Projet : Hidden Markov Model

M2 Ingénierie Mathématique - Université Paris Descartes

Nadia Ouhssaine

10 novembre 2019

Contents

1 Introduction

Nous allons mener une réflexion autour de l'article de Leutenegger et al. (2003) qui étudie une nouvelle manière d'estimer le coefficient de consanguinité d'un individu à partir de données génomiques. Pour cela, les auteurs de l'article ont développé une méthode utilisant d'une part les chaînes de Markov cachées pour représenter le statut IBD par les dépendances de marqueurs, et d'autre part l'estimation du maximum de vraisemblance. Dans le cadre de notre cours, nous utiliserons l'algorithme Expectation-Maximization de Dempster, Laird, and Rubin (1977) qui est un algorithme itératif permettant de trouver les paramètres du maximum de vraisemblance d'un modèle probabiliste lorsque ce dernier dépend de variables latentes non observables.

Pour aborder l'étude de cet article, nous allons dans un premier temps définir le contexte de l'étude et le modèle utilisé, puis nous allons simuler des données et implémenter une méthode de maximisation de la vraisemblance dans le contexte des chaînes de Markov cachées (HMM), finalement nous nous engagerons dans des expérimentations afin d'en tirer des conclusions quant à l'efficacité de cet algorithme.

2 Contexte

Le but de cette étude est d'établir le coefficient de consanguinité d'un individu sans la connaissance de leur généalogie et leurs relations parentales.

Dans un premier temps, rappelons que le coefficient de consanguinité sert principalement à établir les risques d'être atteint de maladies génétiques (récessives). En effet, scientifiquement, le coefficient de consanguinité est la probabilité que les deux allèles d'un même individu à un locus donné soient Identical By Descent (IBD), c'est à dire que ces allèles soient issu d'un apparentement entre deux individus qui a eu pour conséquence que ces individus aient chacun reçu en un locus, une copie du même allèle présent chez un ancêtre commun. Nous nous rendons donc compte qu'il est difficile d'estimer correctement le coefficient de consanguinité si nous ne connaissons pas assez bien les généalogies de l'individu.

Ainsi, les chercheurs s'attèlent à trouver une méthode statistique afin d'estimer le coefficient de consanguinité sans avoir à connaître les relations parentales. Pour cela, l'utilisation de données génomique est plus particulièrement de marqueurs individuels peut s'avérer très utile. Rappelons qu'un marqueur génétique est un gène ou une séquence d'ADN aisément détectable grâce à un emplacement connu sur un chromosome. Ainsi, grâce à ce type de données le processus d'IBD le long du génome de l'individu peut être caractériser par un modèle de Markov cachées. En effet, la particularité d'un modèle de Markov cachées est qu'il peut représenter une généralisation des modèles de mélange en introduisant une dépendance dans les classes latentes à travers une chaîne de Markov.

3 Modèle

Définissons le modèle utilisant les chaînes de Markov cachées et qui nous permettra d'établir le coefficient de consanguinité f par la maximisation de la vraisemblance.

$\forall$ marqueur $i \in \{1, ..., n\}$, avec n=50000 :

- **Données initiales :**

t = Distance entre chaque marqueur

ε = Taux d'erreur de génotypage

a = Taux de recombinaison par cM (à estimer)

f = Coefficient de consanguinité (à estimer)

- **Variables latentes :**

→ Dans notre modèle , la variable non observée S_i est le ***statut IBD*** au marqueur i. Nous avons :

$$S_i = \begin{cases} 1 & \text{si les 2 allèles sont IBD} \\ 0 & \text{sinon} \end{cases}$$

→ D'après l'approximation de Markov, un statut IBD à un locus i ne dépend que du statut IBD des loci adjacents, ainsi on a :

$\mathbb{P}(S) = \mathbb{P}(S_1) \prod_{i=2}^{n} \mathbb{P}(S_i|S_{\text{i-1}}) = \mu_1(S_1) \prod_{i=2}^{n} \pi_i(S_{\text{i-1}}, S_i)$

où μ_1 est la loi initiale tel que : $\mu_1(0) = \mathbb{P}(S_1 = 0) = (1-f)$ et $\mu_1(1) = \mathbb{P}(S_1 = 1) = f$.

et π_i est la matrice de transition des IBD $\forall i \in \{2, ..., n\}$ tel que :

$\pi_i(1,1) = \mathbb{P}(S_i = 1|S_{\text{i-1}} = 1) = (1 - e^{\text{-at}})f + e^{\text{-at}}$

$\pi_i(1,0) = \mathbb{P}(S_i = 0|S_{\text{i-1}} = 1) = (1 - e^{\text{-at}})(1-f)$

$\pi_i(0,1) = \mathbb{P}(S_i = 1|S_{\text{i-1}} = 0) = (1 - e^{\text{-at}})f$

$\pi_i(0,0) = \mathbb{P}(S_i = 0|S_{\text{i-1}} = 0) = (1 - e^{\text{-at}})(1-f) + e^{\text{-at}}$

où t = distance génétique entre deux loci/marqueur

et a = taux de recombinaison par cM

Rappelons que le terme $(1 - e^{\text{-at}})$ correspond à la probabilité d'avoir une recombinaison des allèles. Le terme f correspond à la probabilité que les allèles soient IBD. A contrario, $e^{\text{-at}}$ correspond à la probabilité de ne pas avoir de recombinaison des allèles et $(1-f)$ à la probabilité que les allèles ne soient pas IBD. Ainsi les probabilités de la matrice π_i s'explique facilement. Par exemple, pour $\pi_i(0,0)$ nous avons que la probabilité de conserver le statut IBD = 0 (ie les allèles ne sont pas IBD) est la probabilité de ne pas avoir de recombinaison des allèles + la probabilité d'avoir une recombinaison des allèles et que ces allèles ne soient pas IBD. Les autres probabilités de la matrice π_i s'expliquent de la même manière.

- **Variable observée :**

→ Dans notre modèle, la variable observée X_i est le ***génotype*** au marqueur i. Nous avons :

$X_i \in \{RR, RV, VV\}$ tel que :

$\mathbb{P}(X_i = RR|S_i = 0) = p_{\text{R}}^2$

$\mathbb{P}(X_i = VV|S_i = 0) = p_{\text{V}}^2$

$\mathbb{P}(X_i = RV|S_i = 0) = 2 \times p_{\text{R}} \times p_{\text{V}}$

$\mathbb{P}(X_i = RV|S_i = 1) = 2 \times \varepsilon \times p_{\text{R}} \times p_{\text{V}}$

$\mathbb{P}(X_i = RR|S_i = 1) = (1-\varepsilon) \times p_{\text{R}} + \varepsilon \times p_{\text{R}}^2$

$\mathbb{P}(X_i = VV|S_i = 1) = (1-\varepsilon) \times p_{\text{V}} + \varepsilon \times p_{\text{V}}^2$

où ε = taux d'erreur de génotypage

et p_{R} = probabilité de l'allèle R au locus ; p_{V} = probabilité de l'allèle V au locus.

→ Nous pouvons maintenant définir notre modèle comme ceci :

$$\mathbb{P}(X, S|\theta) = \mathbb{P}(S|\theta)\mathbb{P}(X|S;\theta) = \underbrace{\mathbb{P}(S_1) \prod_{i=2}^{n} \mathbb{P}(S_i|S_{\text{i-1}})}_{\text{Chaînes de Markov}} \underbrace{\prod_{i=1}^{n} \mathbb{P}(X_i|S_i)}_{\text{Mélange}}$$

où $\theta = (f, a)$

4 Simulations

Nous allons commencer à simuler la matrice π_i de transition des IBD et les probabilités $\mathbb{P}(X_i|S_i)$. À partir de ces derniers, nous simulerons les statuts IBD S_i puis nous terminerons par la simulation des génotypes X_i.

4.1 Simulation de π_i et $\mathbb{P}(X_i|S_i)$

Dans un premier temps, nous fixons à l'aide des données de l'article la taille t entre chaque marqueur, le vrai coefficient de consanguinité f, le taux d'erreur de génotypage ε et le taux de recombinaison a. Puis nous allons simuler la probabilité p_V que l'allèle soit variant au locus i. Pour cela, nous faisons une simulation de la loi $\mathcal{U}[0.05,0.49]$. De cette simulation est aussi déduit $p_R = 1 - p_V$.
Nous pourrons finalement construire à partir de ces derniers la loi initiale μ_1, notre matrice de transition des IBD π_i et les probabilités $\mathbb{P}(X_i|S_i)$.

```
set.seed(1234)
n=50000
# parametres

p=runif(n,min=0.05,max=0.49) # p_V différent à chaque locus

t=1.67 # distance entre chaque marqueur en cM

f=1/16 # First cousin

a=0.09 # choix arbitraire

eps=0.05 # taux d'erreur genotypage

# loi initiale
mu1=c(1-f,f)

# transition (la même pour tous les loci)
pi=matrix(NA,2,2)
rownames(pi)=colnames(pi)=c("IBD=0","IBD=1")
pi[1,1]=(1-exp(-a*t))*(1-f)+exp(-a*t) # IBD=0 -> IBD=0
pi[1,2]=(1-exp(-a*t))*f # IBD=0 -> IBD=1
pi[2,1]=(1-exp(-a*t))*(1-f) # IBD=1 -> IBD=0
pi[2,2]=(1-exp(-a*t))*f+exp(-a*t) # IBD=1 -> IBD=1
apply(pi,1,sum) # verification
```

```
## IBD=0 IBD=1
##     1     1
```

```
# emission (differente pour chaque locus)
e=array(NA,dim=c(n,3,2),
        dimnames=list(paste0("i=",1:n),
                      c("RR","RV","VV"),
                      c("IBD=0","IBD=1")))

e[,"RR","IBD=0"]=(1-p)^2
e[,"RV","IBD=0"]=2*p*(1-p)
```

```r
e[,"VV","IBD=0"]=p^2
e[,"RR","IBD=1"]=(1-eps)*(1-p)+eps*(1-p)^2
e[,"RV","IBD=1"]=eps*2*p*(1-p)
e[,"VV","IBD=1"]=(1-eps)*p+eps*p^2

#Vérification
head(apply(e,c(1,3),sum))
```

```
##      IBD=0 IBD=1
## i=1      1     1
## i=2      1     1
## i=3      1     1
## i=4      1     1
## i=5      1     1
## i=6      1     1
```

Afin de vérifier que notre matrice de transition est bien définit, nous sommons à l'aide de la fonction *apply* les lignes de notre matrice de transition (ie nous sommons $\pi_i(0,k)$ et nous sommons $\pi_i(1,k)$ pour $k \in \{0,1\}$), nous trouvons un résultat égal à 1 qui confirme que la matrice est bien définit.

De même pour la matrice représentant les probabilités $\mathbb{P}(X_i|S_i)$, nous vérifions à l'aide de la fonction *apply*, qu'avec nos probabilités p_V simulées pour chaque locus, nous avons bien une cohérence (ie en sommant les $\mathbb{P}(X_i = k|S_i = 0)$, nous trouvons un résultat égal à 1 et en sommant $\mathbb{P}(X_i = k|S_i = 1)$, nous trouvons également un résultat égal à 1).

