STA211 - sujet 1
Estimation d'une taille de population à partir de données de capture-marquage-recapture

Anthony Kalaydjian - Mathieu Occhipinti

2023-05-03

Contents

Introduction

Les méthodes de capture-marquage-recapture sont des méthodes astucieuses d'échantillonnage non destructif pour évaluer le nombre (inconnu) d'individus N dans une population. Dans le domaine de la gestion halieutique, par exemple, elles consistent à effectuer un certain nombre de pêches successives, avec remise, à l'aide d'un dispositif de capture (le plus souvent pêche électrique) d'efficacité π. L'efficacité est la probabilité de capture d'un poisson: elle dépend du milieu et de l'effort de pêche, et peut dépendre de la taille du poisson d'intérêt. Les poissons capturés à chaque pêche sont marqués puis remis à l'eau. Utilisées principalement en écologie, les méthodes de capture-marquage-recapture trouvent aussi des applications de portée bien plus large pour des recensements menés dans différents domaines.

Dans cet exercice, nous considérons le cas de 2 expériences successives de pêche (avec marquage et remise) réalisées afin d'estimer le nombre (inconnu) de poissons N dans un lac. On appelle C_1 et C_2 le nombre total de poissons capturés et marqués lors des pêches 1 et 2 respectivement. On appelle C_{20} le nombre de poissons non marqués capturés lors de la deuxième pêche et C_{21} le nombre de poissons marqués capturés lors de la deuxième pêche. On a donc : $C_2 = C_{20} + C_{21}$.

Les données disponibles proviennent d'une expérience réelle "miniature" de capture-marquage-recapture réalisée par des étudiants à l'aide d'un saladier ("le lac") rempli de riz ("l'eau du lac") et de haricots blancs ("les poissons"). Les données observées par les étudiants sont les suivantes : $C_1 = 125$, $C_{20} = 134$ et $C_{21} = 21$.

On considère le modèle probabiliste $\mathcal{M}$ suivant:

$$
\begin{aligned}
C_1 &\sim Binomial(N, \pi) \\
C_{20} &\sim Binomial(N - C_1, \pi) \\
C_{21} &\sim Binomial(C_1, \pi)
\end{aligned}
$$

1 Pour commencer...

1.1 Vraissemblance du modèle

La vraissemblance du modèle $\mathcal{M}$ s'écrit comme suit :

$$\begin{aligned}
[C_1 = c_1, C_{20} = c_{20}, C_{21} = c_{21} | \pi, N] &= [C_1 = c_1 | \pi, N]\,[C_{20} = c_{20}, | \pi, N, C_1 = c_1]\,[C_{21} = c_{21} | \pi, N, C_1 = c_1, C_{20} = c_{20},] \\
&= [C_1 = c_1 | \pi, N]\,[C_{20} = c_{20}, | \pi, N, C_1 = c_1]\,[C_{21} = c_{21} | \pi, N, C_1 = c_1] \\
&= C_{c_1}^{N} \pi^{c_1} (1-\pi)^{N-c_1} C_{c_20}^{N-c_1} \pi^{c_{20}} (1-\pi)^{N-c_1-c_{20}} \\
&= C_{N}^{c_1} \pi^{c_1} (1-\pi)^{N-c_1} C_{N-c_1}^{c_{20}} \pi^{c_{20}} (1-\pi)^{N-c_1-c_{20}} C_{c_1}^{c_{21}} \pi^{c_{21}} (1-\pi)^{C_1-c_{21}}
\end{aligned}$$

