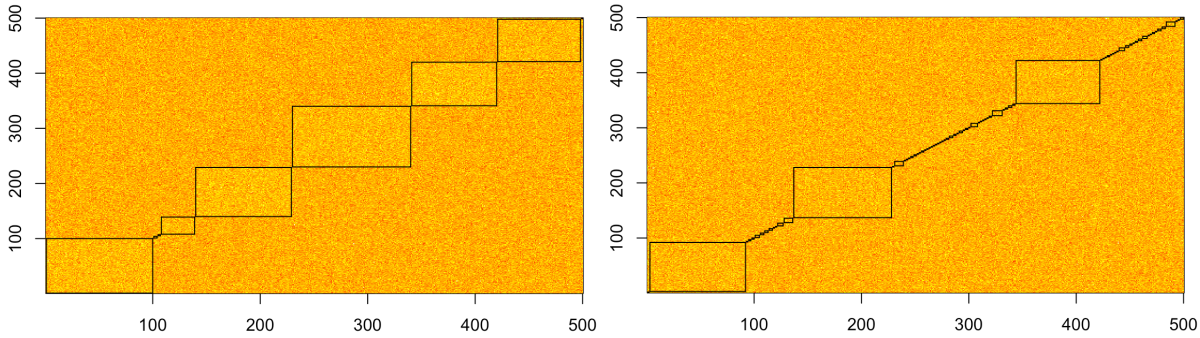


FIGURE 12 – Matrices simulées pour $K_{max} = 100$, $\sigma = 2$ (Gauche) et pour $\sigma = 5$ (Droite)

Dans la matrice de gauche, les blocs sont relativement bien détectés, à part D_2 qui possède la moyenne la plus faible ($\mu_2 = 0.7$), comme nous avons pu en discuter précédemment. En effet, t_hat nous propose 9 points de rupture où 5 auraient du être identifiés.

À droite, la variance a été augmentée dans nos données, ce qui se traduit immédiatement par des difficultés à estimer les blocs diagonaux de la matrice. Comme le nombre maximal de points de rupture n'a pas été contraint par nos soins ($K_{max} = 100$), le programme en propose 81. Ceci illustre le fait que l'auto-pénalisation ne fonctionne plus bien s'il y a trop de bruit dans les données.

Ces exemples montrent que le choix de K_{max} est crucial dans la méthode de segmentation. En effet, K_{max} agit comme un "filet de sécurité", et évite que le programme multiplie les blocs de manière excessive quand il n'arrive pas à détecter un ou plusieurs blocs dont les éléments sont trop proches de ceux de E_0.

5 Conclusion

Ce projet nous a permis d'étudier les différences majeures entre la segmentation en une et deux dimensions, dans le cadre de données **HiC**.

En premier lieu, nous avons présenté la méthode d'estimation des différents paramètres K, T et μ, par maximum de vraisemblance dans un cadre uni-dimensionnel. Nous utilisons également la programmation dynamique pour résoudre le problème du plus court chemin dans la recherche des points de rupture.

Ensuite, nous nous sommes focalisés sur la segmentation 2D et avons discuté de plusieurs changements qui peuvent poser problème. D'après la structure de nos matrices, nous aimerions reconnaître des blocs diagonaux $(D_k)_k$ correspondant aux zones d'interactions entre les chromosomes.

Premièrement, il n'était pas évident que la programmation dynamique s'applique ici, du fait de la zone E_0 qui a pour moyenne μ_0 et vient interagir dans la maximisation de la vraisemblance. La solution est de considérer des zones verticales en considérant chaque rectangle au dessus des blocs $(D_k)_k$. Ceci nous permet de nous ramener à un problème de segmentation en une dimension.

De plus, nous avons expliqué pourquoi le problème de segmentation 2D ne nécessite pas, en théorie, de pénalisation sur K le nombre de segments, contrairement à la segmentation 1D. Cela vient de la forme de la vraisemblance qui est séparée en deux parties, entre D_k et E_0. En segmentant davantage, certaines parties des blocs passent dans E_0 ce qui pénalise naturellement la vraisemblance.

En pratique, nous n'avons que partiellement observé le fait que la segmentation 2D s'auto-pénalise. En effet, cela ne fonctionne pas si bien lorsqu'il y a beaucoup de bruit dans les données.

Nos simulations nous ont aussi permis d'illustrer le fait que les blocs dont la moyenne μ_k est basse (dans le sens où elles sont proches de μ_0) sont plus difficilement détectés, et que cela empire quand la valeur de la variance est élevée.

6 Remerciements

Nous tenons à remercier Emilie Lebarbier de nous avoir présenté le principe de la segmentation 1D dans le cadre du cours de modélisation statistique à l'ENSAE, et de nous en avoir appris davantage sur la segmentation bi-dimensionnelle pour l'analyse des données **HiC**. Nous sommes ravis d'avoir pris part à ce projet qui a été pour nous une très belle ouverture sur les applications de la statistique à la biologie.

7 Sources

Celine Lévy-Leduc, M. Delattre, T. Mary-Huard, S. Robin, Two-dimensional segmentation for analyzing Hi-C data, *Bioinformatics*, Volume 30, Issue 17, 1 September 2014, Pages i386–i392
https://doi.org/10.1093/bioinformatics/btu443