4.2 Simulations des S_i

Nous allons maintenant simuler les variables latentes par un tirage aléatoire selon la loi initiale μ_1 pour S_1 et par un tirage aléatoire selon les probabilités de la matrice π_i pour S_i $\forall i \in \{2, ..., n\}$.

```r
set.seed(1234)
s=rep(NA,n)
s[1]=sample(0:1,prob=mu1,size=1)
for (i in 2:n)
  s[i]=sample(0:1,prob=pi[s[i-1]+1,],size=1)
plot(s,t='s',xlim=c(0,250),xlab="Locus/marqueur i", ylab="Statut IBD")
```

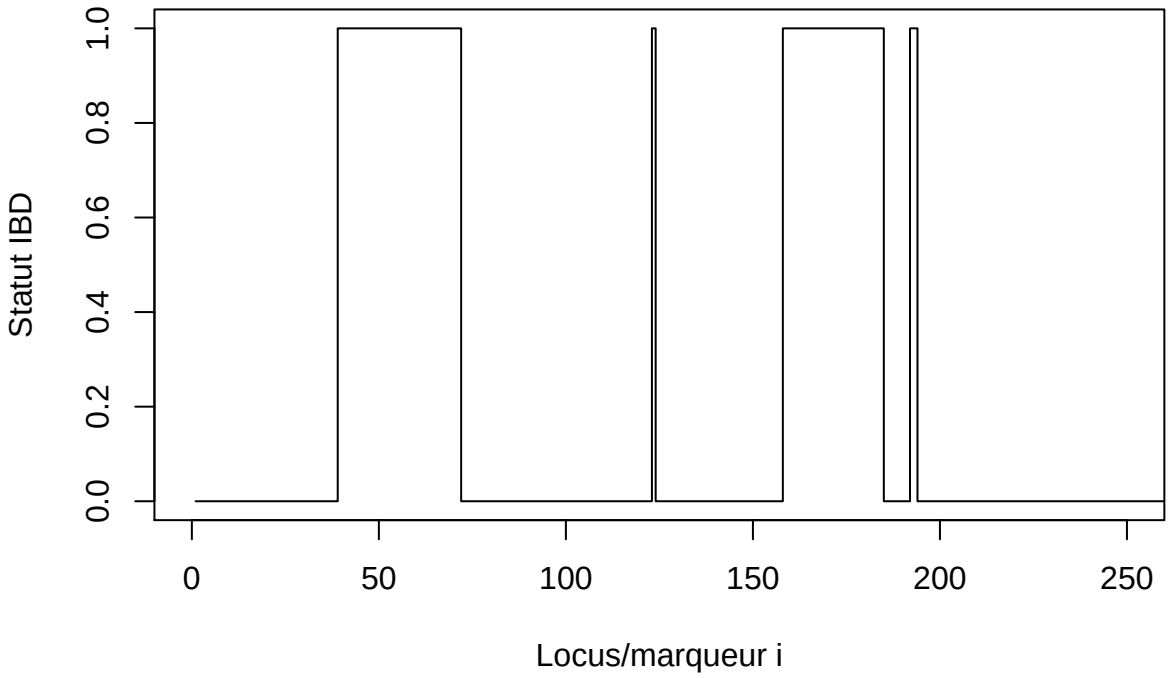

Figure 1: Graphique en marche d'escalier représentant les transitions du statut IBD pour nos données de marqueurs

La Figure 1, ci-dessus, représente les transitions entre le statut IBD : "les 2 allèles sont IBD" et le statut IBD : "les 2 allèles ne sont pas IBD" obtenu lorsque la variable aléatoire S passe de l'état 0 à l'état 1 ou inversement.

Nous remarquons, par exemple, qu'au locus n°40 jusqu'au locus n°60 environ, le statut IBD est passé à 1. C'est à dire que les allèles entre le locus n°40 et le locus n°60 sont IBD. Les allèles des loci précédent le locus n°40 n'étaient pas IBD et les allèles de ceux suivant le locus n°40 jusqu'à la prochaine marche d'escalier ne le seront pas non plus. Nous observons de la même manière 4 marches d'escaliers au total qui représentent les différentes transitions du statut IBD dans nos données. Nous pouvons également faire la remarque que, le plus souvent, le statut IBD est à l'état 0, c'est à dire que les allèles des loci sont plus souvent non IBD.

4.3 Simulations des X_i

Finalement, nous simulons la variable observée X_i par un tirage aléatoire selon les probabilités $\mathbb{P}(X_i|S_i)$ précédemment construites.

```
x=rep(NA,n)

for (i in 1:n)
  x[i]=sample(c("RR","RV","VV"),size=1,prob=e[i,,s[i]+1])
```

```r
table(x)
```

```
## x
##    RR    RV    VV
## 27903 17076  5021
```

```r
table(x[s==0])
```

```
##
##    RR    RV    VV
## 25664 17024  4198
```

```r
table(x[s==1])
```

```
##
##   RR   RV   VV
## 2239   52  823
```

```r
table(x[s==0])/sum(s==0)
```

```
##
##          RR          RV          VV
## 0.54737022 0.36309346 0.08953632
```

```r
table(x[s==1])/sum(s==1)
```

```
##
##          RR          RV          VV
## 0.71901092 0.01669878 0.26429030
```

Les tables ci-dessous, nous permettent d'avoir un petit aperçu de la répartition des génotypes RR, RV et VV. Dans un premier temps, nous remarquons que plus de la moitié des loci ont le génotype RR, un tiers des loci ont le génotypes RV et le reste des loci (environ 1/10) ont le génotype VV. C'est une répartition cohérente au vue de notre modèle.

Nous pouvons également observer la répartion des génotypes en fonction du statut IBD. Pour les loci dont les allèles ne sont pas IBD, nous faisons les mêmes remarques que précédement. En revanche pour les loci dont les allèles sont IBD, nous observons cette fois moins de génotypes RV que de génotypes VV. Ceci est une observation tout à fait cohérente avec la notion d'allèles IBD.

5 Estimation par MLE en supposant S observée

Nous venons de simuler nos données. Le but est maintenant d'estimer par le maximum de vraisemblance le paramètre $\theta = (a, f)$.

Etant donné que le paramètre S est une variable latente, l'utilisation de l'algorithme EM peut être une solution à l'estimation. Néanmoins, une maximisation explicite de $Q(\theta|\theta^{\text{old}})$ en déterminant l'expression du paramètre θ pour laquelle le gradient de la vraisemblance s'annule ne semble pas envisageable. L'article suggère d'utiliser Gemini qui est un programme informatique pour l'optimisation de fonctions non linéaires

générales. Quant à nous, nous allons opter pour une maximisation numérique grâce à la fonction *optim* du package *stats*.

Afin de mettre correctement en place cette démarche, déterminons par cette fonction une estimation du maximum de vraisemblance lorsque S est observable.

La log-vraisemblance avec S observé s'écrit comme ceci :

$$\Rightarrow \ell(\theta) = \sum_{S_1 \dots S_n} \log(\mathbb{P}(X, S_i|\theta))$$

$$= \sum_{S_1 \dots S_n} \log\left(\mathbb{P}(S_1)\prod_{i=2}^{n}\mathbb{P}(S_i|S_{i-1})\prod_{i=1}^{n}\mathbb{P}(X_i|S_i)\right)$$

$$= \sum_{S_1 \dots S_n} \left(\log(\mathbb{P}(S_1)) + \log\left(\prod_{i=2}^{n}\mathbb{P}(S_i|S_{i-1})\right) + \log\left(\prod_{i=1}^{n}\mathbb{P}(X_i|S_i)\right)\right)$$

$$= \sum_{S_1 \dots S_n} \left(\log(\mathbb{P}(S_1)) + \sum_{i=2}^{n}\log(\mathbb{P}(S_i|S_{i-1})) + \sum_{i=1}^{n}\log(\mathbb{P}(X_i|S_i))\right)$$

$$= \sum_{S_1 \dots S_n} \log(\mathbb{P}(S_1)) + \sum_{S_1 \dots S_n}\sum_{i=2}^{n}\log(\mathbb{P}(S_i|S_{i-1})) + \sum_{S_1 \dots S_n}\sum_{\mathrm{i}=1}^{n}\log(\mathbb{P}(X_i|S_i))$$

$$= \text{cst.} + \sum_{\mathrm{j,k}}\sum_{i=2}^{n}\mathbf{1}_{S_{\mathrm{i\text{-}1}}=j, S_{\mathrm{i}}=k}\log(\mathbb{P}(S_i = k|S_{\mathrm{i\text{-}1}} = j))$$

$$= \text{cst.} + \sum_{\mathrm{j,k}} N_{\mathrm{jk}}\log(\mathbb{P}(S_i = k|S_{i-1} = j))$$

$$= \text{cst.} + N_{00} \times \log((1-e^{-at})(1-f) + e^{-at}) + N_{01} \times \log((1-e^{-at})f)$$

$$+N_{10} \times \log((1-e^{-at})(1-f)) + N_{11} \times \log((1-e^{-at})f + e^{-at})$$

$$= \text{cst.} + (N_{00} + N_{01} + N_{10} + N_{11})\log(1-e^{-at}) + N_{01}\log f + N_{10}\log(1-f) +$$

$$N_{00}\log\left((1-f) + \frac{e^{-at}}{1-e^{-at}}\right) + N_{11}\log\left(f + \frac{e^{-at}}{1-e^{-at}}\right)$$

La matrice N représente donc un comptage de 4 situations différentes représentants l'état IBD d'un locus et du locus qui le suit. C'est à dire que nous avons les 4 situations suivantes :

- le locus est IBD (ie $S_{\mathrm{i\text{-}1}} = 0$) et le locus qui le suit est aussi IBD (ie $S_{\mathrm{i}} = 0$).
- le locus est IBD (ie $S_{\mathrm{i\text{-}1}} = 0$) mais le locus qui le suit n'est pas IBD (ie $S_{\mathrm{i}} = 1$).
- le locus n'est pas IBD (ie $S_{\mathrm{i\text{-}1}} = 1$) mais le locus qui le suit est IBD (ie $S_{\mathrm{i}} = 0$).
- le locus n'est pas IBD (ie $S_{\mathrm{i\text{-}1}} = 1$) et le locus qui le suit n'est pas IBD lui aussi(ie $S_{\mathrm{i}} = 1$).

Nous allons donc dans un premier temps implémenter cette matrice, puis nous implémenterons une fonction *loglik* qui calcule la log-vraisemblance en fonction de la matrice N précédemment construite.

```
# calcul des stat. exhaustive
#ie comptage des situations repésentant l'état IBD d'un locus et du locus qui le suit

N=matrix(NA,2,2)
rownames(N)=colnames(N)=0:1
```

```r
for (j in 0:1) for (k in 0:1) {
  N[j+1,k+1]=sum((s[-n]==j)&(s[-1]==k))
}

loglik=function(par,N){
  a=par[1]; f=par[2];
  #cat("a=",a,"f=",f,"\n")
  aux=exp(-a*t)/(1-exp(-a*t))
  sum(N)*log(1-exp(-a*t))+N[1,2]*log(f)+N[2,1]*log(1-f)+N[1,1]*log(1-f+aux)+N[2,2]*log(f+aux)
}
loglik(c(a,f),N)
```

```
## [1] -3464.939
```

```r
par2theta=function(par){
  res=c(exp(par[1]),exp(par[2])/(1+exp(par[2])))
  names(res)=c("a","f")
  return(res)
}

opt=optim(par=c(log(a),log(f/(1-f))),fn=function(par){-loglik(par2theta(par),N)},method="BFGS")
rbind(par2theta(opt$par),c(a,f))
```

```
##               a          f
## [1,] 0.08702967 0.06228126
## [2,] 0.09000000 0.06250000
```

Nous constatons, qu'avec la fonction *optim*, nous obtenons des estimations statisfaisantes des paramètres a et f avec nos données précédement simulées. Cependant, nous supposons ici que S_i est observable car nous considérons l'égalité suivante : $\sum_{i=2}^{n} \mathbb{P}(S_{\text{i-1}} = j, S_{\text{i}} = k) = \sum_{i=2}^{n} \mathbf{1}_{S_{\text{i-1}}=j, S_{\text{i}}=k}$. Dans le cadre de notre étude S_i est latente, nous allons donc voir la loi a posteriori qui nous sera très utile pour pour utiliser la fonction *optim* sans que S_i ne soit observable.

6 Loi a posteriori

La loi a posteriori est en effet utile pour deviner le statut IBD à un locus donné à partir d'une observation.

Dans notre modèle actuel, $\mathbb{P}(S|X,\theta) = \sum_{S_1} ... \sum_{S_n} \mathbb{P}(X,S|\theta)$ serait trop long à calculer pour un $n > 10$.

