# Modélisation de la structure de chromosomes de souris par modèles de Markov cachés.

Julia Guerra [1], Maxime Jaunatre [2], Ellie Tideswell [3] | Master 2 BEE Grenoble
Mail [1], Mail [2] Mail [3]| 30 novembre 2019

L'avancée des techniques de séquençage a permis d'obtenir de grandes quantités d'informations génomiques. Durant ces dernières années, la génétique a embrassé les outils mathématiques de modélisation, parvenant à une caractérisation statistique des données de séquençage. Cette approche a permis de décrire avec précision les motifs observés dans des régions étudiées , et de prédire leurs présences dans des séquences encore inconnues (Wu *et al.* , 2010). Une des méthodes les plus connues dans ce domaine est l'utilisation des chaînes de Markov, introduites premièrement par Churchill (1992) pour l'analyse de séquences génomiques puis par Durbin *et al.* (1998) pour la détection de régions CGI.

Dans le génome des deutérostomiens, la fréquence du dinucléotide C-G est moins importante qu'attendu sous une distribution aléatoire indépendante des quatre bases azotées. Ceci est une conséquence des mécanismes de protection contre la mutation spontanée du génome. Cependant, dans certaines régions de l'ADN nommées îlots CpG (ou CGI) ce processus de mutation est évolutivement reprimé et donc la fréquence des dinucléotides C-G est donc plus élevée ; par exemple aux alentours de certains promoteurs (Haque *et al.* , 2011, Saxonov *et al.* , 2006, Wu *et al.* , 2010).

La grande variabilité dans la taille, la composition et l'emplacement de ces CGI rend difficile leurs définitions et donc l'établissement d'un algorithme unique permettant leurs détections indubitable (Wu *et al.* , 2010). Ainsi, les modèles de Markov permettent de modéliser les fréquences des nucléotides en fonction de séquences déjà connues ; ces séquences contenant ou non des CGI. En supplément des chaînes de Markov simples, il existe aussi les chaînes de Markov cachées (HMM, Churchill (1992)) : ces dernières décrivent de nombreux processus réels qui suivent un modèle de Markov, mais qui ne sont pas observables. Une chaîne de Markov cachée permettrait donc l'utilisation d'un seul modèle pour identifier un nucléotide (l'observation) et si ce dernier est à l'intérieur d'un îlot CpG ou non (aussi appelé l'état de la région). Les HMM permettent ainsi d'augmenter la résolution de l'analyse, c'est-à-dire de détecter l'emplacement des régions CGI à l'intérieur des séquences.

## Matériel et méthodes

### Modèles de Markov simples

Les modèles de Markov réalisés dans cette étude ont été construits à partir de deux jeux de séquences de souris (Mus musculus). Ces jeux de séquences avaient été caractérisés en avance comme contenant des îlots CpG (on notera "CpG+"), ou ne contenant pas d'îlots CpG ("CpG-"). Les deux jeux de séquences "app", pour la construction des modèles CpG+ et CpG- ,contenaient 1160 et 5755 séquences respectivement. Des jeux supplémentaires "test" également caractérisés comme CpG+ ou CpG- (1163 et 5137 séquences) ont servi à évaluer la performance des modèles.

Dans un premier temps, les fréquences relatives d'observation des bases A, C, G, T ont été calculées pour la totalité des séquences de chaque jeu de données "app" (R, fonction `count` du package `seqinr` ; Charif & Lobry (2007)). Ces données ont permis de construire la matrices de probabilité A du modèle d'ordre 0 (M0). Le terme "ordre" fait référence au nombre de bases précédentes conditionnant la probabilité de présence de la base étudiée. De cette manière, le M0 considère la probabilité d'occurrence de chaque base comme une variable aléatoire (équation 1) dont les probabilités d'occurrence (équation 2)) sont différentes. En plus, des résultats différents sont attendus en fonction de la nature CpG+

ou CpG- des séquences ; c'est pourquoi deux matrices de probabilité A+ et A- ont été construites, provenant respectivements des comptages du jeu CpG+ et CpG-.

Le modèle de Markov d'ordre 1 (M1) rassemble les occurrences de chaque base en fonction de la base précédente. Les matrices de transition q1+ et q1- sont matrices 4x4 qui ont été donc construites à partir des comptages de chaque couple de bases. Vu qu'elle ne peut pas dépendre d'une base précédente, la probabilité d'occurrence de chaque base initiale a été considérée comme une variable aléatoire (équation 3) à probabilités équivalentes (équation 4). Ce protocole de construction de modèle a été refait pour l'ordre 2, obtenant une matrice 16x4. Pareil pour l'ordre 3 (matrice 64 x 4), l'ordre 4 ...jusqu'à l'ordre 5. Pour les modèles d'ordre supérieur 0, les lignes de la matrice ont été rangées de sorte que la somme de chaque ligne soit égal à 1, à cause de la nature conditionnelle des probabilités, comme dans l'exemple suivant (table 1.

$$Y \in B; B = a, c, g, t \tag{1}$$

$$P(Y_i = k) \forall k \in B \tag{2}$$

$$X \in B; B = a, c, g, t \tag{3}$$

$$P(A) = P(C) = P(G) = P(T) = P(X \in B) = \frac{1}{4} \tag{4}$$

A partir des matrices de transition, on peut calculer la log-Vraisemblance d'une séquence sous un modèle MX correspondant comme la somme du log de la probabilité de premières bases (région de taille égale à l'ordre) avec la somme du produit de la matrice de transition par la matrice d'occurence des mots dans la séquence (voir équation 5).

| | a | c | g | t |
|---|---|---|---|---|
| a | 0.29 | 0.21 | 0.30 | 0.20 |
| c | 0.26 | 0.30 | 0.17 | 0.27 |
| g | 0.24 | 0.27 | 0.30 | 0.20 |
| t | 0.18 | 0.26 | 0.28 | 0.29 |

TABLE 1 – Matrice de transition du modèle CpG+ d'ordre 1

$$P_+(Sequence) = log\left[P_{initial}(mot_{initial})\right] + \sum_{i=1}^{n=4^{ordre+1}} log\left[P_i(mot_i) \cdot N_i(mot_i)\right] \tag{5}$$

## Choix du meilleur modele

La performance du M1 a été testée sur les deux jeux de séquences de test. La log-vraisemblance de chaque séquence a été calculé pour chaque modèle (CpG+ et CpG-) et la séquence est donc associée à l'état pour lequel la log-vraisemblance est la plus grande. Pour le jeu de données CpG+, les séquences caractérisées comme CpG+ sont considérées comme vrais positifs (VP) et les séquences caractérisées CpG- comme faux négatifs (FN). Pour le jeu de données CpG-, les séquences caractérisées comme CpG+ sont considérées comme faux positifs (FP) et celles caractérisées comme CpG- comme vrais négatifs (VN). Le même protocole a été suivi pour tester la performance des modèles 1 à 6.

La spécificité et la sensibilité de chaque modèle ont été calculées à partir de ces résultats, selon les équations illustrées en 6 et 7.

$$Sensibility = \frac{VP}{VP + FN} \tag{6}$$

$$Specificity = \frac{VN}{VN + FP} \tag{7}$$

Ce processus, répété pour toutes les combinaisons de Mi+/Mj- (avec i et j allant de 0 à 5), a permis de connaître la meilleure combinaison de modèle. Pour l'obtenir, les données de sensibilité et spécificité pour les modèles ont été sommés entre elles. La combinaison d'ordres portant la valeur maximale étant la valeur (5,4) de la matrice ; les calculs de la chaîne de Markov cachée ont été réalisés sur un modèle d'ordre 5 pour les séquences CpG+ et un modèle d'ordre 4 pour les séquences CpG-. La figure 1 montre les résultats de sensibilité et spécificité mentionnées ici.

Modèles de Markov cachés Les bases mathématiques de l'analyse des îlots CGI parmi des HMM dans cet étude suit le protocole décrit dans Churchill (1992). Pour la construction des HMM, il a été nécessaire de calcu-