On en déduit donc la log-vraissemblance en passant au log :

$$\begin{aligned}
\mathrm{l}(N, \pi) &= \ln\left(C_N^{c_1} C_{N-c_1}^{c_{20}} C_{c_1}^{c_{21}}\right) + (c_1 + c_{20} + c_{21}) \ln(\pi) + (2N - 2c_1 - c_{20} + c_1 - c_{21}) \ln(1-\pi) \\
&= \ln\left(C_N^{c_1} C_{N-c_1}^{c_{20}} C_{c_1}^{c_{21}}\right) + (c_1 + c_2) \ln(\pi) + (2N - c_1 - c_2) \ln(1-\pi)
\end{aligned}$$

car $c_{20} + c_{21} = c_2$

1.2 Simulation du tirage de C_1 par inversion générique

La fonction de répartition de la loi discrète de $C_1 \sim \mathcal{B}(N, \pi)$ est la suivante :

$\forall x \in [0,1], \qquad F(x) = \sum_{k=0}^{N} \mathbb{P}(C_1 = k) 1_{\{k \leq x\}}$

On remarque que $\forall u \in [0,1], \exists p \in [0, N] \quad / \quad \sum_{k=0}^{p-1} \mathbb{P}(C_1 = k) \leq u \leq \sum_{k=0}^{p} \mathbb{P}(C_1 = k)$ [1]

Ainsi, $\forall x \in [k, k+1], \quad F(x) = \sum_{k=0}^{p} \mathbb{P}(C_1 = k) \geq u$

L'inverse généralisée de la loi discrète s'écrit donc : $F^{-1}(u) = p$

Finalement, on a :

$$\boxed{\forall u \in [0,1], \quad F^{-1}(u) = \inf_{p=1,\dots,N} \left\{ p \mid \sum_{k=0}^{p} \mathbb{P}(C_1 = k) \geq u \right\}}$$

```
my.qbinom <- function(u, N, pi){
  p <- sapply(c(0:N), FUN=function(n) choose(N, n)*pi^n*(1-pi)^{N-n})
  cdf <- cumsum(p)
  return(findInterval(u, cdf))
}

my.rbinom <- function(N, pi, n.iter=1){
  U <- runif(n=n.iter, min=0, max=1)
  res <- sapply(U, FUN=function(u) my.qbinom(u,N, pi))
  return(res)
}
```

[1] Avec la convention $\sum_{k=0}^{-1} \mathbb{P}(C_1 = k) = 0$

```r
n.iter <- 10000
N <- 125
pi <- 0.15

generated.C1 <- my.rbinom(N, pi, n.iter)
```

```r
resultats <- data.frame(n=1:n.iter, valeurs=factor(generated.C1, levels = 0:N))
```

```r
#frequence theorique
freq_theo =dbinom(0:N, N, pi)

#calcul de la frequence empirique
freq_emp <- c()
for (k in 0:N){
 freq_emp <- c(freq_emp, mean(generated.C1==k))
}
freq_binom <- tibble( x=0:N, freq_emp=freq_emp, freq_theo=freq_theo)

#Représentation graphique
ggplot(freq_binom) + #Tableau représenter
aes(x = x) + #Abscisse commune
geom_col(mapping = aes(y = freq_emp), #Ordonne des frquences empiriques
width = 0.2, fill = "lightblue") +
geom_point(aes(y = freq_theo), #On ajoute le point des frquences thoriques
shape = 3, col = "red", size = 3) +
xlim(0, 40) +
labs(y = "Frequence", x = "Nombre de succes")
```