Pour remédier à ce problème, nous allons introduire les quantités Forward et Backward tel que :

$F_i(k) = \mathbb{P}(X_1 = x_1, ..., X_i = x_i, S_i = k)$

$B_i(k) = \mathbb{P}(X_{\text{i+1}} = x_{\text{i+1}}, ..., X_n = x_n | S_i = k)$

$\mathbb{P}(S_i = k, X_i = x_i) = F_i(k)B_i(k)$

Ainsi, nous obtenons la loi a posteriori suivante :

$$\begin{aligned}\eta_{\mathrm{i}}(j) = \mathbb{P}(S_{\mathrm{i}}|X_{\mathrm{i}},\theta) &= \frac{\mathbb{P}(S_{\mathrm{i}},\theta)\mathbb{P}(X_{\mathrm{i}}|S_{\mathrm{i}},\theta)}{\mathbb{P}(X_{\mathrm{i}},\theta)} \quad \text{, d'après la formule de Bayes} \\ &= \frac{\mathbb{P}(X_{\mathrm{i}},S_{\mathrm{i}})}{\mathbb{P}(X_{\mathrm{i}})} \\ &= \frac{\mathbb{P}(X_{\mathrm{i}},S_{\mathrm{i}})}{\sum_{\mathrm{S}}\mathbb{P}(X_{\mathrm{i}},S_{\mathrm{i}})} \\ &= \frac{F_{\mathrm{i}}(k)B_{\mathrm{i}}(k)}{\sum_{\mathrm{k}=1}^{\mathrm{n}}F_{\mathrm{i}}(k)B_{\mathrm{i}}(k)}\end{aligned}$$

Nous allons donc implémenter, sous la forme de matrice, les deux quantités Forward et Backward pour chaque locus parallèlement à un statut IBD. Puis nous implémenterons la loi a posteriori $\eta_i(j)$.

```
# Forward/Backward efficient and stable

# Forward
Fw=matrix(NA,n,2)
L=rep(NA,n)
Fw[1,]=mu1*e[1,x[1],]
aux=sum(Fw[1,]); L[1]=log(aux)
Fw[1,]=Fw[1,]/aux
for (i in 2:n) {
  for (k in 1:2)
    Fw[i,k]=sum(Fw[i-1,]*pi[,k]*e[i,x[i],k])
  aux=sum(Fw[i,]); L[i]=L[i-1]+log(aux)
  Fw[i,]=Fw[i,]/aux
}

# backward
Bk=matrix(NA,n,2)
M=rep(NA,n)
Bk[n,]=1
aux=sum(Bk[n,]); M[n]=log(aux)
Bk[n,]=Bk[n,]/aux
for (i in n:2) {
  for (j in 1:2)
    Bk[i-1,j]=sum(pi[j,]*e[i,x[i],]*Bk[i,])
  aux=sum(Bk[i-1,]); M[i-1]=log(aux)+M[i]
  Bk[i-1,]=Bk[i-1,]/aux
}
# verification
head(log(apply(Fw*Bk,1,sum))+L+M)
```

```
## [1] -42074.68 -42074.68 -42074.68 -42074.68 -42074.68 -42074.68
```

```
# loi a posteriori
eta=Fw*Bk
eta=eta/apply(eta,1,sum)

plot(s,t="s",xlim=c(0,250),xlab="Locus/marqueur i",ylab="Statut IBD")
lines(eta[,2],col="red")
lines(s,t="s")
```

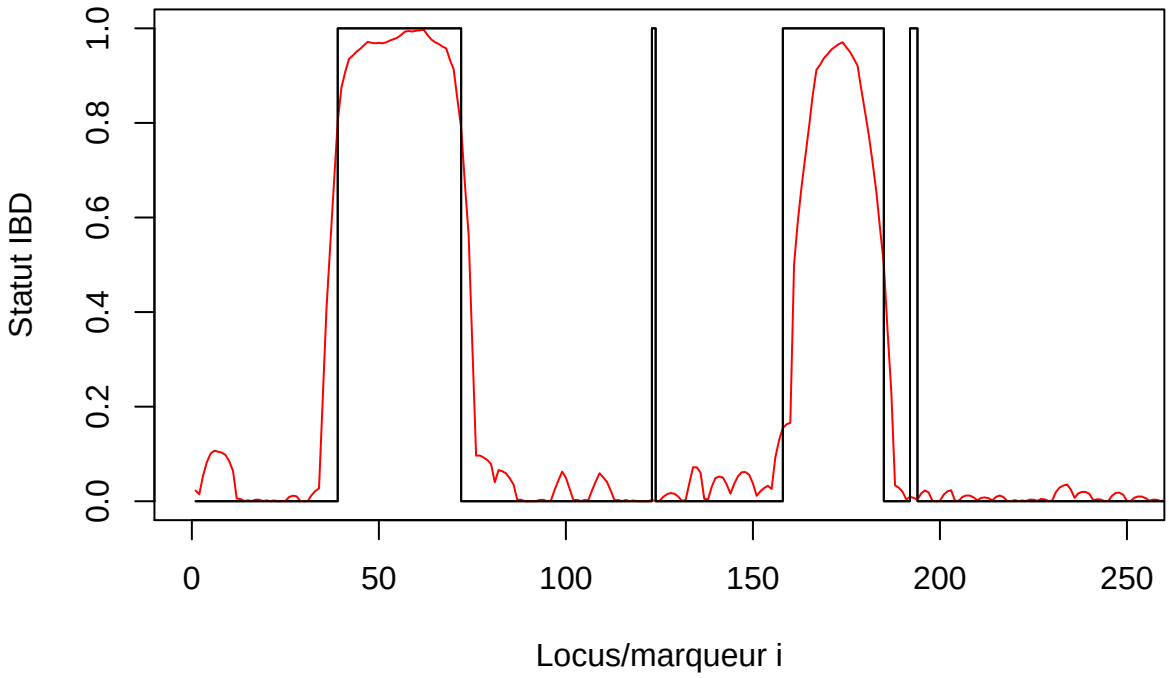

Figure 2: Graphique en marche d'escalier représentant les transitions du statut IBD pour nos données de marqueurs et son estimation par la loi a posteriori

```
tab=table(s,apply(eta,1,which.max))
colnames(tab)=c("IBDpred=0","IBDpred=1")
rownames(tab)=c("IBD*=0","IBD*=1")
knitr::kable(tab,format="markdown")
```

Table 1: IBD prédit vs le vrai.

	IBDpred=0	IBDpred=1
IBD*=0	46457	429
IBD*=1	1553	1561

Dans la Figure 2, nous constatons que la courbe du statut IBD, estimée par la loi a posteriori, se superposent assez bien avec la vrai représentation des statuts IBD de nos données simulées. Ceci confirme donc bien l'utilité de la loi a posteriori afin de devinier le statut IBD lorsque celui ci est latent.

Par ailleurs, la Table 1 nous confirme une nouvelle fois ce fait car nous constatons qu'il y a davantage de statut IBD correctement prédit que de statut IBD mal prédit. Au total, 48018 loci ont fait l'objet d'une prédiction juste sur 50000 loci.

##

```
## Call:
## roc.default(response = s, predictor = eta[, 2], ci = TRUE)
##
## Data: eta[, 2] in 46886 controls (s 0) < 3114 cases (s 1).
## Area under the curve: 0.9462
## 95% CI: 0.9422-0.9502 (DeLong)
```

Nous remarquons que l'aire sous la courbe ROC vaut 0.9462 ce qui est un très bon résultat car plus l'aire sous la courbe se rapproche de 1 et plus la prediction est performante. L'intervalle de confiance nous confirme la bonne qualité de la prédiction par la loi a posteriori.

7 Algorithme EM

Nous allons maintenant effectuer l'implémentation de l'algorithme EM à partir des données précédemment simulées avec le paramètre $\theta = (f, a) = (1/16, 0.09)$ et à l'aide de la fonction *optim* précédement utilisée.

Rappelons les étapes de l'algorithme EM :

$\Rightarrow$ Initialisation du paramètre θ

$\Rightarrow$ Répéter pour i $= \{1, 2, \dots\}$ jusqu'à convergence de θ les phases E et M suivante :

- Phase E : calculer l'espérance $Q(\theta|\theta_{\text{i-1}})$
- Phase M : maximiser $Q(\theta|\theta_{\text{i-1}})$ pour obtenir $\theta_{\text{i}} = M(\theta_{\text{i-1}})$

Nous considérons ici seulement la log-vraisemblance et non l'espérance de la log-vraisemblance car le paramètre ε est fixé.

Précédemment, nous avons vu comment maximiser la log-vraisemblance à partir de la fonction *optim* en supposant que S_i est observée. Le fait que S_i soit observée nous avait permis de construire une matrice de comptage des situations de transition potentielle du statut IBD.
Dans notre contexte d'étude, la variable S_i est latente. En revanche, par la loi a posteriori, nous avons la possibilité de définir $\sum_{\text{i}} \mathbb{P}(S_{\text{i-1}} = j, S_{\text{i}} = k|\theta)$ qui se substitue à cette matrice dans le cadre où S_i est latente.

Ainsi, à titre d'exemple, nous allons implémenter notre algorithme pour une initialisation $\theta = (f, a) = (0.10, 0.15)$ et 20 itérations de la maximisation de la log-vraisemblance grâce à la fonction *optim* qui prend en compte NN, la matrice représentant $\sum_{\text{i}} \mathbb{P}(S_{\text{i-1}} = j, S_{\text{i}} = k|\theta)$ et qui a été déterminé par l'intermédiaire des quantités Forward et Backward présentent dans la loi a posteriori.

À chaque itération, nous affectons à θ les nouveaux paramètres trouvés grâce à cette maximisation et recommençons le processus de la maximisation jusqu'à échéance des 20 itérations.

```
astar=a
fstar=f

# initialization
a=0.15; f=0.10

for (iter in 1:20) {

  # loi initiale
  mu1=c(1-f,f)
```

```r
# transition (la même pour tous les loci)
pi=matrix(NA,2,2)
rownames(pi)=colnames(pi)=c("IBD=0","IBD=1")
pi[1,1]=(1-exp(-a*t))*(1-f)+exp(-a*t) # IBD=0 -> IBD=0
pi[1,2]=(1-exp(-a*t))*f # IBD=0 -> IBD=1
pi[2,1]=(1-exp(-a*t))*(1-f) # IBD=1 -> IBD=0
pi[2,2]=(1-exp(-a*t))*f+exp(-a*t) # IBD=1 -> IBD=1


# emission (differente pour chaque locus)
e=array(NA,dim=c(n,3,2),
        dimnames=list(paste0("i=",1:n),
                      c("RR","RV","VV"),
                      c("IBD=0","IBD=1")))

e[,"RR","IBD=0"]=(1-p)^2
e[,"RV","IBD=0"]=2*p*(1-p)
e[,"VV","IBD=0"]=p^2
e[,"RR","IBD=1"]=(1-eps)*(1-p)+eps*(1-p)^2
e[,"RV","IBD=1"]=eps*2*p*(1-p)
e[,"VV","IBD=1"]=(1-eps)*p+eps*p^2
# Forward
Fw=matrix(NA,n,2)
L=rep(NA,n)
Fw[1,]=mu1*e[1,x[1],]
aux=sum(Fw[1,]); L[1]=log(aux)
Fw[1,]=Fw[1,]/aux
for (i in 2:n) {
  for (k in 1:2)
    Fw[i,k]=sum(Fw[i-1,]*pi[,k]*e[i,x[i],k])
  aux=sum(Fw[i,]); L[i]=L[i-1]+log(aux)
  Fw[i,]=Fw[i,]/aux
}

# backward
Bk=matrix(NA,n,2)
M=rep(NA,n)
Bk[n,]=1
aux=sum(Bk[n,]); M[n]=log(aux)
Bk[n,]=Bk[n,]/aux
for (i in n:2) {
  for (j in 1:2)
    Bk[i-1,j]=sum(pi[j,]*e[i,x[i],]*Bk[i,])
  aux=sum(Bk[i-1,]); M[i-1]=log(aux)+M[i]
  Bk[i-1,]=Bk[i-1,]/aux
}

# statistique exhaustive
ee=matrix(NA,n,2)
for (i in 1:n) ee[i,]=e[i,x[i],]
NN=matrix(NA,2,2)
for (j in 1:2) for (k in 1:2)
  NN[j,k]=sum(Fw[-n,j]*pi[j,k]*ee[-1,k]*Bk[-1,k]*exp(L[-n]+M[-1]-L[n]))
```

```
  # update parameters
  opt=optim(par=c(log(a),log(f/(1-f))),fn=function(par){-loglik(par2theta(par),NN)},method="BFGS")
  aux=par2theta(opt$par)
  a=aux[1]; f=aux[2]

  cat("iter=",iter,"a=",a,"f=",f,"\n")
}
```

```
## iter= 1 a= 0.1544691 f= 0.086648
## iter= 2 a= 0.1523962 f= 0.08105977
## iter= 3 a= 0.1479667 f= 0.07814496
## iter= 4 a= 0.1428864 f= 0.07628416
## iter= 5 a= 0.137849 f= 0.07486662
## iter= 6 a= 0.1331196 f= 0.07365254
## iter= 7 a= 0.1287826 f= 0.07255049
## iter= 8 a= 0.1248496 f= 0.0715281
## iter= 9 a= 0.121304 f= 0.0705753
## iter= 10 a= 0.1181195 f= 0.06968946
## iter= 11 a= 0.1152674 f= 0.06886975
## iter= 12 a= 0.1127186 f= 0.06811524
## iter= 13 a= 0.1104454 f= 0.06742417
## iter= 14 a= 0.1084217 f= 0.06679415
## iter= 15 a= 0.1066231 f= 0.06622244
## iter= 16 a= 0.1050268 f= 0.06570546
## iter= 17 a= 0.1036122 f= 0.06523966
## iter= 18 a= 0.1023879 f= 0.06478658
## iter= 19 a= 0.1012575 f= 0.06443635
## iter= 20 a= 0.1002735 f= 0.06410818
```

```
#Tableau de comparaison des vrais paramètres vs les paramètres estimés
rbind(c(a,f),c(astar,fstar))
```

```
##              a          f
## [1,] 0.1002735 0.06410818
## [2,] 0.0900000 0.06250000
```