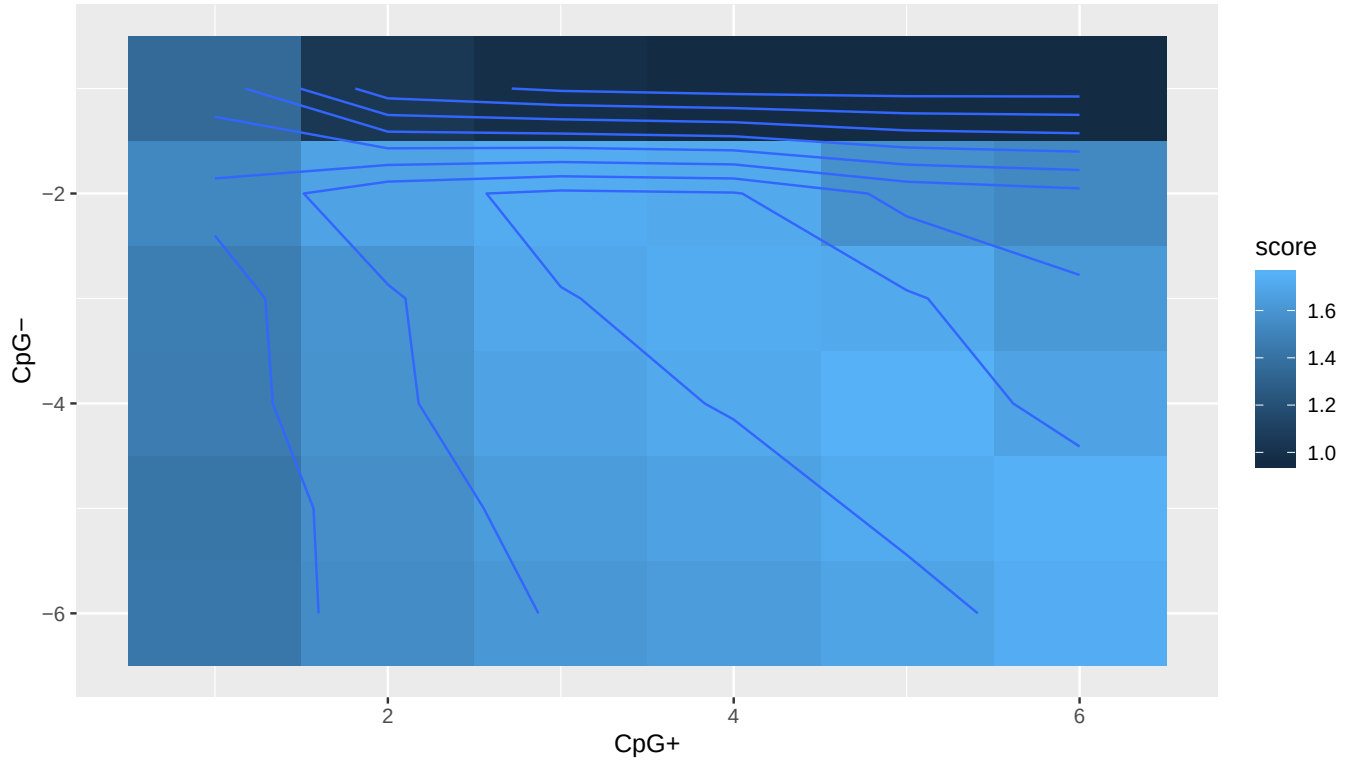


FIGURE 1 – Evolution du score de sensibilité + spécificité selon les ordre de modèles

ler la probabilité de transition entre état CpG+ et état CpG- à l'intérieur d'une séquence. Les valeurs de cette matrice de transition 2x2 contenant d'états ont été obtenues à partir de la bibliographie, en prenant compte de la longueur moyenne des îlots CpG. Cette matrice (2) s'ajoute donc à la matrice des probabilités d'occurrence de chaque base (ou combinaison de bases) et à la matrice d'occurrence des bases initiales.

## L'algorithme de Viterbi

Afin de trouver la séquence optimale d'états qui correspond à une séquence donnée d'observations, il est possible d'utiliser une fenêtre glissante (un algorithme naïf), dans laquelle les log vraisemblances sont calculés pour des segments de bases d'une longueur donnée. Bien que facile à implémenter, les résultats (en terme des prédictions des CGI) dépendent de la taille de la fenêtre choisi, ceci peut représenter un biais de cette méthode. Une façon alternative peut être l'algorithme de Viterbi, un exemple de la programmation dynamique, qui permet d'identifier la séquence qui maximise la probabilité de générer les observations . Le chemin le plus probable étant donné un modèle est déterminé via une procédure récursive. L'algorithme de Viterbi est décrit comme suit :

| | M+ | M- |
|---|---|---|
| M+ | -0.00 | -6.91 |
| M- | -11.74 | -0.00 |

TABLE 2 – Matrice de transition du modèle CpG+ d'ordre 1

## Smoothing

La technique de "Smoothing" représente une technique mathématique qui enlève la variabilité parmi les données, impliquant souvent la redistribution du poids entre des régions de haute probabilité, et des régions de "zéro probabilité" . Dans le cadre de cette étude, le "Smoothing" revient donc à lisser la caractérisation des différentes régions en les ré-assignant selon 2 procédés successifs.

Dans un premier temps, les régions de longueur inférieur à un certain seuil (S) sont assignée à une nouvelle catégorie "Ambiguous", en vert dans la figure

2. Cette première étape comporte également un algorithme qui compile ces nouvelles régions en une seule quand elles se suivent dans la séquence (voir bases 9 à 13), afin de mesurer la longueur de cette nouvelle région dont la catégorie est devenue unique.

Le second procédé vérifie la longueur de ces nouvelles régions ambiguës et leurs situations sur la séquence. En effet, il arrive qu'une région de petite taille soit considérée comme ambiguë entre deux régions d'une même catégorie (voir base 5). On peut donc supposer qu'il s'agit de bruit et que cette région est probablement de la même catégorie que celles qui l'entoure. Ainsi, le second procédé de smoothing va ré-assigner des régions ambiguës si leurs tailles sont inférieures à un seuil et que les régions bordantes sont de même nature. On note que l'algorithme de 'Smoothing' ne peut être utilisé qu'avec le second procédé, car en l'absence de régions ambiguës aucune ré-assignation vers CpG+ ou CpG- n'est possible.

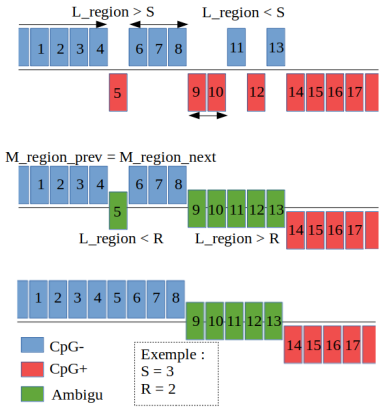


FIGURE 2 – Evolution du score de sensibilité + spécificité selon les ordre de modèles

# Résultats

## mus1

Après un viterbi pour le modèle M5+/M4-, et un smoothing avec S = 60 et R = 40, le chromosome de souris numero 1 met en évidence 19 régions CpG+ contre 29 régions CpG-, avec un grand déséquilibre de longueur entre ces régions. Ainsi, les régions CpG+ font en moyenne 646.053 contre 7658 pour les régions CpG-. A contrario, les régions ambigües sont de faibles longueurs (63.226), valeur proche du seuil de smoothing comme attendu.

Du fait de la grande longueur des régions CpG-, on observe que les régions CpG+ sont groupées et séparées par d'autres régions CpG- de taille limitée.

L'ensemble des résultats pour le chromosome 1 est disponible en annexe.

## mus2

Après un viterbi pour le modèle M5+/M4-, et un smoothing avec S = 60 et R = 40, le chromosome de souris numero 2 met en évidence 22 régions CpG+ contre 26 régions CpG-, avec un grand déséquilibre de longueur entre ces régions. Ainsi, les régions CpG+ font en moyenne 667.727 contre 6179.346 pour les régions CpG-.

Par rapport au chromosome numero 1, les régions CpG+ dont plus longues et centrées sur la fin du chromosome en une région bien définie.

L'ensemble des résultats pour le chromosome 2 est disponible en annexe.

## mus3

Après un viterbi pour le modèle M5+/M4-, et un smoothing avec S = 60 et R = 40, le chromosome de souris numero 1 met en évidence 9 régions CpG+ contre 12 régions CpG-, avec un grand déséquilibre de longueur entre ces régions. Ainsi, les régions CpG+ font en moyenne 523.222 contre 11255.167 pour les régions CpG-.

Concernant le chromosome 3, il présente très peu de régions CpG+ et elles sont de faible longueurs.

L'ensemble des résultats pour le chromosome 3 est disponible en annexe.

# Discussion

L'implémentation d'un algorithme de Viterbi a permis la caractérisation rapide de trois chromosomes de souris et la détection de régions CpG+. Ce genre d'outils montre donc son intéret pour une analyse rapide de données génomiques. On note cependant que la dernière partie de l'algorithme (smoothing) a été implémentée de

manière sommaire et qu'une étude approfondie des seuils sur les résultats manque. La prochaine étape serait donc de tester différents seuils de smoothing (variant S et R), afin d'optimiser les résultats sur chaque chromosome.

L'apprentissage de nos modèles est ici relativement rapide car le jeu de donnée d'entraînement est limité et nos modèles encore simplifiés. Cependant, il est important de noter que des modèles plus complexes entraînent des temps de calculs d'autant plus important. Dans ce contexte, il est donc important de souligner que tout n'a pas été fait pour optimiser l'apprentissage de nos modèles. Il reste encore possible de paralléliser différentes étapes de l'apprentissages ou du test des modèles différents. Une solution encore plus avancée serait de changer de langage de programmation afin de produire un algorithme plus efficace que celui proposé ici sour `R`.

Afin d'améliorer la performance de notre modèle, il serait raisonnable de raffiner la sélection des CGI identifiés, selon des propriétés connus des CGIs, tels que le contenu de GC, la fraction de CpG et un seuil de longueur (Lan *et al.* , 2009). De nombreux études ont relevé la fausse identification dans les CGI de petites quantités des nucléotides CpG+, identifiées comme CpG-. Un seuil minimal de longueur entre des nucléotides CpG+ en voisinage pourrait résoudre ce problème, donc un 'smoothing' plus dynamique ou les paramètres changent en fonction de l'état dans lequel la nucléotide se situe (CpG+ ou CpG-). Les CGI contiennent également une ratio élevé de G/C, ce qui est normalement de 60% au minimum. L'application d'un tel seuil aux CGI identifiés pourrait représenter une amélioration supplémentaire du modèle. Les CGI sont souvent définis comme des régions d'un longueur de 140 paires de base, cependant certains auteurs indiquent qu'il existe une certaine variabilité qui rend cette définition éronée. Un seuil minimum de longueur a donc été suggéré par certains auteurs et il pourrait s'avérer intéressant de comparer les prédictions sans et avec son application, à faire avec caution (Lan *et al.* , 2009).