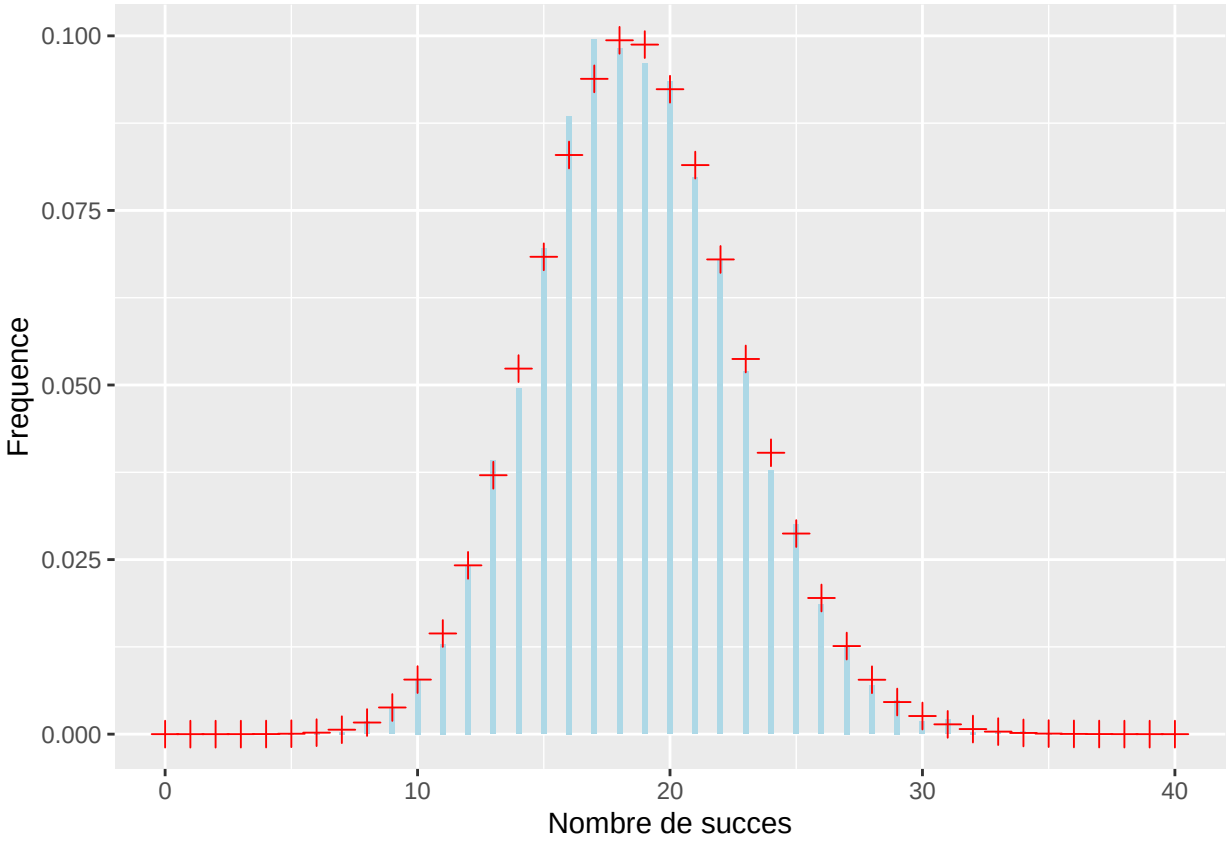

Figure 1: Comparaison des fréquences

Afficher la densité calculée par inversion générique et la densité théorique montre la qualité de l'estimation de notre calcul.

1.3 Simulation d'une réalisation possible de capture-marquage-recapture

```
capture.sim <- function(N, pi){
  C1 <- my.rbinom(N=N, pi=pi)
  C20 <- my.rbinom(N=N-C1, pi=pi)
  C21 <- my.rbinom(N=C1, pi=pi)
  return(list(C1=C1, C20=C20, C21=C21))
}

capture.sim(N, pi) %>% as.data.frame()
```

```
##   C1 C20 C21
## 1 19  16   3
```

2 Supposons N connu

Supposons tout d'abord que $N = 950$ (connu) et estimons l'efficacité π.

2.1 Estimation par maximum de vraissemblance de π

Nous allons d'abord calculer $\hat{\pi}_{MLE}$, l'estimateur de maximum de vraissemblance de π

$$\begin{aligned}\frac{dl}{d\pi}(N,\pi) &= (c_1+c_2)\frac{1}{\pi} + (2N-c_1-c_2)\frac{1}{1-\pi}(-1)\\ &= (c_1+c_2)\frac{1}{\pi} - (2N-c_1-c_2)\frac{1}{1-\pi}\end{aligned}$$