Nous constatons qu'avec 20 itérations, nous nous approchons déjà des vrais valeurs de f et a. Nous pouvons supposer qu'avec plus d'iterations nous arrions eu de meilleurs estimations des paramètres f et a. Ceci, nous permet de penser que l'algorithme EM peut être efficace dans le cadre d'un modèle de chaînes de Markov cachées.
Pour essayer de confirmer cela, nous allons effectuer quelques expérimentations sur un grand nombre de simulations de données.

8 Expérimentations

Précédemment, nous nous étions chargé de simuler les données et implémenter l'algorithme EM. Dans cette partie, nous allons effectuer des expérimentations en faisant varier nos différents paramètres afin de tirer des conclusions intéressante en ce qui concerne l'efficacité de cet algorithme pour l'estimation du paramètre θ selon un modèle de chaînes de Markov cachées.

Pour effectuer efficacement nos expérimentations, nous avons créer deux fonctions, *simul* et *estim*. Ces deux fonctions reprennent tout notre cheminement de la simulation des données à l'implémentation de l'algorithme EM.

La fonction *simul* simule nos données selon un vrai coefficient de consanguinité f, un taux d'erreur de génotypage ε, un taux de recombinaison a et les proportions des allèles R et V, p_R et p_V.

La fonction *estim* effectue l'algorithme EM selon un modèle de chaînes de Markov cachées afin d'estimer le paramètre θ et s'arrête lorsque le paramètre θ aura convergé et nous retournera le paramètre θ final estimé et le nombre d'itération qu'il a nécessité.

Le code R de l'implémentation des fonctions simul et estim sont en annexe (Annexe 1)

8.1 Expérimentation selon différentes valeurs du nombre de loci n

Dans un premier temps, nous effectuons des expérimentations en faisant varier le nombre de loci lors de la simulation des données.

Nous allons pour cela effectuer 3 séries de 50 simulations et estimations. La première série consituera 50 simulations avec n=50000 suivis des estimations pour chacune d'elle, la seconde constituera 50 simulations avec n=5000 et ses estimations et la troisième série constituera 50 simulations avec n=500 et ses estimations.

Pour toutes nos simulations, nous avons choisi de fixer $a = 0.09$ et $f = 1/16$. Le but est donc d'estimer ces dernières.

Le code R pour effectuer les 50 simulations et estimations pour chacun des cas est en annexe (Annexe 2, code 1, 2 et 3)

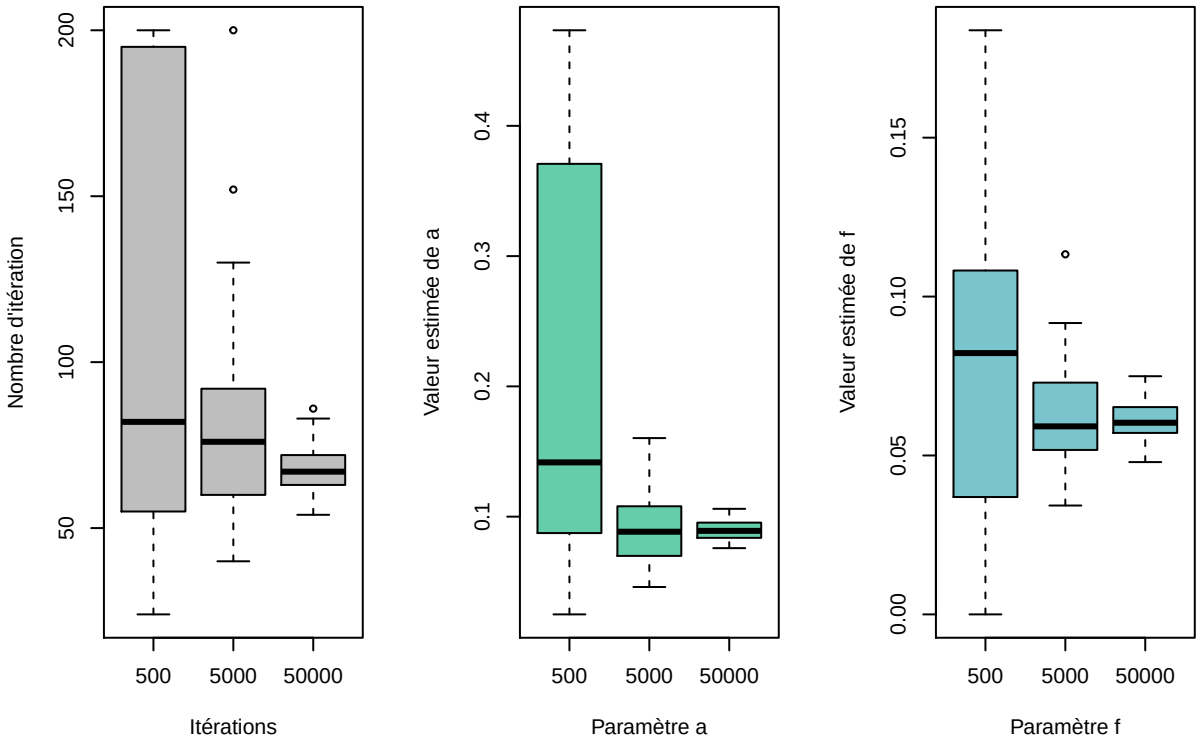

Figure 3: Boxplots des itérations, des estimations de a et des estimations de f selon 50 simulations avec n=500, 50 simulations avec n=5000 et 50 simulations avec n=50000

Dans la Figure 3, nous avons représenté à l'aide de boxplot nos estimations et le nombre d'itérations qui ont été nécessaire. Nous remarquons dans le boxplot des itérations que plus le nombre de loci est important et plus nous avons un nombre d'itérations médiant faible et peu variable entre les estimations. Tandis que pour un faible nombre de loci, le nombre d'itérations semble très variable (de 50 à 200 pour n=500). Le constat semble être le même pour les valeurs estimées de a et f en ce qui concerne la dispersion des valeurs. Plus nous avons un faible nombre de loci est plus les valeurs estimé de a et f sont variables.
Pour tirer de meilleurs conclusions des estimations de a et f, nous allons effectuer des histogrammes représentant la répartition des valeurs estimées.

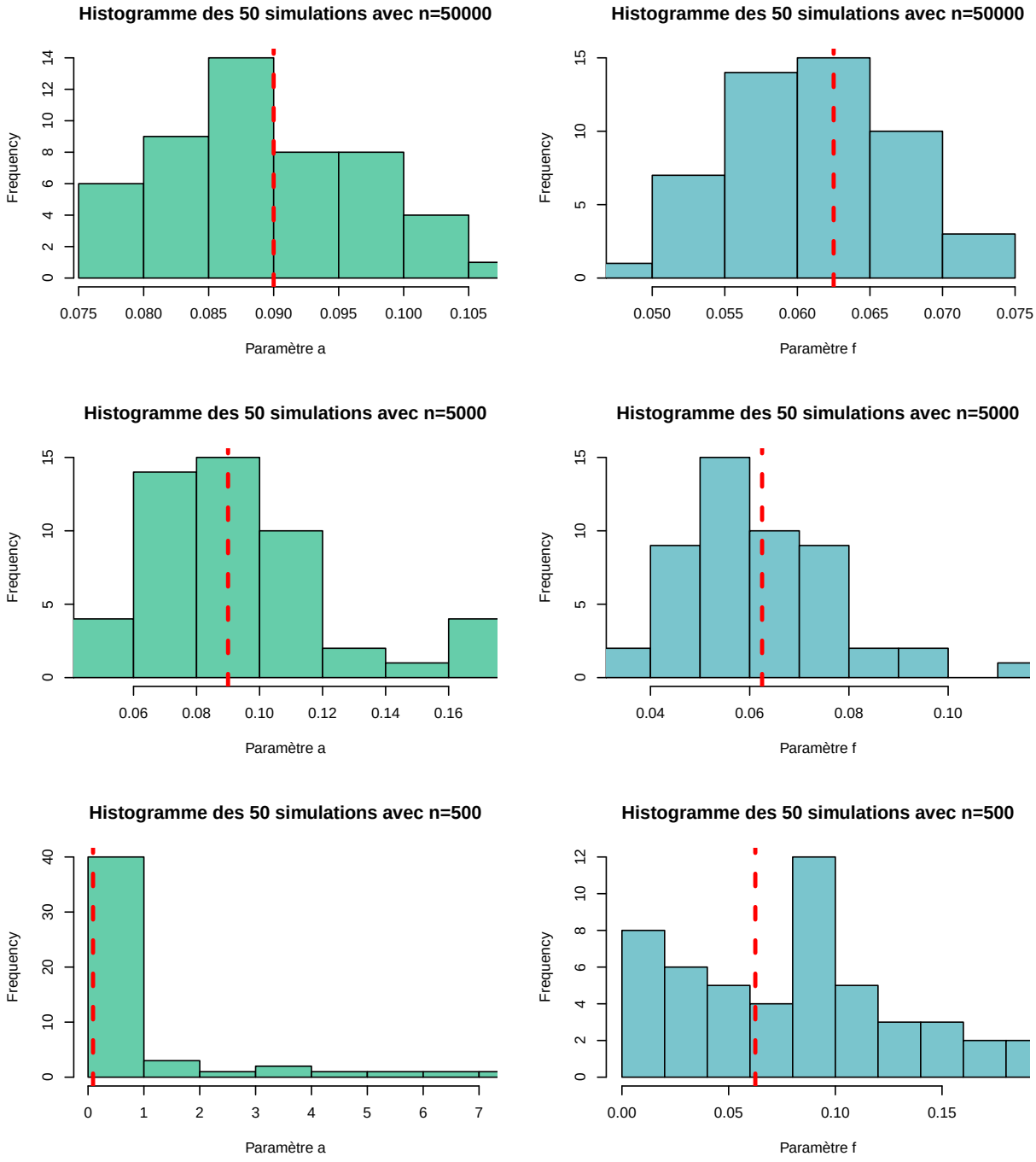

Figure 4: Histogrammes des estimations pour 50 simulations avec n=500, puis 50 simulations avec n=5000 et 50 simulations avec n=50000

Dans la Figure 4, ci-dessus, nous constatons très facilement que les simulations avec un très grand nombre de loci donnent des estimations de a et f meilleurs car nous observons des pics au niveau de la ligne vertical rouge qui représente la vrai valeur à estimer. Cela est un signe de bonne qualité des estimations. Tandis que pour les simulations avec un faible nombre de loci, les estimations pour a contiennent beaucoup d'outliers et les estimations f ne présentent pas de pic au niveau de la vrai valeur de f.