Une des limites des modèles de Markov cachés en général est la supposition que les distributions des paramètres d'observation suivent une loi géométrique. (Matthias, 2013) a identifié d'autres limitations de l'utilisation des modèles de Markov cachés, dans un context des prédictions des CGI. Parmi ces limitations se trouve le constat que les résultats d'un tel modèle dépendent fortement des probabilités initiales, ainsi que des itérations d'entraînement du modèle. Berg a donc suggéré l'entraînement du modèle, et la ré-estimation subséquente des probabilités initiales afin de mieux représenter les états cachés, et ceci pourrait également représenter une amélioration possible de notre modèle.

Le détection précise des îlots CPG reste un sujet important dans un contexte médical, avec, parmi d'autres, de plus en plus d'associations identifiées entre le méthylation modifié et le cancer. Les régions se situant à moins de 20000 paires de base des frontières CGI, pour exemple, ont été identifiés comme des bons prédicteurs pour la location des régions qui subissent un méthylation modifié, spécifique aux cancers ("cancer-specific differentially methylated regions") (Irizarry *et al.* , 2009). L'amélioration des modèles qui nous permettent donc de prédire avec précision ces régions représentent un défi important dans la détection des cancers.

## Ressources

Ce document est disponible en ligne sous format ".Rnw", contenant tout le code néccessaire à la reproduction de l'analyse, réalisée avec un script en langage R (R.Team, 2017), ainsi que le jeu de données de départ. L'ensemble est situé sur Github : `https://github.com/gowachin/BeeMarkov` et peut être installé sur R via les commandes suivantes.

```
> # NOT RUN
> library(devtools)
> install_github("gowachin/BeeMarkov")
> library(BeeMarkov)
```

## Bibliographie

Charif, Delphine, & Lobry, Jean R. 2007. SeqinR 1.0-2 : A Contributed Package to the R Project for Statistical Computing Devoted to Biological Sequences Retrieval and Analysis.

Churchill, Gary A. 1992. Hidden Markov chains and the analysis of genome structure. *Computers and Chemistry*, **16**(2), 107–115.

Durbin, Richard, Eddy, Sean R., Krogh, Anders, & Mitchison, Graeme. 1998. Biological sequence analysis. *Biological sequence analysis*.

Haque, A. N.A., Hossain, M. E., Haque, M. E., Hasan, M. M., Malek, M. A., Rafii, M. Y., & Shamsuzzaman, S. M. 2011. CpG islands and the regulation of transcription. *GENES & DEVELOPMENT*, **25**(1), 1010–1022.

Irizarry, Rafael A., Wu, Hao, & Feinberg, Andrew P. 2009. A species-generalized probabilistic model-based definition of CpG islands. *Mammalian Genome*, **20**(9-10), 674–680.

Lan, Man, Xu, Yu, Li, Lin, Wang, Fei, Zuo, Ying, Chen, Yuan, Tan, Chew Lim, & Su, Jian. 2009. CpG-Discover : A machine learning approach for CpG islands identification from human DNA sequence. *Proceedings of the International Joint Conference on Neural Networks*, **117543**, 1702–1707.

Matthias, Berg. 2013. *Formal Verification of Cryptographic Security Proofs*. Ph.D. thesis, Saarland University.

R.Team. 2017. R : A language and environment for statistical computing (Version 3.4. 2)[Computer software]. *Vienna, Austria : R Foundation for Statistical Computing*.

Saxonov, Serge, Berg, Paul, & Brutlag, Douglas L. 2006. A genome-wide analysis of CpG dinucleotides in the human genome distinguishes two distinct classes of promoters. *Proceedings of the National Academy of Sciences of the United States of America*, **103**(5), 1412–1417.

Wu, Hao, Caffo, Brian, Jaffee, Harris A., Irizarry, Rafael A., & Feinberg, Andrew P. 2010. Redefining CpG islands using hidden Markov models. *Biostatistics*, **11**(3), 499–514.

## Annexes

Représentation schématique de la structure du chromosome de souris numéro 1. Les modèles de Markov cachés permettent la détection des régions CpG+ (en bleu) versus les régions CpG- (en rouge). Le smoothing a été réalisé avec de valeurs de seuillage de s = 60 et r = 40, mais un certain marge de régions ambigües (en vert) est encore visible.

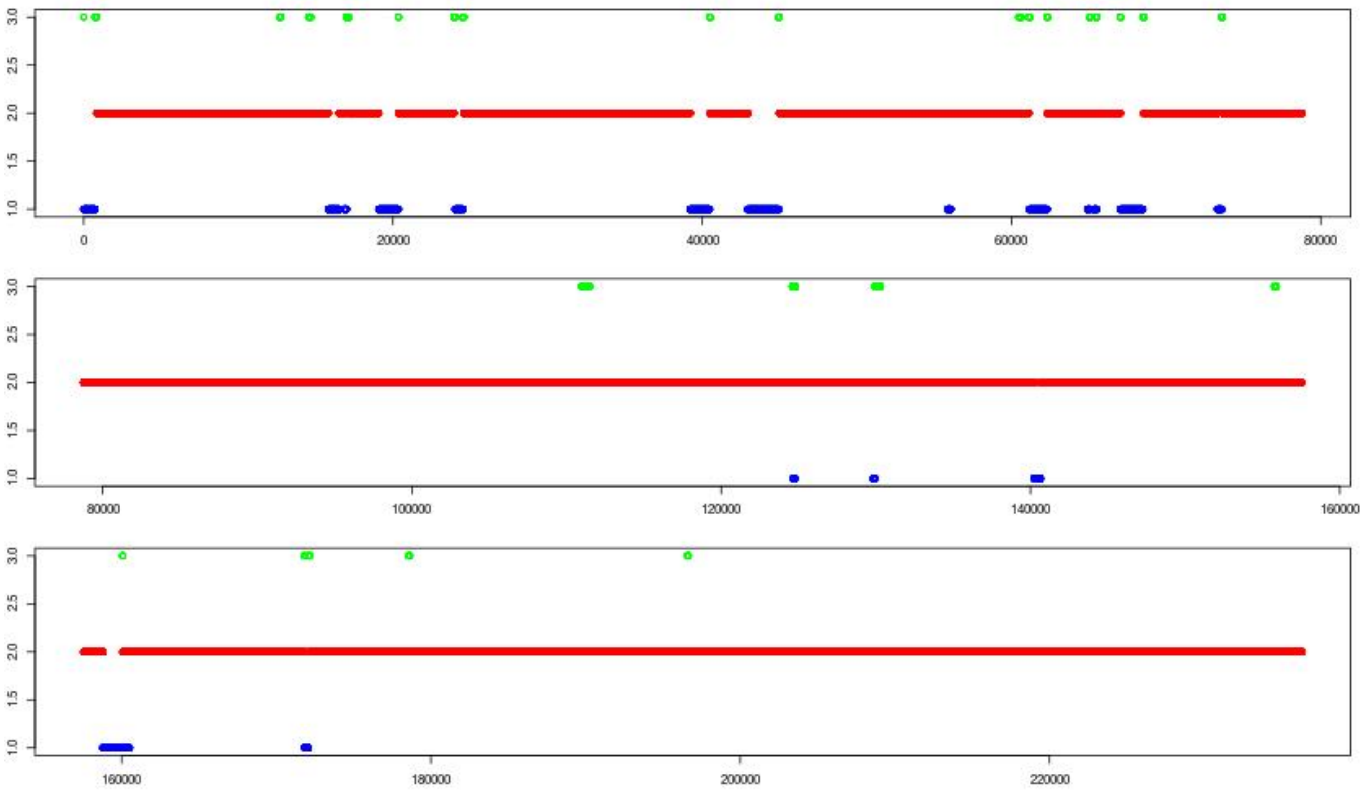


FIGURE 3 – Représentation schématique de la structure du chromosome de souris numéro 1. Les modèles de Markov cachés permettent la détection des régions CpG+ (en bleu) versus les régions CpG- (en rouge). Le smoothing a été réalisé avec de valeurs de seuillage de s = 60 et r = 40, mais un certain marge de régions ambigües (en vert) est encore visible.