$$\begin{aligned}\frac{dl}{d\pi}(N,\pi) > 0 &\iff (c_1+c_2)\frac{1}{\pi} - (2N-c_1-c_2)\frac{1}{1-\pi} > 0\\ &\iff (c_1+c_2)\frac{1}{\pi} > (2N-c_1-c_2)\frac{1}{1-\pi}\\ &\iff \frac{c_1+c_2}{2N-c_1-c_2}(1-\pi) > \pi\\ &\iff \pi\left(1+\frac{c_1+c_2}{2N-c_1-c_2}\right) < \frac{c_1+c_2}{2N-c_1-c_2}\\ &\iff \pi\frac{2N}{2N-c_1-c_2} < \frac{c_1+c_2}{2N-c_1-c_2}\\ &\iff \pi < \frac{c_1+c_2}{2N}\end{aligned}$$

On en déduit que :

$$\boxed{\hat{\pi}_{MLE} = \frac{c_1+c_2}{2N}}$$

2.2 Estimation bayésienne de la loi de π

En plus d'une technique de maximum de vraissemblance, nous pouvons nous intéresser à une technique bayésienne pour estimer la loi de π. On choisit pour cela une loi à priori $\beta(a,b)$ pour π, que l'on note $f(\pi) = \pi^{a-1}(1-\pi)^{b-1}$.

$$\begin{aligned}\ln\left([\pi|N,C_1,C_{20},C_{21}]\right) &= l(N,\pi) + \ln\left(f(\pi)\right) + cte\\ &= (c_1+c_2)\ln(\pi) + (2N-c_1-c_2)\ln(1-\pi) + (a-1)\ln(\pi) + (b-1)\ln(1-\pi) + cte'\\ &= (c_1+c_2+a-1)\ln(\pi) + (2N-c_1-c_2+b-1)\ln(1-\pi) + cte''\end{aligned}$$

On reconnait, à une constante près, le logarithme d'une loi $\beta(c_1+c_2+a, 2N-c_1-c_2+b)$.

Donc:

$$\boxed{\pi|_N \sim \beta\left(c_1+c_2+a, 2N-c_1-c_2+b\right)}$$

On en déduit son espérance :

$$\begin{aligned}\mathbb{E}\left(\pi|_N\right) &= \frac{c_1+c_2+a}{c_1+c_2+a+2N-c_1-c_2+b}\\ \mathbb{E}\left(\pi|_N\right) &= \frac{c_1+c_2+a}{2N+a+b}\end{aligned}$$

2.3 Comparaison des deux méthodes

On choisit $a = 1$, $b = 3$. La loi associée est décentrée et donne plus de poids, ce qui représente l'intuition comme quoi l'efficacité de pêche est relativement faible.

```
curve(dbeta(x, 1, 3))
```