Table 2: Intervalle de confiance et médiane pour le paramètre theta quand n=50000

	a	f
50%	0.0890913	0.0602977
5%	0.0784751	0.0524037
95%	0.1010632	0.0706717

Table 3: Intervalle de confiance et médiane pour le paramètre theta quand n=5000

	a	f
50%	0.0884403	0.0591640
5%	0.0567484	0.0408161
95%	0.1654800	0.0882834

Table 4: Intervalle de confiance et médiane pour le paramètre theta quand n=500

	a	f
50%	0.1416702	0.0822107
5%	0.0619585	0.0002659
95%	5.0691727	0.1638491

Pour conclure ces premières expérimentations sur le nombre de loci, nous avons effectué les Tables 2, 3 et 4 ci-dessus, qui contiennent les intervalles de confiances et médianes des paramètres à estimer.
Nous remarquons dans la Table 2 qui constitue les C.I et médiane pour n=50000, que les estimations semblent de meilleurs qualités que dans la Table 4 qui constitue les C.I et médiane pour n=500. La Table 3, nous montre que les estimations pour n=5000 sont de moins bonnes qualités que pour n=50000 mais de bien meilleur qualité que n=500.

Nous pouvons conclure que l'algorithme EM semble être plus efficace pour l'estimation de a et f avec un très grande nombre de loci à disposition.

8.2 Expérimentation selon différentes valeurs de la distance entre chaque marqueur t

Pour cette deuxième phase d'expérimentation, nous effectuons des expérimentations en faisant varier la distance entre les marqueurs lors de la simulation des données.

Nous allons pour cela effectuer 3 séries de 50 simulations et estimations. La première série consituera 50 simulations avec $t = 0.5$ suivis des estimations pour chacune d'elle, la seconde constituera 50 simulations avec $t = 1.67$ et ses estimations et la troisième série constituera 50 simulations avec $t = 5$ et ses estimations.

Pour toutes nos simulations, nous avons choisi de fixer $a = 0.09$ et $f = 1/16$. Le but est donc d'estimer ces dernières.

Nous fixons également n=5000, qui nous a précédement donné des estimations raisonnables, n=50000 étant trop couteux en temps sur R.

Le code R pour effectuer les 50 simulations et estimations pour chacun des cas est en annexe (Annexe 2, code 4, 2 et 5)

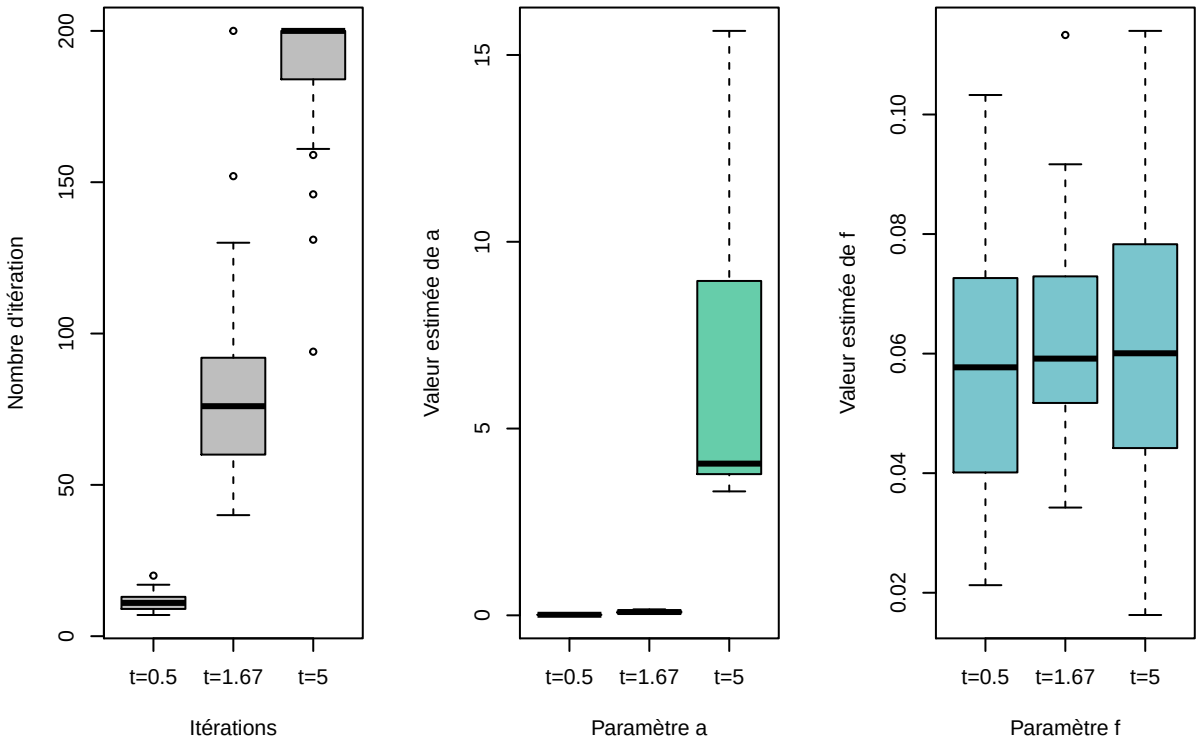

Figure 5: Boxplots des itérations, des estimations de a et des estimations de f selon 50 simulations avec t=0.5, 50 simulations avec t=1.67 et 50 simulations avec t=5

Dans la figure 5, nous avons représenté à l'aide de boxplot nos estimations et le nombre d'itérations qui ont été nécessaire. Nous remarquons dans le boxplot des itérations que plus la distance entre les marqueurs est faible et plus nous avons un nombre d'itérations médiant faible et peu variable entre les différentes estimations. Tandis que pour une grande distance entre les loci, le nombre d'itérations semble très important et arrive le plus souvent au maximum d'itérations proposé par l'algorithme (200, cf Annexe). En ce qui concerne le paramètre a, les estimations pour les simulations avec $t = 5$ semblent complétement erroné mais cela s'explique par le fait que l'algorithme n'a pu converger avant le nombre maximum d'itérations autorisées (nous pouvons supposer que si nous avions autoriser un nombre illimité d'itérations le paramètre a aurait pu être correctement estimé). En ce qui concerne le paramètre f, les estimations pour les différentes distance se ressemblent globalement, elles restent néanmoins plus dispersées pour $t = 0.5$ et $t = 5$.
Pour tirer de meilleurs conclusions des estimations de a et f, nous allons effectuer des histogrammes représentant la répartition des valeurs estimées.

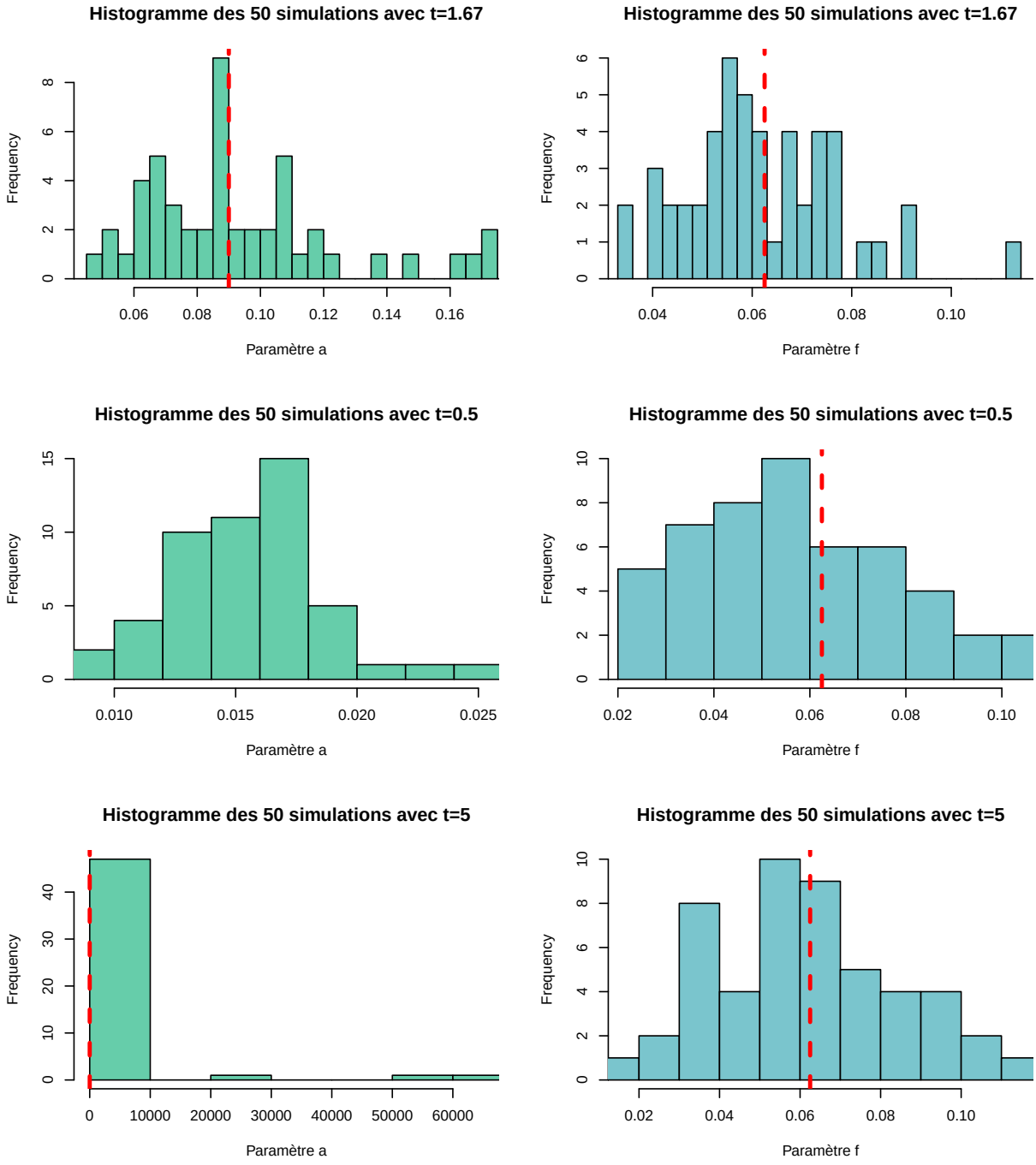

Figure 6: Histogrammes des estimations pour 50 simulations avec t=1.67, puis 50 simulations avec t=0.5 et 50 simulations avec t=5

Dans la Figure 6, ci-dessus, nous constatons très facilement que les simulations avec $t = 1.67$ donnent des estimations de a bien meilleurs que pour les autres distances. En effet, pour $t = 0.5$, nous n'avions pas pu le voir dans le boxplot mais a n'a été pour aucune simulation correctement estimé et pour $t = 5$, il y a beaucoup trop de valeurs estimés aberrantes ce qui confirme ce qui avait été vu dans le boxplot. Tandis que pour $t = 1.67$, un pic est présent au niveau de la vrai valeur de a, ce qui est gage de bonne qualité.

En ce qui concerne le paramètre f, les estimations semblent être plus juste, dans les 3 cas un pic est présent au niveau de la vrai valeur de f.