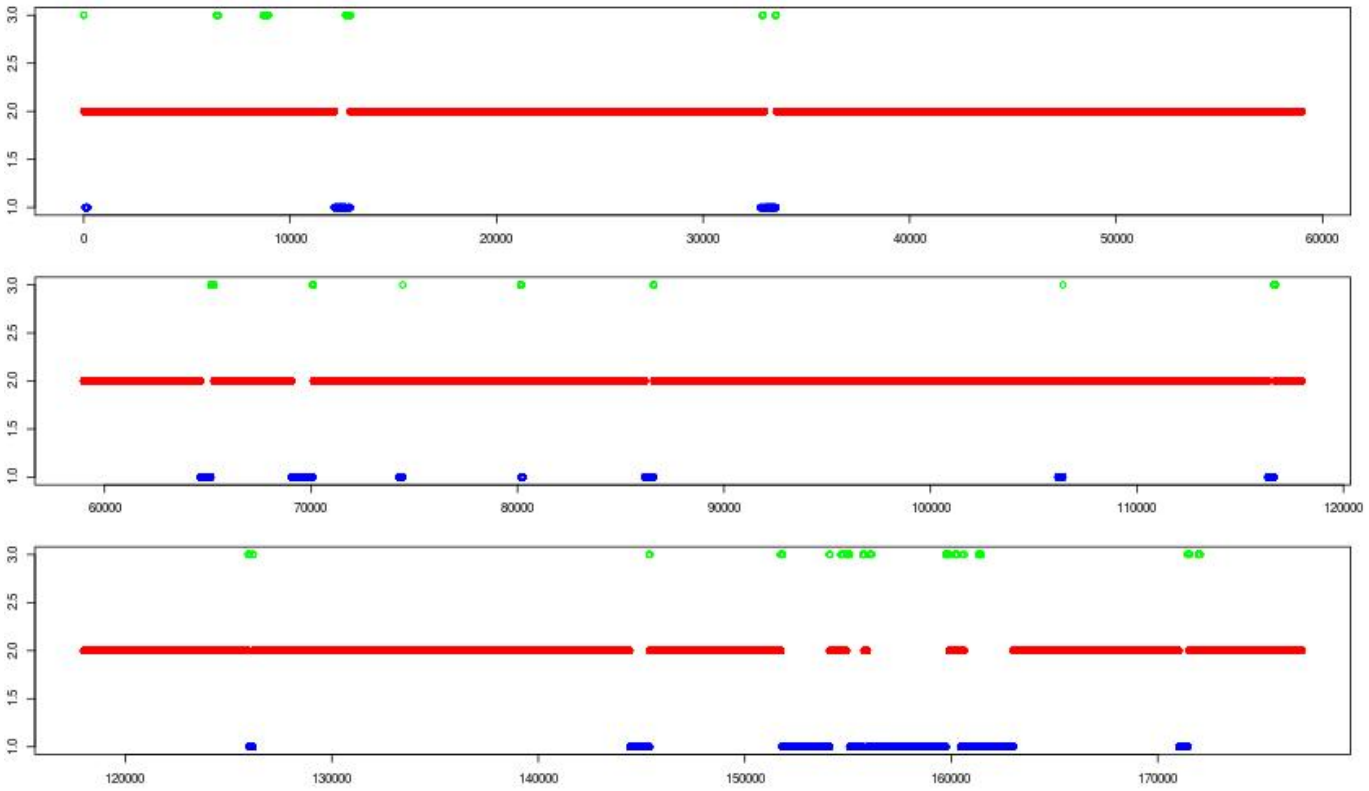


FIGURE 4 – Représentation schématique de la structure du chromosome de souris numéro 2. Les modèles de Markov cachés permettent la détection des régions CpG+ (en bleu) versus les régions CpG- (en rouge). Le smoothing a été réalisé avec de valeurs de seuillage de s = 60 et r = 40, mais un certain marge de régions ambigües (en vert) est encore visible.

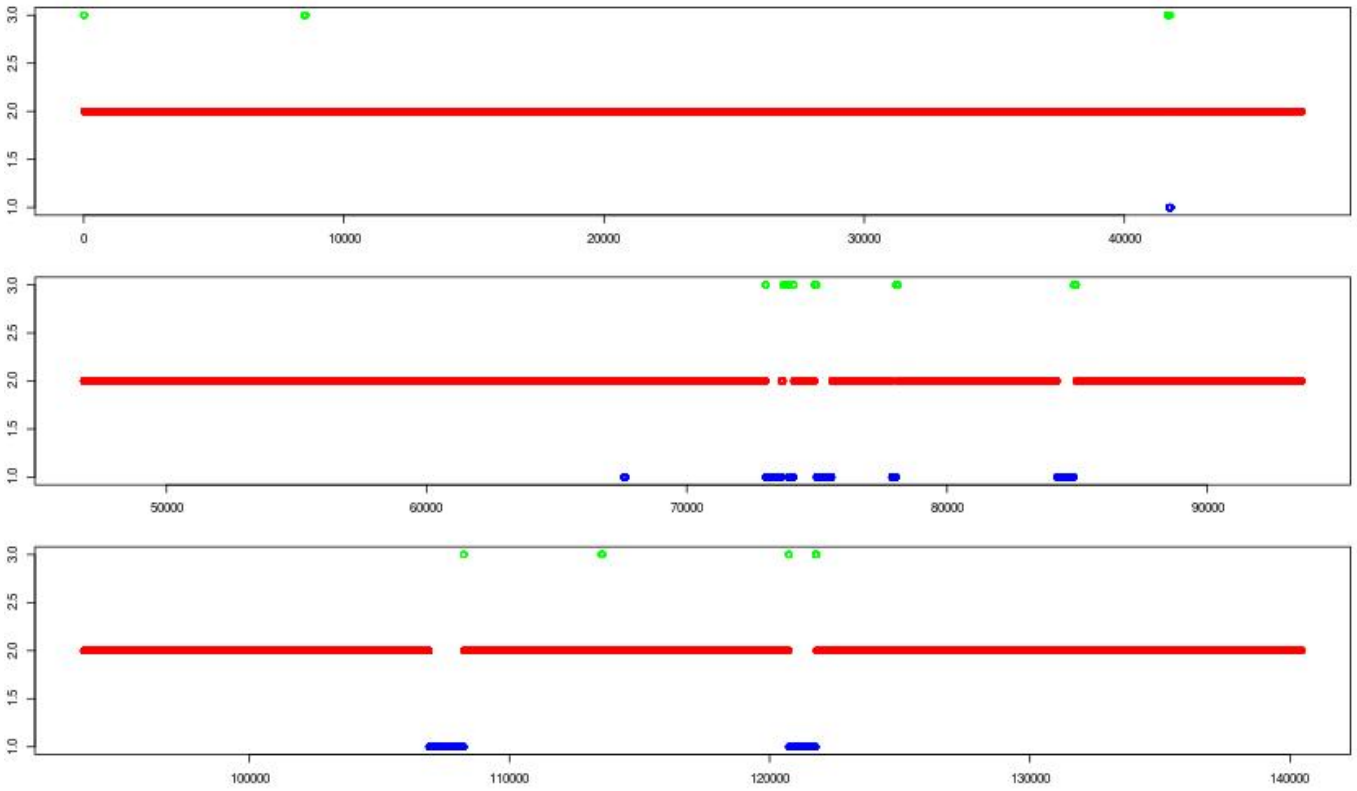


FIGURE 5 – Représentation schématique de la structure du chromosome de souris numéro 3. Les modèles de Markov cachés permettent la détection des régions CpG+ (en bleu) versus les régions CpG- (en rouge). Le smoothing a été réalisé avec de valeurs de seuillage de s = 60 et r = 40, mais un certain marge de régions ambigües (en vert) est encore visible.

### scripts

Les analyses ont été effectuées avec le code ci-dessous, sous Rstudio (Version 1.1.456) :

```
### R code from vignette source 'Markov_Report_Guerra_Jaunatre.Rnw'
### Encoding: UTF-8

###################################################
### code chunk number 1: packages
###################################################
library(xtable)
library(ggplot2)
library(reshape)
# library(plyr)
# library(MASS)
# library(ResourceSelection)
# library(ggpubr)
# library(tidyr)

if("BeeMarkov" %in% installed.packages()) {
  library(BeeMarkov)
}else{
  # library(devtools)
  # install_github("gowachin/BeeMarkov")
  # library(BeeMarkov)
}