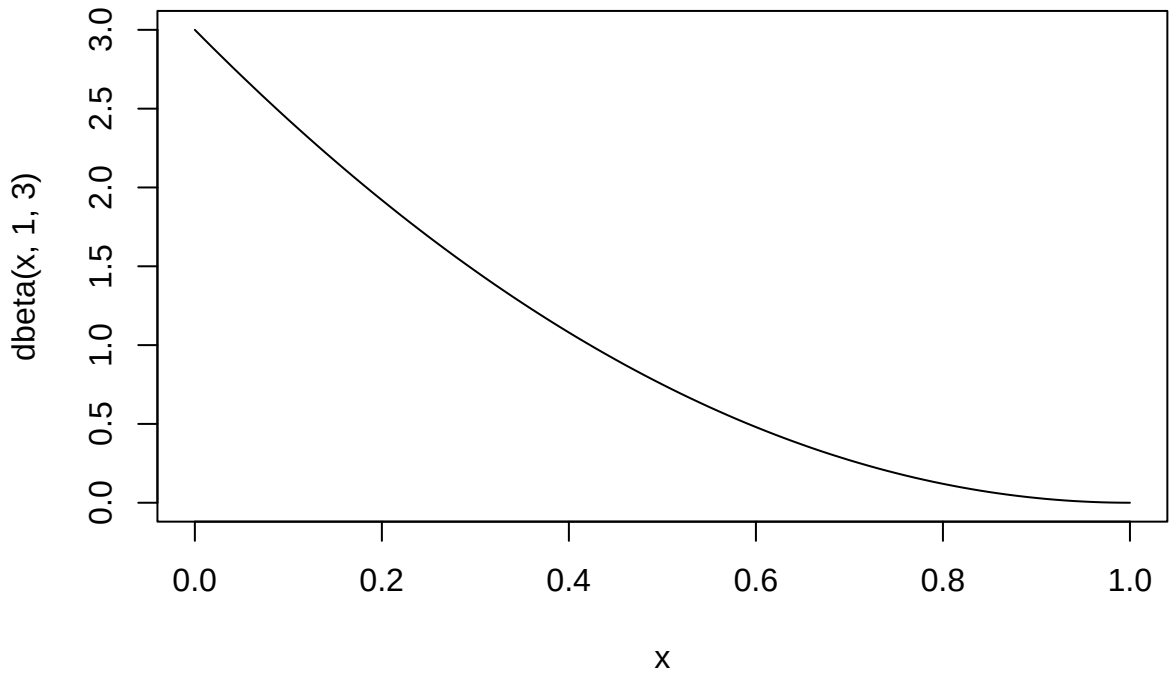

Figure 2: loi Beta(1, 3)

```
set.seed(32)
N = 950
pi = 0.3
df <- capture.sim(N, pi)
C1 <- df$C1
C2 <- df$C20 + df$C21
```

```
a <- 1
b <- 3
a.post <- C1 + C2 + a
b.post <- 2*N - C1 - C2 + b
pi.MLE <- (C1 + C2)/(2*N)

p_val<- seq(0, 1, length.out=100)
```

```r
plot(p_val, dbeta(p_val, a, b),type="l",col="red",ylab="Densité",xlab="pi",
     main=paste("a=",a,"b=",b),ylim=c(0,55))
curve(dbeta(x,a.post,b.post),add=TRUE, col="blue")
abline(v=pi.MLE)
legend("topright", legend=c("a priori", "a posteriori", "max.vraiss"), bty='n',
       pch=rep('_',3), col=c("red","blue","black"))
```