Table 5: Intervalle de confiance et médiane pour le paramètre theta quand t=0.5

	a	f
50%	0.0156100	0.0577052
5%	0.0105924	0.0259734
95%	0.0205579	0.0962587

Table 6: Intervalle de confiance et médiane pour le paramètre theta quand t=1.67

	a	f
50%	0.0884403	0.0591640
5%	0.0567484	0.0408161
95%	0.1654800	0.0882834

Table 7: Intervalle de confiance et médiane pour le paramètre theta quand t=5

	a	f
50%	4.061426	0.0600558
5%	3.566233	0.0302006
95%	15018.707100	0.0996826

Pour conclure cette seconde phase d'expérimentations sur la distance entre les marqueurs, nous avons effectué les Tables 5, 6 et 7 ci-dessus qui contiennent les intervalles de confiances et médianes des paramètres à estimer. Nous remarquons à nouveau dans la Table 5 qui constitue les C.I et médiane pour $t = 0.5$, que les estimations pour a sont érronées car la vrai valeur de a (0.09) n'est pas compris dans l'intervalle de confiance. Même constat pour la Table 7 qui constitue les C.I et médiance pour t=5.
Néanmoins, dans la Table 6 qui constitue les C.I et médiane pour $t = 1.67$, les estimations pour a sont de bien meilleurs qualités avec la vrai valeur de a compris dans l'intervalle de confiance.
En ce qui concerne les estimations de f, comme nous avons dit précédemment les estimations semblent convenable mais nous constatons ici qu'elles restent meilleurs pour $t = 1.67$, l'intervalle de confiance étant plus restreint que pour les deux autres t.

Nous pouvons conclure que pour que l'algorithme EM soit efficace (avec moins de 200 itérations), il ne faut pas avoir une distance entre les marqueurs ni trop grand ni trop petit. Le choix de $t = 1.67$ dans l'article semble être un très bon choix.

8.3 Expérimentation selon différentes valeurs du taux de recombinaison a

Pour cette troisième phase d'expérimentation, nous effectuons des expérimentations en faisant varier le taux de recombinaison a lors de la simulation des données.

Nous allons pour cela effectuer 3 séries de 50 simulations et estimations. La première série consituera 50 simulations avec $a = 0.02$ suivis des estimations pour chacune d'elle, la seconde constituera 50 simulations avec $a = 0.09$ et ses estimations et la troisième série constituera 50 simulations avec $a = 5$ et ses estimations.

Pour toutes nos simulations, nous avons choisi de fixer $t = 1.67$ et $f = 1/16$. Le but est d'estimer f et a mais nous ferons cette fois attention car a constitue cette fois le coeur de nos expérimentation, il s'agira donc de retrouver $a = 0.02$ pour la 1ère série, $a = 0.09$ pour la seconde série et $a = 5$ pour la dernière série.

Nous fixons également n=5000, qui nous a précédement donné des estimations raisonnables, n=50000 étant trop couteux en temps sur R.

Le code R pour effectuer les 50 simulations et estimations pour chacun des cas est en annexe (Annexe 2, code 6, 2 et 7)

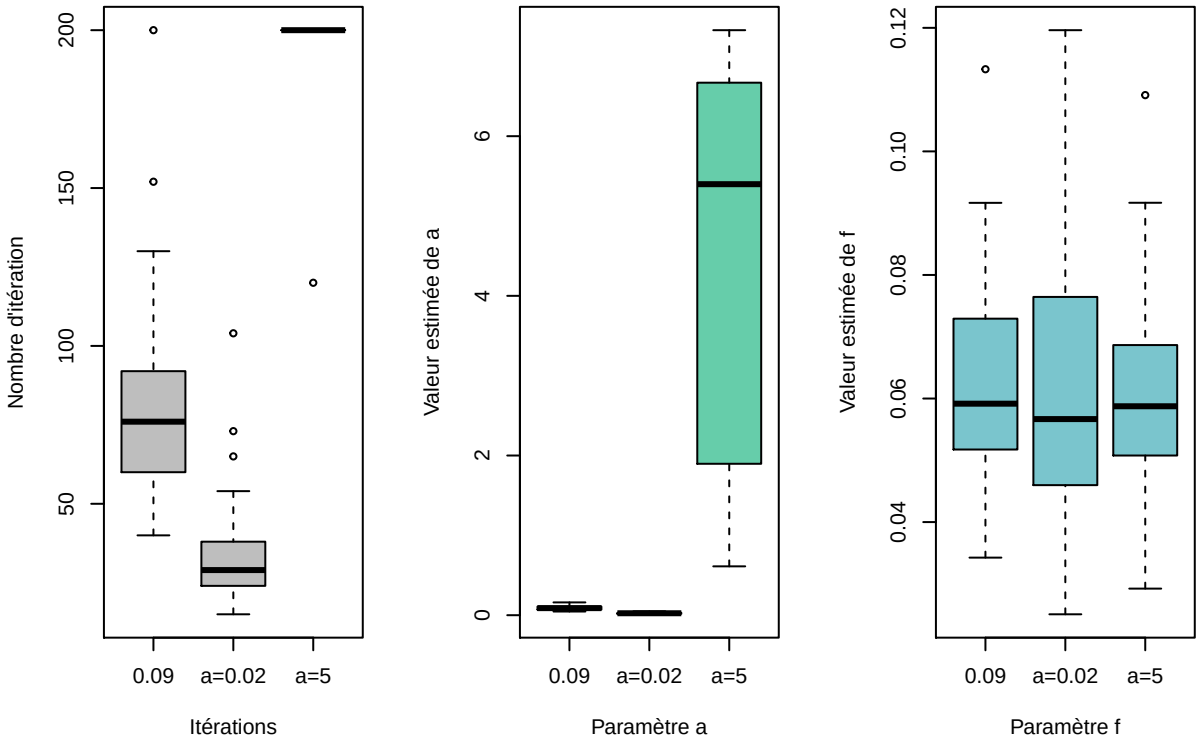

Figure 7: Boxplots des itérations, des estimations de a et des estimations de f selon 50 simulations avec a=0.02, 50 simulations avec a=0.09 et 50 simulations avec a=5

Dans la Figure 7, nous avons représenté à l'aide de boxplot nos estimations et le nombre d'itérations qui ont été nécessaire. Nous remarquons dans le boxplot des itérations que plus le taux de recombinaison a est faible et plus nous avons un nombre d'itérations médiant faible et peu variable entre les différentes estimations. Tandis que pour un grand taux de recombinaison entre les loci, le nombre d'itérations semble très important et arrive presque systématiquement au maximum d'itérations autorisées par l'algorithme (200, cf Annexe).
En ce qui concerne le paramètre a, les estimations pour les simulations avec $a = 5$ semblent beaucoup trop dispersées (a estimé allant de 2 à 7).
En ce qui concerne le paramètre f, les estimations pour les différentes distance se ressemblent globalement, elles restent néanmoins plus dispersées pour $a = 0.02$.
Pour tirer de meilleurs conclusions des estimations de a et f, nous allons effectuer des histogrammes représentant la répartition des valeurs estimées.

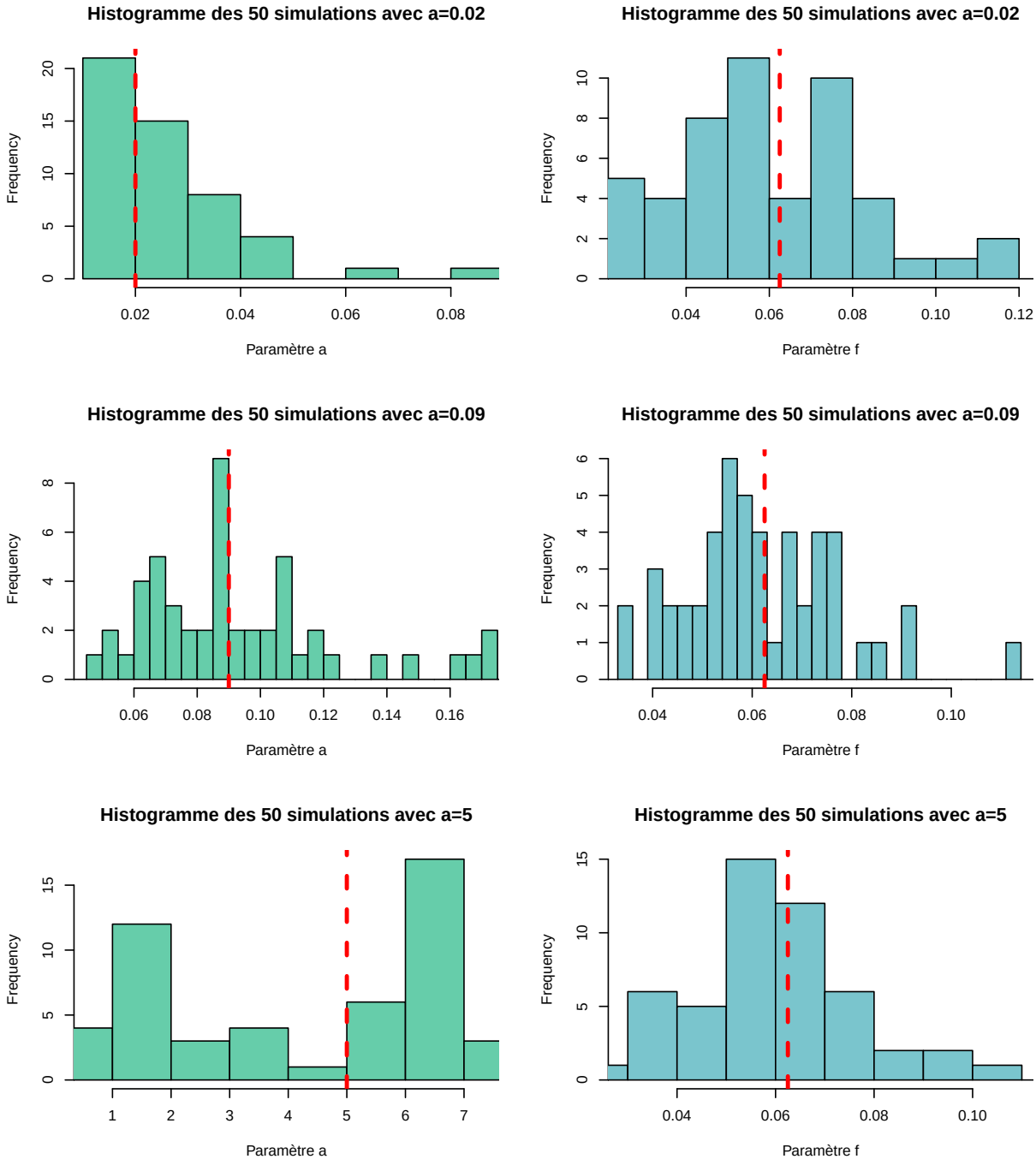

Figure 8: Histogrammes des estimations pour 50 simulations avec a=0.02, puis 50 simulations avec a=0.09 et 50 simulations avec a=5

Dans la Figure 8, ci-dessus, nous constatons très facilement que les simulations avec $a = 0.09$ et $a = 0.02$ donnent des estimations de a bien meilleurs que pour $a = 5$. En effet, pour $a = 5$, les a estimés sont trop dispersés. Tandis que pour les deux autres, un pic est présent au niveau de la vrai valeur de a.
En ce qui concerne le paramètre f, les estimations semblent être plus juste, dans les 3 cas un pic est présent au niveau de la vrai valeur de f.

Table 8: Intervalle de confiance et médiane pour le paramètre theta quand a=0.02

	a	f
50%	0.0243719	0.0566709
5%	0.0134464	0.0259072
95%	0.0484070	0.0992585

Table 9: Intervalle de confiance et médiane pour le paramètre theta quand a=0.09

	a	f
50%	0.0884403	0.0591640
5%	0.0567484	0.0408161
95%	0.1654800	0.0882834

Table 10: Intervalle de confiance et médiane pour le paramètre theta quand a=5

	a	f
50%	5.3986084	0.0587479
5%	0.8939358	0.0361868
95%	7.0922990	0.0889675

Pour conclure cette troisième phase d'expérimentations sur le taux de recombinaison, nous avons effectué les Tables 8, 9 et 10, ci-dessus, qui contiennent les intervalles de confiances et médianes des paramètres à estimer. Nous remarquons dans la Table 10 qui constitue les C.I et médiane pour $a = 5$, que les estimations pour a ne semblent pas de très bonnes qualités car l'intervalle de confiance est très grand. Néanmoins, la Table 8 et 9 qui constitue respectivement les C.I et médiane pour $a = 0.02$ et $a = 0.09$, les estimations pour a sont de bien meilleurs qualités avec un intervalle de confiance plus restreint. En ce qui concerne les estimations de f, comme nous avons dit précédemment les estimations semblent convenable quelque soit le a.