```

| | beg | end | long | model |
|---|---|---|---|---|
| 1 | 1.00 | 4.00 | 4.00 | 3.00 |
| 2 | 5.00 | 738.00 | 734.00 | 1.00 |
| 3 | 739.00 | 845.00 | 107.00 | 3.00 |
| 4 | 846.00 | 12701.00 | 11856.00 | 2.00 |
| 5 | 12702.00 | 12750.00 | 49.00 | 3.00 |
| 6 | 12751.00 | 14547.00 | 1797.00 | 2.00 |
| 7 | 14548.00 | 14678.00 | 131.00 | 3.00 |
| 8 | 14679.00 | 15823.00 | 1145.00 | 2.00 |
| 9 | 15824.00 | 16527.00 | 704.00 | 1.00 |
| 10 | 16528.00 | 16844.00 | 317.00 | 2.00 |
| 11 | 16845.00 | 16992.00 | 148.00 | 1.00 |
| 12 | 16993.00 | 17135.00 | 143.00 | 3.00 |
| 13 | 17136.00 | 19081.00 | 1946.00 | 2.00 |
| 14 | 19082.00 | 20351.00 | 1270.00 | 1.00 |
| 15 | 20352.00 | 20361.00 | 10.00 | 3.00 |
| 16 | 20362.00 | 23943.00 | 3582.00 | 2.00 |
| 17 | 23944.00 | 24021.00 | 78.00 | 3.00 |
| 18 | 24022.00 | 24517.00 | 496.00 | 1.00 |
| 19 | 24518.00 | 24580.00 | 63.00 | 3.00 |
| 20 | 24581.00 | 39223.00 | 14643.00 | 2.00 |
| 21 | 39224.00 | 40471.00 | 1248.00 | 1.00 |
| 22 | 40472.00 | 40512.00 | 41.00 | 3.00 |
| 23 | 40513.00 | 42950.00 | 2438.00 | 2.00 |
| 24 | 42951.00 | 44942.00 | 1992.00 | 1.00 |
| 25 | 44943.00 | 44986.00 | 44.00 | 3.00 |
| 26 | 44987.00 | 55887.00 | 10901.00 | 2.00 |
| 27 | 55888.00 | 56071.00 | 184.00 | 1.00 |
| 28 | 56072.00 | 60474.00 | 4403.00 | 2.00 |
| 29 | 60475.00 | 60626.00 | 152.00 | 3.00 |
| 30 | 60627.00 | 61128.00 | 502.00 | 2.00 |
| 31 | 61129.00 | 61168.00 | 40.00 | 3.00 |
| 32 | 61169.00 | 62295.00 | 1127.00 | 1.00 |
| 33 | 62296.00 | 62327.00 | 32.00 | 3.00 |
| 34 | 62328.00 | 64912.00 | 2585.00 | 2.00 |
| 35 | 64913.00 | 65065.00 | 153.00 | 1.00 |
| 36 | 65066.00 | 65088.00 | 23.00 | 3.00 |
| 37 | 65089.00 | 65349.00 | 261.00 | 2.00 |
| 38 | 65350.00 | 65489.00 | 140.00 | 1.00 |
| 39 | 65490.00 | 65522.00 | 33.00 | 3.00 |
| 40 | 65523.00 | 67039.00 | 1517.00 | 2.00 |
| 41 | 67040.00 | 67058.00 | 19.00 | 3.00 |
| 42 | 67059.00 | 68489.00 | 1431.00 | 1.00 |
| 43 | 68490.00 | 68551.00 | 62.00 | 3.00 |
| 44 | 68552.00 | 73322.00 | 4771.00 | 2.00 |
| 45 | 73323.00 | 73588.00 | 266.00 | 1.00 |
| 46 | 73589.00 | 73640.00 | 52.00 | 3.00 |
| 47 | 73641.00 | 110982.00 | 37342.00 | 2.00 |
| 48 | 110983.00 | 111035.00 | 53.00 | 3.00 |
| 49 | 111036.00 | 111141.00 | 106.00 | 2.00 |
| 50 | 111142.00 | 111255.00 | 114.00 | 3.00 |
| 51 | 111256.00 | 111436.00 | 181.00 | 2.00 |
| 52 | 111437.00 | 111476.00 | 40.00 | 3.00 |
| 53 | 111477.00 | 124605.00 | 13129.00 | 2.00 |
| 54 | 124606.00 | 124642.00 | 37.00 | 3.00 |
| 55 | 124643.00 | 124777.00 | 135.00 | 1.00 |
| 56 | 124778.00 | 124810.00 | 33.00 | 3.00 |
| 57 | 124811.00 | 129831.00 | 5021.00 | 2.00 |
| 58 | 129832.00 | 129949.00 | 118.00 | 1.00 |
| 59 | 129950.00 | 130144.00 | 195.00 | 3.00 |
| 60 | 130145.00 | 130219.00 | 75.00 | 2.00 |
| 61 | 130220.00 | 130300.00 | 81.00 | 3.00 |
| 62 | 130301.00 | 140220.00 | 9920.00 | 2.00 |
| 63 | 140221.00 | 140638.00 | 418.00 | 1.00 |
| 64 | 140639.00 | 155782.00 | 15144.00 | 2.00 |
| 65 | 155783.00 | 155899.00 | 117.00 | 3.00 |
| 66 | 155900.00 | 158791.00 | 2892.00 | 2.00 |
| 67 | 158792.00 | 160057.00 | 1266.00 | 1.00 |
| 68 | 160058.00 | 160073.00 | 16.00 | 3.00 |
| 69 | 160074.00 | 160311.00 | 238.00 | 2.00 |
| 70 | 160312.00 | 160508.00 | 197.00 | 1.00 |
| 71 | 160509.00 | 171818.00 | 11310.00 | 2.00 |
| 72 | 171819.00 | 171834.00 | 16.00 | 3.00 |
| 73 | 171835.00 | 172082.00 | 248.00 | 1.00 |
| 74 | 172083.00 | 172151.00 | 69.00 | 3.00 |
| 75 | 172152.00 | 178572.00 | 6421.00 | 2.00 |
| 76 | 178573.00 | 178630.00 | 58.00 | 3.00 |
| 77 | 178631.00 | 196596.00 | 17966.00 | 2.00 |
| 78 | 196597.00 | 196644.00 | 48.00 | 3.00 |
| 79 | 196645.00 | 236317.00 | 39673.00 | 2.00 |

TABLE 3 – Table récapitulative de la structure du chromosome de souris numéro 1. Les régions CpG+ (model = 1) versus les régions CpG- (model = 2). Le smoothing a été réalisé avec de valeurs de seuillage de s = 60 et r = 40, mais un certain marge de régions ambigües (model = 3) est encore visible.

| | beg | end | long | model |
|---|---|---|---|---|
| 1 | 1.00 | 4.00 | 4.00 | 3.00 |
| 2 | 5.00 | 5.00 | 1.00 | 3.00 |
| 3 | 6.00 | 77.00 | 72.00 | 2.00 |
| 4 | 78.00 | 187.00 | 110.00 | 1.00 |
| 5 | 188.00 | 6451.00 | 6264.00 | 2.00 |
| 6 | 6452.00 | 6506.00 | 55.00 | 3.00 |
| 7 | 6507.00 | 8719.00 | 2213.00 | 2.00 |
| 8 | 8720.00 | 8762.00 | 43.00 | 3.00 |
| 9 | 8763.00 | 8878.00 | 116.00 | 2.00 |
| 10 | 8879.00 | 8938.00 | 60.00 | 3.00 |
| 11 | 8939.00 | 12146.00 | 3208.00 | 2.00 |
| 12 | 12147.00 | 12696.00 | 550.00 | 1.00 |
| 13 | 12697.00 | 12813.00 | 117.00 | 3.00 |
| 14 | 12814.00 | 12898.00 | 85.00 | 1.00 |
| 15 | 12899.00 | 12911.00 | 13.00 | 3.00 |
| 16 | 12912.00 | 32789.00 | 19878.00 | 2.00 |
| 17 | 32790.00 | 32872.00 | 83.00 | 1.00 |
| 18 | 32873.00 | 32887.00 | 15.00 | 3.00 |
| 19 | 32888.00 | 32970.00 | 83.00 | 2.00 |
| 20 | 32971.00 | 33511.00 | 541.00 | 1.00 |
| 21 | 33512.00 | 33539.00 | 28.00 | 3.00 |
| 22 | 33540.00 | 64651.00 | 31112.00 | 2.00 |
| 23 | 64652.00 | 65154.00 | 503.00 | 1.00 |
| 24 | 65155.00 | 65302.00 | 148.00 | 3.00 |
| 25 | 65303.00 | 69054.00 | 3752.00 | 2.00 |
| 26 | 69055.00 | 70074.00 | 1020.00 | 1.00 |
| 27 | 70075.00 | 70116.00 | 42.00 | 3.00 |
| 28 | 70117.00 | 74258.00 | 4142.00 | 2.00 |
| 29 | 74259.00 | 74438.00 | 180.00 | 1.00 |
| 30 | 74439.00 | 74440.00 | 2.00 | 3.00 |
| 31 | 74441.00 | 80147.00 | 5707.00 | 2.00 |
| 32 | 80148.00 | 80189.00 | 42.00 | 3.00 |
| 33 | 80190.00 | 80264.00 | 75.00 | 1.00 |
| 34 | 80265.00 | 86157.00 | 5893.00 | 2.00 |
| 35 | 86158.00 | 86592.00 | 435.00 | 1.00 |
| 36 | 86593.00 | 86605.00 | 13.00 | 3.00 |
| 37 | 86606.00 | 106174.00 | 19569.00 | 2.00 |
| 38 | 106175.00 | 106415.00 | 241.00 | 1.00 |
| 39 | 106416.00 | 106417.00 | 2.00 | 3.00 |
| 40 | 106418.00 | 116345.00 | 9928.00 | 2.00 |
| 41 | 116346.00 | 116631.00 | 286.00 | 1.00 |
| 42 | 116632.00 | 116691.00 | 60.00 | 3.00 |
| 43 | 116692.00 | 125951.00 | 9260.00 | 2.00 |
| 44 | 125952.00 | 125999.00 | 48.00 | 3.00 |
| 45 | 126000.00 | 126184.00 | 185.00 | 1.00 |
| 46 | 126185.00 | 126195.00 | 11.00 | 3.00 |
| 47 | 126196.00 | 144426.00 | 18231.00 | 2.00 |
| 48 | 144427.00 | 145380.00 | 954.00 | 1.00 |
| 49 | 145381.00 | 145393.00 | 13.00 | 3.00 |
| 50 | 145394.00 | 151739.00 | 6346.00 | 2.00 |
| 51 | 151740.00 | 151805.00 | 66.00 | 3.00 |
| 52 | 151806.00 | 154102.00 | 2297.00 | 1.00 |
| 53 | 154103.00 | 154114.00 | 12.00 | 3.00 |
| 54 | 154115.00 | 154664.00 | 550.00 | 2.00 |
| 55 | 154665.00 | 154729.00 | 65.00 | 3.00 |
| 56 | 154730.00 | 154923.00 | 194.00 | 2.00 |
| 57 | 154924.00 | 155072.00 | 149.00 | 3.00 |
| 58 | 155073.00 | 155717.00 | 645.00 | 1.00 |
| 59 | 155718.00 | 155775.00 | 58.00 | 3.00 |
| 60 | 155776.00 | 155938.00 | 163.00 | 2.00 |
| 61 | 155939.00 | 156069.00 | 131.00 | 1.00 |
| 62 | 156070.00 | 156130.00 | 61.00 | 3.00 |
| 63 | 156131.00 | 159741.00 | 3611.00 | 1.00 |
| 64 | 159742.00 | 159892.00 | 151.00 | 3.00 |
| 65 | 159893.00 | 160209.00 | 317.00 | 2.00 |
| 66 | 160210.00 | 160270.00 | 61.00 | 3.00 |
| 67 | 160271.00 | 160441.00 | 171.00 | 2.00 |
| 68 | 160442.00 | 160524.00 | 83.00 | 1.00 |
| 69 | 160525.00 | 160527.00 | 3.00 | 3.00 |
| 70 | 160528.00 | 160619.00 | 92.00 | 2.00 |
| 71 | 160620.00 | 160623.00 | 4.00 | 3.00 |
| 72 | 160624.00 | 161324.00 | 701.00 | 1.00 |
| 73 | 161325.00 | 161456.00 | 132.00 | 3.00 |
| 74 | 161457.00 | 162999.00 | 1543.00 | 1.00 |
| 75 | 163000.00 | 171019.00 | 8020.00 | 2.00 |
| 76 | 171020.00 | 171450.00 | 431.00 | 1.00 |
| 77 | 171451.00 | 171521.00 | 71.00 | 3.00 |
| 78 | 171522.00 | 171976.00 | 455.00 | 2.00 |
| 79 | 171977.00 | 172047.00 | 71.00 | 3.00 |
| 80 | 172048.00 | 176974.00 | 4927.00 | 2.00 |

TABLE 4 – Table récapitulative de la structure du chromosome de souris numéro 2. Les régions CpG+ (model = 1) versus les régions CpG- (model = 2). Le smoothing a été réalisé avec de valeurs de seuillage de s = 60 et r = 40, mais un certain marge de régions ambigües (model = 3) est encore visible.

| | beg | end | long | model |
|---|---|---|---|---|
| 1 | 1.00 | 4.00 | 4.00 | 3.00 |
| 2 | 5.00 | 22.00 | 18.00 | 3.00 |
| 3 | 23.00 | 8486.00 | 8464.00 | 2.00 |
| 4 | 8487.00 | 8535.00 | 49.00 | 3.00 |
| 5 | 8536.00 | 41656.00 | 33121.00 | 2.00 |
| 6 | 41657.00 | 41714.00 | 58.00 | 3.00 |
| 7 | 41715.00 | 41785.00 | 71.00 | 1.00 |
| 8 | 41786.00 | 41787.00 | 2.00 | 3.00 |
| 9 | 41788.00 | 67575.00 | 25788.00 | 2.00 |
| 10 | 67576.00 | 67644.00 | 69.00 | 1.00 |
| 11 | 67645.00 | 73026.00 | 5382.00 | 2.00 |
| 12 | 73027.00 | 73033.00 | 7.00 | 3.00 |
| 13 | 73034.00 | 73625.00 | 592.00 | 1.00 |
| 14 | 73626.00 | 73693.00 | 68.00 | 2.00 |
| 15 | 73694.00 | 73892.00 | 199.00 | 3.00 |
| 16 | 73893.00 | 74101.00 | 209.00 | 1.00 |
| 17 | 74102.00 | 74112.00 | 11.00 | 3.00 |
| 18 | 74113.00 | 74911.00 | 799.00 | 2.00 |
| 19 | 74912.00 | 74987.00 | 76.00 | 3.00 |
| 20 | 74988.00 | 75568.00 | 581.00 | 1.00 |
| 21 | 75569.00 | 77877.00 | 2309.00 | 2.00 |
| 22 | 77878.00 | 78049.00 | 172.00 | 1.00 |
| 23 | 78050.00 | 78118.00 | 69.00 | 3.00 |
| 24 | 78119.00 | 84219.00 | 6101.00 | 2.00 |
| 25 | 84220.00 | 84877.00 | 658.00 | 1.00 |
| 26 | 84878.00 | 84961.00 | 84.00 | 3.00 |
| 27 | 84962.00 | 106909.00 | 21948.00 | 2.00 |
| 28 | 106910.00 | 108242.00 | 1333.00 | 1.00 |
| 29 | 108243.00 | 108252.00 | 10.00 | 3.00 |
| 30 | 108253.00 | 113529.00 | 5277.00 | 2.00 |
| 31 | 113530.00 | 113576.00 | 47.00 | 3.00 |
| 32 | 113577.00 | 120726.00 | 7150.00 | 2.00 |
| 33 | 120727.00 | 120742.00 | 16.00 | 3.00 |
| 34 | 120743.00 | 121766.00 | 1024.00 | 1.00 |
| 35 | 121767.00 | 121797.00 | 31.00 | 3.00 |
| 36 | 121798.00 | 140452.00 | 18655.00 | 2.00 |

TABLE 5 – Table récapitulative de la structure du chromosome de souris numéro 3. Les régions CpG+ (model = 1) versus les régions CpG- (model = 2). Le smoothing a été réalisé avec de valeurs de seuillage de s = 60 et r = 40, mais un certain marge de régions ambigües (model = 3) est encore visible.

```
library(seqinr)
rerun = FALSE


#########################################################
### code chunk number 2: working (eval = FALSE)
#########################################################
## setwd("manuscrit")
##


#########################################################
### code chunk number 3: transition
#########################################################
transition <- function(file, # fichier
                       l_word = 1, # longueur des mots
                       n_seq = 1, # Nombre de sequences a analyser
                       log = TRUE,
                       type = "") {
  seq <-seqinr::read.fasta(file)
  Nseq <- length(seq)

  if (Nseq < n_seq) { # check if user want to input too many seq in the matrix learning
    stop(paste("n_seq is larger than the number of sequence in this file (", Nseq, " sequences for
    this file).", sep = ""))
  }

  l <- sapply(seq, length)
  if (l_word > min(l) | l_word >= 10) {
    stop("This is really too much for me, abort!!!")
  }

```

```
55    tmp <-seqinr::count(seq[[1]], l_word) + 1 # add 1 occurence to have at least 1 obs
56
57    cat(paste("      ========== Training model M",type,l_word," ==========        \n", sep = ""))
58    pb <- utils::txtProgressBar(min = 0, max = n_seq, style = 3)
59    for (i in 2:n_seq) {
60      utils::setTxtProgressBar(pb, i)
61
62      tmp <- tmp +seqinr::count(seq[[i]], l_word)
63    }
64    close(pb)
65
66    ## alternativ way, slower but prettier
67    # cat(paste("      ========== Training model M",type,l_word," ==========        \n", sep = "")
        )
68    # l_count = function(Seq,n = l_word){count(Seq,n)}
69    # tmp <- rowSums(sapply(seq,l_count))
70
71    if (l_word>1) {
72      i <- 1
73      wind = 4
74      for(j in 0:((length(tmp)/wind)-1)){
75        tmp[(1+wind*j):(wind+wind*j)] <- tmp[(1+wind*j):(wind+wind*j)] * 4^(i-1) / sum(tmp[(1+wind*j)
        :(wind+wind*j)] )
76      }
77      cat('      ========== Computing conditionnal probabilities ==========        \n')
78    } else {
79      tmp = tmp / sum(tmp)
80    }
81    # possibility to compute without log
82    if (log) {
83      tmp = log(tmp)
84    }
85    return(tmp)
86  }
87
88  mP <-transition(file = "../raw_data/mus_cpg_app.fa", n_seq = 1160, l_word = 2, log = F)
89
90  mP <- matrix(mP, byrow = TRUE, ncol = 4, dimnames = list(unique(substr(names(mP),1,1)),unique(substr
        (names(mP),1,1))))
91
92  matable <- xtable(x = mP , label = "tab:trans")
93  # Notez les doubles \\ n cessaires dans R, c'est la "double escape rule"
94  print(matable, file = "fig/tab_trans.tex",size = "tiny", NA.string = "NA",
95        table.placement = "!t",
96        floating = FALSE,
97        caption.placement="top",
98        include.rownames = TRUE,
99        include.colnames = TRUE,
100       latex.environments = "center")
101
102
103 ################################################
104 ### code chunk number 4: threshold
```

```r
##################################################
quality <- function(file , # fichier
                    pos_training = NULL, # file to train with for positive
                    neg_training = NULL, # file to train with for negative
                    trans_pos = NULL, #transition matrice pos
                    trans_neg = NULL, #transition matrice pos
                    l_word_pos = 1, # word lenght for transition table positif
                    l_word_neg = 1, # word length for transition table negatif
                    n_train = 1, # number of sequences to train with
                    n_seq = 1, # number of sequences to analyse
                    quiet = FALSE
){
  if(is.null(c(trans_pos,trans_neg))){
    trans_pos <-transition(file = pos_training, n_seq = n_train, l_word = l_word_pos, type ="+")
    trans_neg <- transition(file = neg_training, n_seq = n_train, l_word = l_word_neg, type = "-")
  }

  seq <-seqinr::read.fasta(file)
  Nseq <- length(seq)

  if(Nseq < n_seq){ # check if user want to input too many seq in the matrix learning
    stop(paste("n_seq is larger than the number of sequence in this file (",Nseq," sequences for
    this file).", sep = ""))
  }

  result <- data.frame(VP = rep(FALSE,n_seq),
                       FN = rep(TRUE,n_seq),
                       pos = rep(NA,n_seq),
                       neg = rep(NA,n_seq)
  )

  p_init_pos = log(1/4^l_word_pos)
  p_init_neg = log(1/4^l_word_neg)

  if(!quiet) {cat('      ========== Computing sensi and speci for the test sequences ==========
    \n')
    pb <- utils::txtProgressBar(min = 0, max = n_seq, style = 3)}
  for(i in 1:n_seq){
    if(!quiet) utils::setTxtProgressBar(pb, i)
    n_word_pos <-seqinr::count(seq[[i]], l_word_pos)
    n_word_neg <-seqinr::count(seq[[i]], l_word_neg)
    result$pos[i] <- p_init_pos + sum( trans_pos * n_word_pos )
    result$neg[i] <- p_init_neg + sum( trans_neg * n_word_neg )
    if(result$pos[i] > result$neg[i]){
      result$VP[i] <- TRUE ; result$FN[i] <- FALSE
    } else {
      result$VP[i] <- FALSE ; result$FN[i] <- TRUE
    }
  }
  if(!quiet) close(pb)

  tmp <- colSums(result[,1:2])
  return(tmp)
```

```
}

threshold <- function(pos_test, # fichier
                      neg_test,
                      pos_training, # file to train with for positive
                      neg_training, # file to train with for negative
                      pos_seq = c(1:2),
                      neg_seq = c(1:2),
                      n_train = 1, # number of sequences to train with
                      n_seq = 1 # number of sequences to analyse
){

  # choix =utils::menu(c("yes","no"),
  #              title = "You will launch long computation, do you wish to procede further ?")
  # if (choix ==1) cat("\n      ========== Go take a good coffee ==========       \n\n")
  # if (choix ==2) stop("You stopped the computations")


  trans_pos <- list()
  trans_neg <- list()

  cat('\n      ========== Training modeles ==========       \n\n')
  for(i in pos_seq){
    trans_pos[[i]] <- transition(file = pos_training, n_seq = n_train, l_word = i, type = "+")
  }
  cat('\n')
  for(j in neg_seq){
    trans_neg[[j]] <- transition(file = neg_training, n_seq = n_train, l_word = j, type = "-")
  }

  cat('\n      ========== Training modeles ==========       \n\n')
  sensi <- speci <- matrix(rep(0,length(pos_seq)*length(neg_seq)),ncol = length(pos_seq),nrow =
    length(neg_seq))
  for(i in pos_seq){
    for(j in neg_seq){
      cat('\n')
      cat(paste("      ========== Model M+ (",i,"/",j,") ==========       \n", sep = ""))
      pos <- quality(file = pos_test, # fichier
                     trans_pos = trans_pos[[i]], #transition matrice pos
                     trans_neg = trans_neg[[j]], #transition matrice neg
                     l_word_pos = i, # transition table positif
                     l_word_neg = j, # transition table negatif
                     n_train = n_train, # number of sequences to train with
                     n_seq = n_seq, # number of sequences to analyse
                     quiet = TRUE
      )
      cat(paste("      ========== Model M- (",i,"/",j,") ==========       \n", sep = ""))
      neg <- quality(file = neg_test, # fichier
                     trans_pos = trans_pos[[i]], #transition matrice pos
                     trans_neg = trans_neg[[j]], #transition matrice neg
                     l_word_pos = i, # transition table positif
                     l_word_neg = j, # transition table negatif
                     n_train = n_train, # number of sequences to train with
```

```
                    n_seq = n_seq, # number of sequences to analyse
                    quiet = TRUE
      )

      sensi[i,j] <- pos[1] / sum(pos)
      speci[i,j] <- neg[2] / sum(neg)

    }
  }

  cat(paste("      ========== Come back from your coffee ==========        \n", sep = ""))
  colnames(sensi) <- colnames(speci) <- paste("-",neg_seq, sep="")
  rownames(sensi) <- rownames(speci) <- paste("+",pos_seq, sep="")

  final <-list(sensi,speci) ; names(final) = c("sensi","speci")

  return(final)
}



#######################################################
### code chunk number 5: seuil
#######################################################
if("final.RData" %in% list.files("fig/") && !rerun){
  load("fig/final.RData")
}else{
  final <- threshold("../raw_data/mus_cpg_test.fa", # fichier
  "../raw_data/mus_tem_test.fa",
  "../raw_data/mus_cpg_app.fa", # file to train with for positive
  "../raw_data/mus_tem_app.fa", # file to train with for negative
  pos_seq = c(1:6),
  neg_seq = c(1:6),
  n_train = 1160, # number of sequences to train with
  n_seq = 1163 # number of sequences to analyse
)
save(final,file = "fig/final.RData")
}

table <- cbind(melt(final$sensi), melt(final$speci)[, 3])
colnames(table) <- c("M", "m", "Sensi", "Speci")
table$score <- table$Sensi + table$Speci

visual <- ggplot(table, aes(M, m)) +
  geom_raster(aes(fill = score), hjust = 0.5, vjust = 0.5, interpolate = FALSE) +
  geom_contour(aes(z = score)) + xlab("CpG+") + ylab("CpG-")
visual
ggsave('visual.pdf', plot = visual, device = "pdf", path = 'fig/',scale = 3, width = 7, height = 4,
    units = "cm", limitsize = T)
rm(visual)

table[which(table$tot == max(table$tot)),]