Nous pouvons donc conclure que l'algorithme EM semble moins efficace en ce qui concerne l'estimation de a si celui-ci est grand. Il permet néanmoins d'estimer correctement f. L'idéal serait d'avoir un taux de recombinaison a petit.

8.4 Expérimentation selon différentes valeurs du coefficient de consanguinité f

Pour cette dernière phase d'expérimentation, nous effectuons des expérimentations en faisant varier le coefficient de consanguinité f lors de la simulation des données.

Nous allons pour cela effectuer 3 séries de 50 simulations et estimations. La première série consituera 50 simulations avec $f = 1/4$ (frère/soeur) suivis des estimations pour chacune d'elle, la seconde constituera 50 simulations avec $f = 1/16$ (cousin/cousine) et ses estimations et la troisième série constituera 50 simulations avec $f = 1/8$ (demi-frère/demi-soeur) et ses estimations.

Pour toutes nos simulations, nous avons choisi de fixer $t = 1.67$ et $a = 0.09$. Le but est d'estimer f et a mais nous ferons cette fois attention car f constitue le coeur nos expérimentation, il s'agira donc de retrouver $f = 1/4$ pour la 1ère série, $f = 1/16$ pour la seconde série et $f = 1/8$ pour la dernière série.

Nous fixons également n=5000, qui nous a précédement donné des estimations raisonnables, n=50000 étant trop couteux en temps sur R.

Le code R pour effectuer les 50 simulations et estimations pour chacun des cas est en annexe (Annexe 2, code 8, 2 et 9)

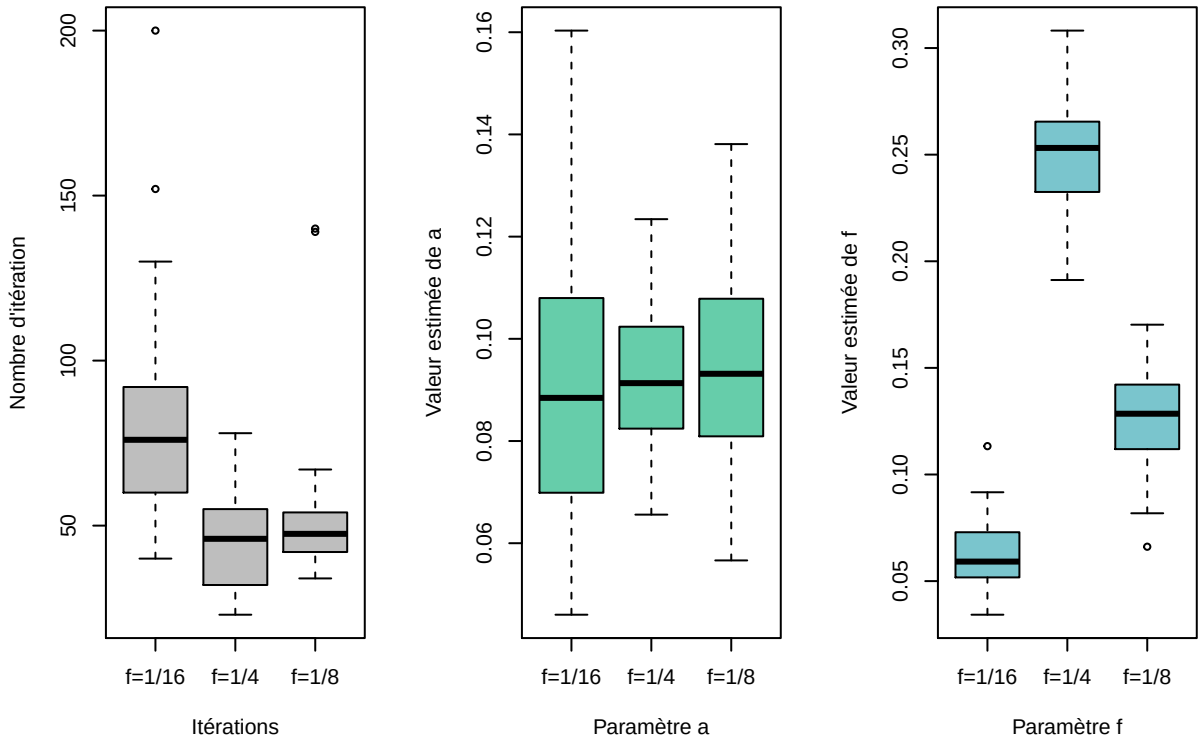

Figure 9: Boxplots des itérations, des estimations de a et des estimations de f selon 50 simulations avec f=1/4, 50 simulations avec f=1/16 et 50 simulations avec f=1/8

Dans la Figure 9, nous avons représenté à l'aide de boxplot nos estimations et le nombre d'itérations qui ont été nécessaire. Nous remarquons dans le boxplot des itérations que $f = 1/16$ nécessite plus d'itérations que pour les autres f mais cela reste convenable. En ce qui concerne le paramètre a, les estimations pour les simulations avec $f = 1/16$ semblent un peu plus dispersées que pour $f = 1/4$ et $f = 1/8$.
En ce qui concerne le paramètre f, les estimations sont peu dispersées et semblent se concentrer vers les vrais valeurs de f (respectivement 1/16, 1/4 et 1/8).
Pour tirer de meilleurs conclusions des estimations de a et f, nous allons effectuer des histogrammes représentant la répartition des valeurs estimées.

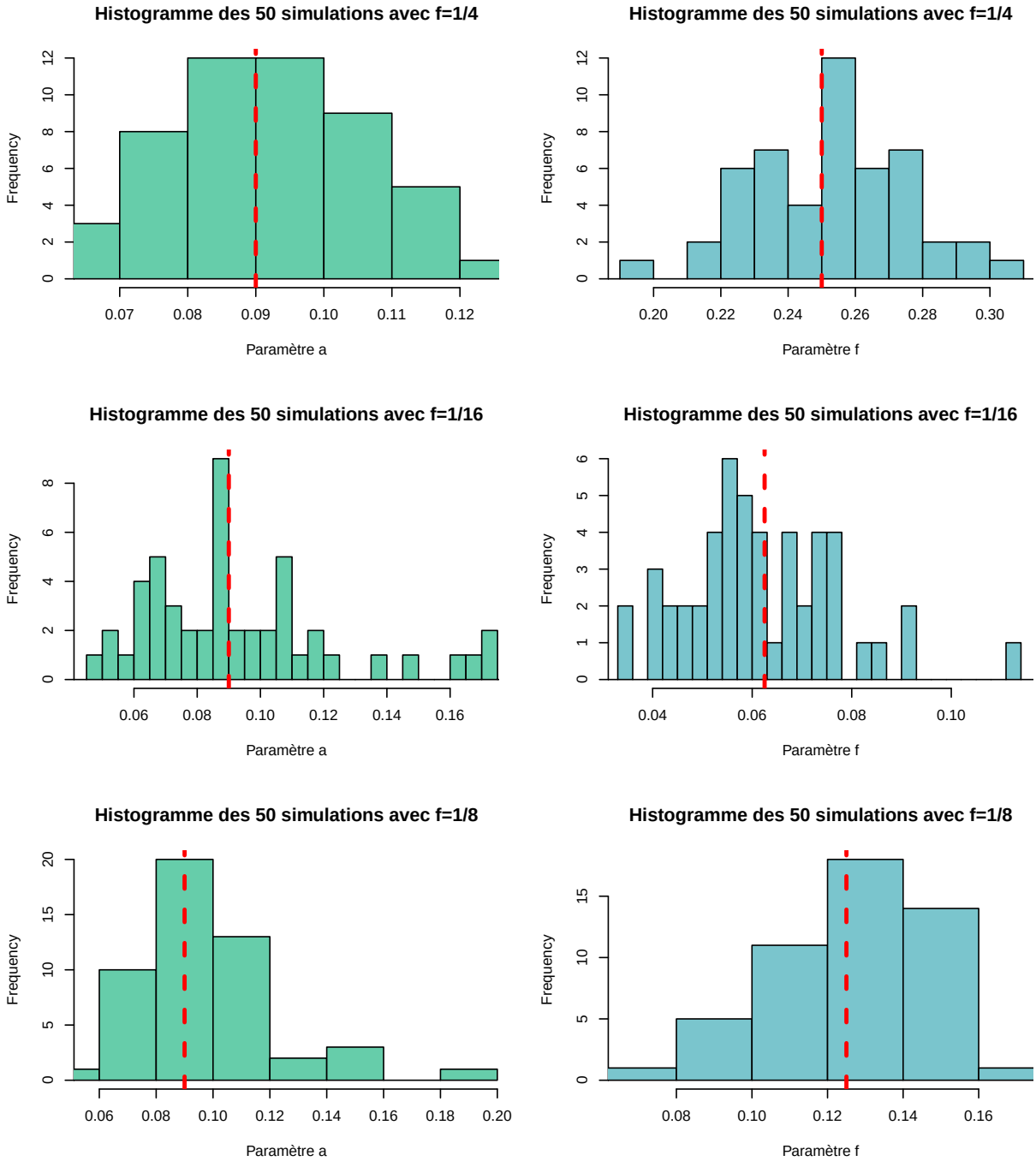

Figure 10: Histogrammes des estimations pour 50 simulations avec f=1/4, puis 50 simulations avec f=1/16 et 50 simulations avec f=1/8

Dans la Figure 8, ci-dessus, nous constatons très facilement que les simulations avec $f = 1/8$, $f = 1/16$ et $f = 1/4$ donnent de plutôt bonnes estimations de f et de a. En effet, dans les 3 cas et pour chaque paramètre, un pic est présent au niveau de la vrai valeur du paramètre.

Table 11: Intervalle de confiance et médiane pour le paramètre theta quand f=1/4

	a	f
50%	0.0913256	0.2531609
5%	0.0698012	0.2203738
95%	0.1163101	0.2876676

Table 12: Intervalle de confiance et médiane pour le paramètre theta quand f=1/16

	a	f
50%	0.0884403	0.0591640
5%	0.0567484	0.0408161
95%	0.1654800	0.0882834

Table 13: Intervalle de confiance et médiane pour le paramètre theta quand f=1/8

	a	f
50%	0.0931582	0.1285009
5%	0.0697643	0.0892806
95%	0.1551380	0.1510144

Pour conclure cette dernière phase d'expérimentations sur le coefficient de consanguinité, nous avons effectué les Tables 11, 12 et 13, ci-dessus, qui contiennent les intervalles de confiances et médianes des paramètres à estimer. Nous remarquons dans les 3 cas et pour chaque paramètre que les estimations sont de bonnes qualités avec des intervalles de confiances correct (ie pas trop grand).

Nous pouvons conclure que le coefficient de consanguinité n'a pas d'influence sur l'efficacité de l'algorithme EM sous un modèle de chaînes de Markov cachées.

9 Conclusions

L'idée de Leutenegger et al. (2003) qui consiste à utiliser les chaînes de Markov cachées pour représenter le statut IBD par les dépendances de marqueurs et d'utiliser le maximum de vraisemblance semblait théoriquement être une très bonne idée qui nous menerait à une estimation du coefficient de consanguinité sans avoir à connaître les généalogies d'un individu. Nous avons donc utilisé l'algorithme EM pour le maximum de vraisemblance afin de pouvoir discuter des différents paramètres qui consistue le modèle. Par nos expérimentations, nous avons pu voir que le nombre de loci n est déterminant car s'il n'est pas assez grand, une estimation du coefficient de consanguinité f ne sera pas envisageable. Par ailleurs, des données avec une distance entre les marqueurs trop petit ou trop grand, nous conduira à une estimation du coefficient de consanguité f moins bonne (et une estimation de taux de recombinaison a complétement éronné). En revanche, le taux de recombinaison a n'a visiblement pas de "mauvaise" influence sur l'estimation du coefficient de consanguinité f. Et finalement, parmi les 3 coefficient de consanguinité (f=1/4 : frère/soeur , f=1/16 : cousin/cousine , f=1/8 : demi-frère/demi-soeur) que nous avons expérimenté, aucun d'eux n'a présenté plus de difficultés à être estimé.