```

```
260
261 ######################################################
262 ### code chunk number 6: trans_mod
263 ######################################################
264   l_c <- 1 / 1000
265   l_nc <- 1 / 125000
266
267    trans_mod <- log(matrix(c(
268     1 - l_c, l_nc,
269     l_c, 1 - l_nc
270   ),
271   ncol = 2, nrow = 2 , dimnames = list(c("M+","M-"),c("M+","M-"))
272   ))
273
274 matable <- xtable(x = trans_mod , label = "tab:trans_mod")
275 # Notez les doubles \\ n cessaires dans R, c'est la "double escape rule"
276 print(matable, file = "fig/trans_mod.tex",size = "tiny", NA.string = "NA",
277       table.placement = "!t",
278       floating = FALSE,
279       caption.placement="top",
280       include.rownames = TRUE,
281       include.colnames = TRUE,
282       latex.environments = "center")
283
284
285 ######################################################
286 ### code chunk number 7: viterbi
287 ######################################################
288 viterbi <- function(file ,
289                     pos_training = "../raw_data/mus_cpg_app.fa", # file to train with for positive
290                     neg_training = "../raw_data/mus_tem_app.fa", # file to train with for negative
291                     l_word_pos = 1,
292                     l_word_neg = 1,
293                     n_train = 1160,
294                     n_ana = 1,
295                     l_c = 1000, # length coding
296                     l_nc = 125000 # length non-coding
297 ) {
298   trans_pos <- transition(file = pos_training , n_seq = n_train , l_word = l_word_pos, type = "+")
299   trans_neg <- transition(file = neg_training , n_seq = n_train , l_word = l_word_neg, type = "-")
300
301   seq <- seqinr::read.fasta(file)
302   if (n_ana > 1) stop("Analysis␣for␣multiple␣files␣is␣not␣implemented␣yet")
303
304   raw_seq <- seq[[1]]
305   long <- length(raw_seq)
306
307   beg <- max(l_word_pos, l_word_neg)
308
309   # compute p_initial and transition matrix for markov
310   pos_init <- neg_init <- log(0.5)
311   l_c <- 1 / l_c
312   l_nc <- 1 / l_nc
```

```
313
314 trans_mod <- log(matrix(c(
315   1 - l_c, l_nc,
316   l_c, 1 - l_nc
317 ),
318 ncol = 2, nrow = 2
319 ))
320 colnames(trans_mod) <- rownames(trans_mod) <- c("c", "nc")
321
322 # initialisation
323 ncol <- 7
324 proba <- matrix(rep(NA, long * ncol), ncol = ncol)
325 colnames(proba) <- c("M+", "M-", "model", "length", "rep_length", "begin", "end")
326
327 # time explode if it is not a matrix anymore !!!
328 # proba <- as.data.frame(proba)
329
330 # old version is v2 takes too long
331 # v1 = function(){
332 # trans_pos[which(names(trans_pos)==paste(raw_seq[(beg-l_word_pos+1):beg],collapse = ""))]
333 # }
334 # v2 = function(){
335 # min(count(raw_seq[(beg-l_word_pos+1):beg], l_word_pos) * trans_pos)
336 # }
337 # system.time(v1())
338 # system.time(v2())
339
340 # compute first base and initiate Viterbi
341 proba[beg, "M+"] <- trans_pos[which(names(trans_pos) == paste(raw_seq[(beg - l_word_pos + 1):beg],
      collapse = ""))] + pos_init
342 proba[beg, "M-"] <- trans_neg[which(names(trans_neg) == paste(raw_seq[(beg - l_word_neg + 1):beg],
      collapse = ""))] + neg_init
343 if (proba[beg, "M+"] > proba[beg, "M-"]) {
344   proba[beg, "model"] <- 1
345 } else {
346   proba[beg, "model"] <- 2
347 }
348 proba[beg, c("length", "rep_length")] <- 1
349 tmp <- proba[beg, c(6, 7)] <- beg
350 proba[1:beg - 1, "rep_length"] <- beg - 1
351 proba[1:beg - 1, "model"] <- 3
352 proba[1, "begin"] <- 1
353 proba[beg - 1, c(4, 7)] <- beg - 1
354
355 beg <- beg + 1
356 cat("\n      ========== Viterbi is running ==========       \n\n")
357 pb <- utils::txtProgressBar(min = beg, max = long, style = 3)
358 for (i in beg:long) {
359   utils::setTxtProgressBar(pb, i)
360
361   # proba d'avoir la base sous M
362   pM <- trans_pos[which(names(trans_pos) == paste(raw_seq[(i - l_word_pos + 1):i], collapse = ""))
    ] + max(
```

```
      proba[i - 1, "M+"] + trans_mod[1, 1],
      proba[i - 1, "M-"] + trans_mod[2, 1]
    )

    # proba d'avoir la base sous m
    pm <- trans_neg[which(names(trans_neg) == paste(raw_seq[(i - l_word_neg + 1):i], collapse = ""))
    ] + max(
      proba[i - 1, "M-"] + trans_mod[2, 2],
      proba[i - 1, "M+"] + trans_mod[1, 2]
    )

    proba[i, "M+"] <- pM
    proba[i, "M-"] <- pm

    if (proba[i, "M+"] > proba[i, 2]) {
      proba[i, "model"] <- 1
    } else {
      proba[i, "model"] <- 2
    }
    # length information
    if (proba[i, "model"] == proba[i - 1, "model"]) {
      proba[i, "length"] <- proba[i - 1, "length"] + 1 # increase part length
      proba[i - 1, c(4, 7)] <- NA # erase length in previous ligne
    } else {
      proba[i, "length"] <- 1 # initiate new part length
      proba[i - 1, "end"] <- i - 1 # put end value of precedent part
      proba[c(tmp:(i - 1)), "rep_length"] <- proba[i - 1, "length"] # rep value of length for
    precedent part
      proba[i, "begin"] <- tmp <- i # put begin value of the actual part
    }
  }
  close(pb)

  # closing table
  proba[i, "end"] <- i # put end value of precedent part
  proba[c(tmp:(i)), "rep_length"] <- proba[i, "length"] # rep value of length for precedent part

  # add a column for the line number
  proba <- cbind(c(1:dim(proba)[1]), proba)
  colnames(proba)[1] <- "n"

  return(proba)
}


#######################################################
### code chunk number 8: smoothing
#######################################################
smoothing <- function(seq,
                      l_word_pos = 5,
                      l_word_neg = 4,
                      smooth_win = 10,
                      reject_win = 1) {
```

```
414 beg <- max(l_word_pos, l_word_neg)
415
416 # finding ambiguous regions which are shorter than a certain windows
417 cat("      ========== Smoothing ==========      \n")
418 seq <- cbind(seq, seq[, 4])
419 colnames(seq)[9] <- c("smoothed")
420 seq[which(seq[, "rep_length"] <= smooth_win), "smoothed"] <- 3
421
422 # old version is v1 takes too long
423 # v1 = function(){
424 #   cat("      ========== Smoothing boucle ==========      \n")
425 #   pb <- utils::txtProgressBar(min = 1, max = max(seq[, 1]), style = 3)
426 #   for (i in 1:dim(seq)[1]) {
427 #     utils::setTxtProgressBar(pb, i)
428 #     if (seq[i, 6] <= smooth_win) {
429 #       seq[i, "smoothed"] <- 3
430 #     }
431 #   }
432 #   close(pb)
433 # }
434 # v2 = function(){
435 #   seq[which(seq[, 6] <= smooth_win), "smoothed"] <- 3}
436 # system.time(v1())
437 # system.time(v2())
438
439 # unified ambiguous regions
440 seq <- cbind(seq, seq[, c(5:8)])
441 colnames(seq)[10:13] <- paste("S_", colnames(seq[, c(10:13)]), sep = "")
442
443 seq[beg, "S_length"] <- 1
444 tmp <- seq[beg, c("S_begin", "S_end")] <- beg
445 seq[-c(1:beg), c(10:13)] <- NA
446
447 beg <- beg + 1
448 cat("\n      ========== Smoothing length ==========      \n")
449 pb <- utils::txtProgressBar(min = beg - 1, max = dim(seq)[1], style = 3)
450 for (i in beg:dim(seq)[1]) {
451   utils::setTxtProgressBar(pb, i)
452   if (seq[i, "smoothed"] == seq[i - 1, "smoothed"]) {
453     seq[i, "S_length"] <- seq[i - 1, "S_length"] + 1 # increase part length
454     seq[i - 1, c("S_length", "S_end")] <- NA # erase length in previous ligne
455   } else {
456     seq[i, "S_length"] <- 1 # initiate new part length
457     seq[i - 1, "S_end"] <- i - 1 # put end value of precedent part
458     seq[c(tmp:(i - 1)), "S_rep_length"] <- seq[i - 1, "S_length"] # rep value of length for
    precedent part
459     seq[i, "S_begin"] <- tmp <- i # put begin value of the actual part
460   }
461 }
462 close(pb)
463
464 # closing table
465 seq[i, 13] <- i # put end value of precedent part
```

```
  seq[c(tmp:(i)), 11] <- seq[i, 10] # rep value of length for precedent part

  # to reject some region and put arbitrary models on them
  if(reject_win > 1){
    colnames(seq)[10:13] <- paste("R_", colnames(seq[, c(10:13)]), sep = "")

    head(seq)

    cat("      ========== Rejecting ==========      \n")
    # finding regions of length < reject_win between tho regions of same model. Changing the model to surrounding
    solo_l <-seq[which(seq[,"R_S_length"] %in% unique(seq[, "R_S_rep_length"])),c("smoothed","R_S_rep_length")]
    for(i in 2:(dim(solo_l)[1]-1)){
      if(solo_l[i-1,1]==solo_l[i+1,1] && solo_l[i,2] < reject_win && solo_l[i,1] == 3) {solo_l[i,1] <- solo_l[i-1,1]}
    }

    seq[,"smoothed"] <- rep(solo_l[,1],solo_l[,2])

    beg <- beg - 1

    seq[beg, "R_S_length"] <- 1
    tmp <- seq[beg, c("R_S_begin", "R_S_end")] <- beg
    seq[-c(1:beg), c(10:13)] <- NA

    beg <- beg + 1
    seq[-c(1:beg), c(10:13)] <- NA
    cat("\n      ========== Smoothing length after reject ==========      \n")
    pb <- utils::txtProgressBar(min = beg - 1, max = dim(seq)[1], style = 3)
    for (i in beg:dim(seq)[1]) {
      utils::setTxtProgressBar(pb, i)
      if (seq[i, "smoothed"] == seq[i - 1, "smoothed"]) {
        seq[i, "R_S_length"] <- seq[i - 1, "R_S_length"] + 1 # increase part length
        seq[i - 1, c("R_S_length", "R_S_end")] <- NA # erase length in previous ligne
      } else {
        seq[i, "R_S_length"] <- 1 # initiate new part length
        seq[i - 1, "R_S_end"] <- i - 1 # put end value of precedent part
        seq[c(tmp:(i - 1)), "R_S_rep_length"] <- seq[i - 1, "R_S_length"] # rep value of length for precedent part
        seq[i, "R_S_begin"] <- tmp <- i # put begin value of the actual part
      }
    }
    close(pb)

    # closing table
    seq[i, 13] <- i # put end value of precedent part
    seq[c(tmp:(i)), 11] <- seq[i, 10] # rep value of length for precedent part

  }

  return(seq)
}
```

```
515
516
517 ######################################################
518 ### code chunk number 9: plot_resume
519 ######################################################
520 graph <- function(seq,colors = c("blue","red","green"),nrow = 3,ycol = 4){
521   par(mfrow = c(nrow,1), mar = c(3,2,1,1))
522   # c(5, 4, 4, 2)
523
524   long <- max(seq[,1])/nrow
525   for(i in 0:(nrow-1)){
526     plot(x = seq[((1+long*i):(long+long*i)),1],
527          y = seq[((1+long*i):(long+long*i)),ycol],
528          col = colors[seq[((1+long*i):(long+long*i)),9]], xlab = "␣", ylab = "␣")
529   }
530   par(mfrow = c(1,1))
531 }
532
533 resume <- function(seq){
534   beg <- seq[which(!is.na(seq[,12])),12]
535   end <- seq[which(!is.na(seq[,13])),13]
536   long <- seq[which(!is.na(seq[,10])),10]
537   model <- seq[which(!is.na(seq[,10])),9]
538   length(beg) ; length((end)) ; length(long) ; length(model)
539
540   table <- data.frame(beg,end,long,model)
541   return(table)
542 }
543
544
545 ######################################################
546 ### code chunk number 10: mus1
547 ######################################################
548 if("mus1.RData" %in% list.files("fig/") && !rerun){
549   load("fig/mus1.RData")
550 }else{
551 mus1 <- viterbi(file = "../raw_data/mus1.fa",
552   l_word_pos = 5,
553   l_word_neg = 4
554 )
555 save(mus1,file = "fig/mus1.RData")
556 }
557
558 mus1 <- smoothing(mus1,5,4,60, 40)
559
560 jpeg("fig/mus1_s.jpg", width = 836, height = 496)
561 # 2. Create the plot
562 graph(mus1,nrow = 3, ycol = 9)
563 # 3. Close the file
564 dev.off()
565
566 mus1_t <- resume(mus1)
567
```

```
568
569
570 ##################################################
571 ### code chunk number 11: mus2
572 ##################################################
573 if("mus2.RData" %in% list.files("fig/") && !rerun){
574   load("fig/mus2.RData")
575 }else{
576 mus2 <- viterbi(file = "../raw_data/mus2.fa",
577   l_word_pos = 5,
578   l_word_neg = 4
579 )
580 save(mus2,file = "fig/mus2.RData")
581 }
582
583 mus2 <- smoothing(mus2,5,4,60, 40)
584
585 jpeg("fig/mus2_s.jpg", width = 836, height = 496)
586 # 2. Create the plot
587 graph(mus2,nrow = 3, ycol = 9)
588 # 3. Close the file
589 dev.off()
590
591 mus2_t <- resume(mus2)
592
593
594
595 ##################################################
596 ### code chunk number 12: mus3
597 ##################################################
598 if("mus3.RData" %in% list.files("fig/") && !rerun){
599   load("fig/mus3.RData")
600 }else{
601 mus3 <- viterbi(file = "../raw_data/mus3.fa",
602   l_word_pos = 5,
603   l_word_neg = 4
604 )
605 save(mus3,file = "fig/mus3.RData")
606 }
607
608 mus3 <- smoothing(mus3,5,4,60, 40)
609
610 jpeg("fig/mus3_s.jpg", width = 836, height = 496)
611 # 2. Create the plot
612 graph(mus3,nrow = 3, ycol = 9)
613 # 3. Close the file
614 dev.off()
615
616 mus3_t <- resume(mus3)
617
618
619 ##################################################
620 ### code chunk number 13: github_install (eval = FALSE)
```

```
##################################################
## # NOT RUN
## library(devtools)
## install_github("gowachin/BeeMarkov")
## library(BeeMarkov)
```