10 Annexes

10.1 Annexe 1 : Implémentation de l'algorithme EM dans le contexte des HMM

```
# simulation
simul=function(n,t,f,a){
  eps=0.05 # taux d'erreur genotypage
  p=runif(n,min=0.05,max=0.49)
  # loi initiale
  mu1=c(1-f,f)

  # transition (la même pour tous les loci)
  pi=matrix(NA,2,2)
  rownames(pi)=colnames(pi)=c("IBD=0","IBD=1")
  pi[1,1]=(1-exp(-a*t))*(1-f)+exp(-a*t) # IBD=0 -> IBD=0
  pi[1,2]=(1-exp(-a*t))*f # IBD=0 -> IBD=1
  pi[2,1]=(1-exp(-a*t))*(1-f) # IBD=1 -> IBD=0
  pi[2,2]=(1-exp(-a*t))*f+exp(-a*t) # IBD=1 -> IBD=1
  apply(pi,1,sum) # verification

  # emission (differente pour chaque locus)
  e=array(NA,dim=c(n,3,2),
          dimnames=list(paste0("i=",1:n),
                        c("RR","RV","VV"),
                        c("IBD=0","IBD=1")))

  e[,"RR","IBD=0"]=(1-p)^2
  e[,"RV","IBD=0"]=2*p*(1-p)
  e[,"VV","IBD=0"]=p^2
  e[,"RR","IBD=1"]=(1-eps)*(1-p)+eps*(1-p)^2
  e[,"RV","IBD=1"]=eps*2*p*(1-p)
  e[,"VV","IBD=1"]=(1-eps)*p+eps*p^2


  s=rep(NA,n)
  s[1]=sample(0:1,prob=mu1,size=1)
  for (i in 2:n)
    s[i]=sample(0:1,prob=pi[s[i-1]+1,],size=1)

  x=rep(NA,n)
  # x[1]=sample(c("RR","RV","VV"),size=1,prob=e[1,,"IBD=1"])
  for (i in 1:n)
    x[i]=sample(c("RR","RV","VV"),size=1,prob=e[i,,s[i]+1])



  return(list(theta=list(a=a,f=f),
              s=s,
              x=x,
              n=n,
              p=p))
}
```

```
estim=function(x,n,p,t,
               a,
               f,
               itermax=500,affiche=FALSE) {

  for(iter in 1:itermax){
    mu1=c(1-f,f)
  # transition (la même pour tous les loci)
  pi=matrix(NA,2,2)
  rownames(pi)=colnames(pi)=c("IBD=0","IBD=1")
  pi[1,1]=(1-exp(-a*t))*(1-f)+exp(-a*t) # IBD=0 -> IBD=0
  pi[1,2]=(1-exp(-a*t))*f # IBD=0 -> IBD=1
  pi[2,1]=(1-exp(-a*t))*(1-f) # IBD=1 -> IBD=0
  pi[2,2]=(1-exp(-a*t))*f+exp(-a*t) # IBD=1 -> IBD=1
  apply(pi,1,sum) # verification
  # emission (differente pour chaque locus)
  e=array(NA,dim=c(n,3,2),
          dimnames=list(paste0("i=",1:n),
                        c("RR","RV","VV"),
                        c("IBD=0","IBD=1")))

  e[,"RR","IBD=0"]=(1-p)^2
  e[,"RV","IBD=0"]=2*p*(1-p)
  e[,"VV","IBD=0"]=p^2
  e[,"RR","IBD=1"]=(1-eps)*(1-p)+eps*(1-p)^2
  e[,"RV","IBD=1"]=eps*2*p*(1-p)
  e[,"VV","IBD=1"]=(1-eps)*p+eps*p^2
  # Forward
  Fw=matrix(NA,n,2)
  L=rep(NA,n)
  #Fw[1,]=mu1*dnorm(x[1],m,sigma) # avant
  Fw[1,]=mu1*e[1,x[1],]
  aux=sum(Fw[1,]); L[1]=log(aux)
  Fw[1,]=Fw[1,]/aux
  for (i in 2:n) {
    for (k in 1:2)
      #Fw[i,k]=sum(Fw[i-1,]*pi[,k]*e[i,k]) # avant
      Fw[i,k]=sum(Fw[i-1,]*pi[,k]*e[i,x[i],k])
    aux=sum(Fw[i,]); L[i]=L[i-1]+log(aux)
    Fw[i,]=Fw[i,]/aux
  }
  # backward
  Bk=matrix(NA,n,2)
  M=rep(NA,n)
  Bk[n,]=1
  aux=sum(Bk[n,]); M[n]=log(aux)
  Bk[n,]=Bk[n,]/aux
  for (i in n:2) {
    for (j in 1:2)
      #Bk[i-1,j]=sum(pi[j,]*e[i,]*Bk[i,]) # avant
      Bk[i-1,j]=sum(pi[j,]*e[i,x[i],]*Bk[i,])
    aux=sum(Bk[i-1,]); M[i-1]=log(aux)+M[i]
    Bk[i-1,]=Bk[i-1,]/aux
```

```
  }
  # statistique exhaustive
  ee=matrix(NA,n,2)
  for (i in 1:n) ee[i,]=e[i,x[i],]
  NN=matrix(NA,2,2)
  for (j in 1:2) for (k in 1:2)
    NN[j,k]=sum(Fw[-n,j]*pi[j,k]*ee[-1,k]*Bk[-1,k]*exp(L[-n]+M[-1]-L[n]))
  # update parameters
  opt=optim(par=c(log(a),log(f/(1-f))),fn=function(par){-loglik(par2theta(par),NN)}
            ,method="BFGS")
  aux=par2theta(opt$par)

    if(round(a,6)==round(aux[1],6) && round(f,6)==round(aux[2],6)){
      if(affiche==TRUE)
        print(cat("iter=",iter,"a=",a,"f=",f,"\n"))
      break
    }else{
      a=aux[1]
      f=aux[2]
      if(affiche==TRUE)print(cat("iter=",iter,"a=",a,"f=",f,"\n"))
    }
  }
  list(
    theta=list(a=a,f=f),
    iter=iter
  )
}
```

10.2 Annexe 2 : Implémentation des expérimentations avec 50 simulations + estimations

10.2.1 Code 1

```
nrep=50
sim1=vector(nrep,mode='list') #List qui comportera tte les simulations
est1=vector(nrep,mode='list') #List qui comportera tte les estimations

for(rep in 1:nrep){
  sim1[[rep]]=simul(n = 50000,t = 1.67,f = 1/16,a = 0.09)
  est1[[rep]]=estim(x = sim1[[rep]]$x,n = 50000,p = sim1[[rep]]$p,t=1.67,a = 0.15,f = 0.10,
                    itermax = 200,affiche = FALSE)
}
```

10.2.2 Code 2

```
nrep=50
sim2=vector(nrep,mode='list') #List qui comportera tte les simulations
est2=vector(nrep,mode='list') #List qui comportera tte les estimations
```

```
for(rep in 1:nrep){
  sim2[[rep]]=simul(n = 5000,t = 1.67,f = 1/16,a = 0.09)
  est2[[rep]]=estim(x = sim2[[rep]]$x,n = 5000,p = sim2[[rep]]$p,t=1.67,a = 0.15,f = 0.10
                    ,itermax = 200,affiche = FALSE)
}
```

10.2.3 Code 3

```
nrep=50
sim3=vector(nrep,mode='list') #List qui comportera tte les simulations
est3=vector(nrep,mode='list') #List qui comportera tte les estimations

for(rep in 1:nrep){
  sim3[[rep]]=simul(n = 500,t = 1.67,f = 1/16,a = 0.09)
  est3[[rep]]=estim(x = sim3[[rep]]$x,n = 500,p = sim3[[rep]]$p,t=1.67,a = 0.15,f = 0.10
                    ,itermax = 200,affiche = FALSE)
}
```

10.2.4 Code 4

```
nrep=50
sim4=vector(nrep,mode='list') #List qui comportera tte les simulations
est4=vector(nrep,mode='list') #List qui comportera tte les estimations

for(rep in 1:nrep){
  sim4[[rep]]=simul(n = 5000,t = 0.5,f = 1/16,a = 0.09)
  est4[[rep]]=estim(x = sim4[[rep]]$x,n = 5000,p = sim4[[rep]]$p,t=0.5,a = 0.15,f = 0.10
                    ,itermax = 200,affiche = FALSE)
}
```

10.2.5 Code 5

```
nrep=50
sim5=vector(nrep,mode='list') #List qui comportera tte les simulations
est5=vector(nrep,mode='list') #List qui comportera tte les estimations

for(rep in 1:nrep){
  sim5[[rep]]=simul(n = 5000,t = 5,f = 1/16,a = 0.09)
  est5[[rep]]=estim(x = sim5[[rep]]$x,n = 5000,p = sim5[[rep]]$p,t=5,a = 0.15,f = 0.10
                    ,itermax = 200,affiche = FALSE)
}
```

10.2.6 Code 6

```
nrep=50
sim6=vector(nrep,mode='list') #List qui comportera tte les simulations
est6=vector(nrep,mode='list') #List qui comportera tte les estimations

for(rep in 1:nrep){
  sim6[[rep]]=simul(n = 5000,t = 1.67,f = 1/16,a = 0.02)
  est6[[rep]]=estim(x = sim6[[rep]]$x,n = 5000,p = sim6[[rep]]$p,t=1.67,a = 0.15,f = 0.10
                    ,itermax = 200,affiche = FALSE)
}
```

10.2.7 Code 7

```
nrep=50
sim7=vector(nrep,mode='list') #List qui comportera tte les simulations
est7=vector(nrep,mode='list') #List qui comportera tte les estimations

for(rep in 1:nrep){
  sim7[[rep]]=simul(n = 5000,t = 1.67,f = 1/16,a = 5)
  est7[[rep]]=estim(x = sim7[[rep]]$x,n = 5000,p = sim7[[rep]]$p,t=1.67,a = 3,f = 0.10
                    ,itermax = 200,affiche = FALSE)
}
```

10.2.8 Code 8

```
nrep=50
sim8=vector(nrep,mode='list') #List qui comportera tte les simulations
est8=vector(nrep,mode='list') #List qui comportera tte les estimations

for(rep in 1:nrep){
  sim8[[rep]]=simul(n = 5000,t = 1.67,f = 1/4,a = 0.09)
  est8[[rep]]=estim(x = sim8[[rep]]$x,n = 5000,p = sim8[[rep]]$p,t=1.67,a = 0.15,f = 0.10
                    ,itermax = 200,affiche = FALSE)
}
```

10.2.9 Code 9

```
nrep=50
sim9=vector(nrep,mode='list') #List qui comportera tte les simulations
est9=vector(nrep,mode='list') #List qui comportera tte les estimations

for(rep in 1:nrep){
  sim9[[rep]]=simul(n = 5000,t = 1.67,f = 1/8,a = 0.09)
  est9[[rep]]=estim(x = sim9[[rep]]$x,n = 5000,p = sim9[[rep]]$p,t=1.67,a = 0.15,f = 0.10
                    ,itermax = 200,affiche = FALSE)
}
```

Références

Dempster, Arthur P, Nan M Laird, and Donald B Rubin. 1977. "Maximum Likelihood from Incomplete Data via the Em Algorithm." *Journal of the Royal Statistical Society: Series B (Methodological)* 39 (1): 1–22.

Leutenegger, Anne-Louise, Bernard Prum, Emmanuelle Génin, Christophe Verny, Arnaud Lemainque, Françoise Clerget-Darpoux, and Elizabeth A Thompson. 2003. "Estimation of the Inbreeding Coefficient Through Use of Genomic Data." *The American Journal of Human Genetics* 73 (3): 516–23.