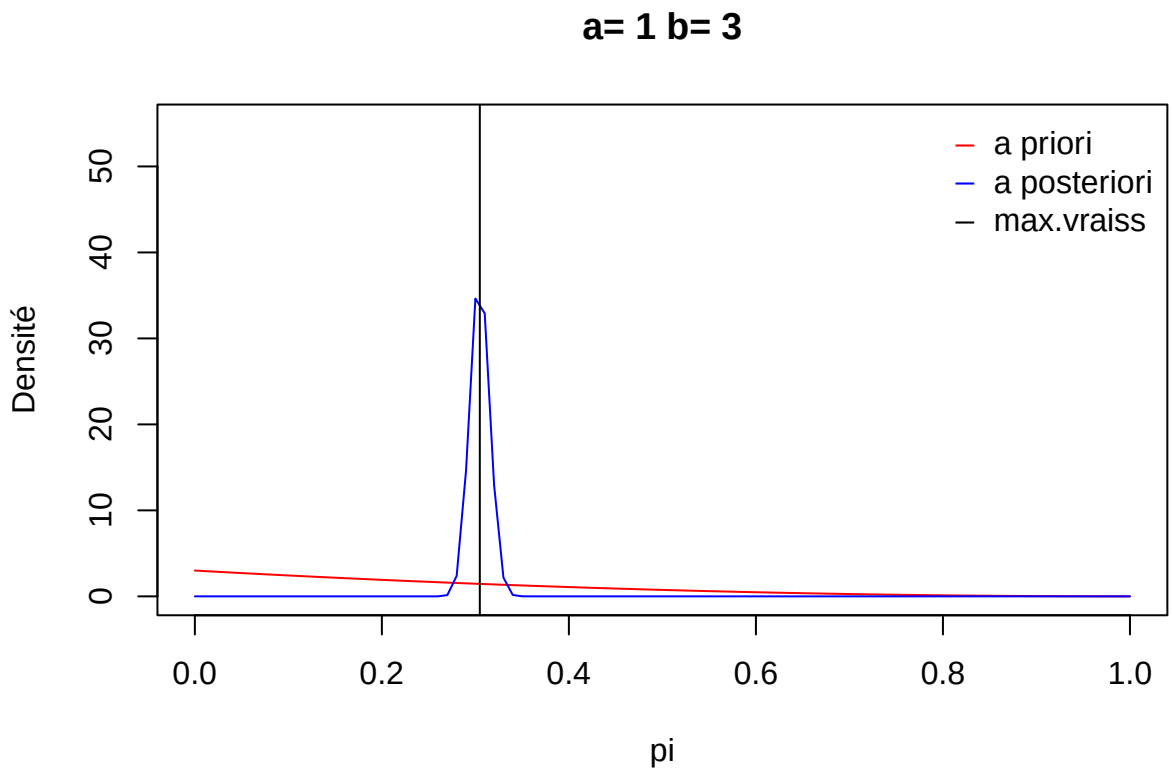

Figure 3: Densités à priori, à postériori et estimateur de maximum de vraissemblance

On remarque d'abord que les données nous ont apporté beaucoup d'information, la densité à postériori étant beaucoup plus piqué et centrée autour d'un unique mode. Ce dernier coincide avec la valeur de l'estimateur de maximum de vraissemblance de π. C'est bon signe, les deux méthodes ont un résultat cohérent.

3 Supposons N et π inconnus

3.1 Approche fréquentiste

3.1.1 Estimateur de Petersen

Pour évaluer le nombre d'individus N dans une population d'intérêt à partir de deux expériences de pêche de type capture-marquage-recapture, un estimateur fréquentiste naïf est l'estimateur de "Petersen" défini par :

$$\hat{N} = \frac{C_1 C_2}{C_{21}}$$

Les données disponibles proviennent d'une expérience réelle "miniature" de capture-marquage-recapture réalisée par des étudiants à l'aide d'un saladier ("le lac") rempli de riz ("l'eau du lac") et de haricots blancs ("les poissons"). Les données observées par les étudiants sont les suivantes : $C_1 = 125$, $C_{20} = 134$ et $C_{21} = 21$.

```
C1 <- 125
C20 <- 134
C21 <- 21
C2 <- C20 + C21

N.hat <- C1*C2/C21
round(N.hat, 0)
```

```
## [1] 923
```

L'estimateur naïf de Petersen nous indique qu'il y a 923 "poissons" dans le lac.

3.1.2 Estimation empirique – par Monte-Carlo – de l'estimateur de Petersen

Supposons ici que les "vraies" valeurs des paramètres soient $N_{true} = 923$ et $\pi_{true} = 0.15$.

```
n.iter <- 100
N.true <- 923
pi.true <- 0.15

N.MC <- function(N, pi, n.iter){
  df.repeated <-
    replicate(n.iter, capture.sim(N, pi)) %>% drop() %>% t() %>% as.data.frame()
  N.hat.repeated <- apply(df.repeated, MARGIN=1,
                          FUN = function(x) x$C1*(x$C20+x$C21)/x$C21)
  df.repeated$N.hat <- N.hat.repeated
  N.monte.carlo <- mean(df.repeated$N.hat)
  return(N.monte.carlo)
}

N.MC(N.true, pi.true, n.iter)
```

```
## [1] 960.7279
```

```
set.seed(57)
N.true.seq <- seq(100, 1000, 10)
N.evolution <- sapply(N.true.seq, FUN=function(n) N.MC(n, pi.true, n.iter))
```

```
plot(N.true.seq, (N.evolution-N.true.seq)/N.true.seq, xlab="N.true", ylab="Biais relatif")
```

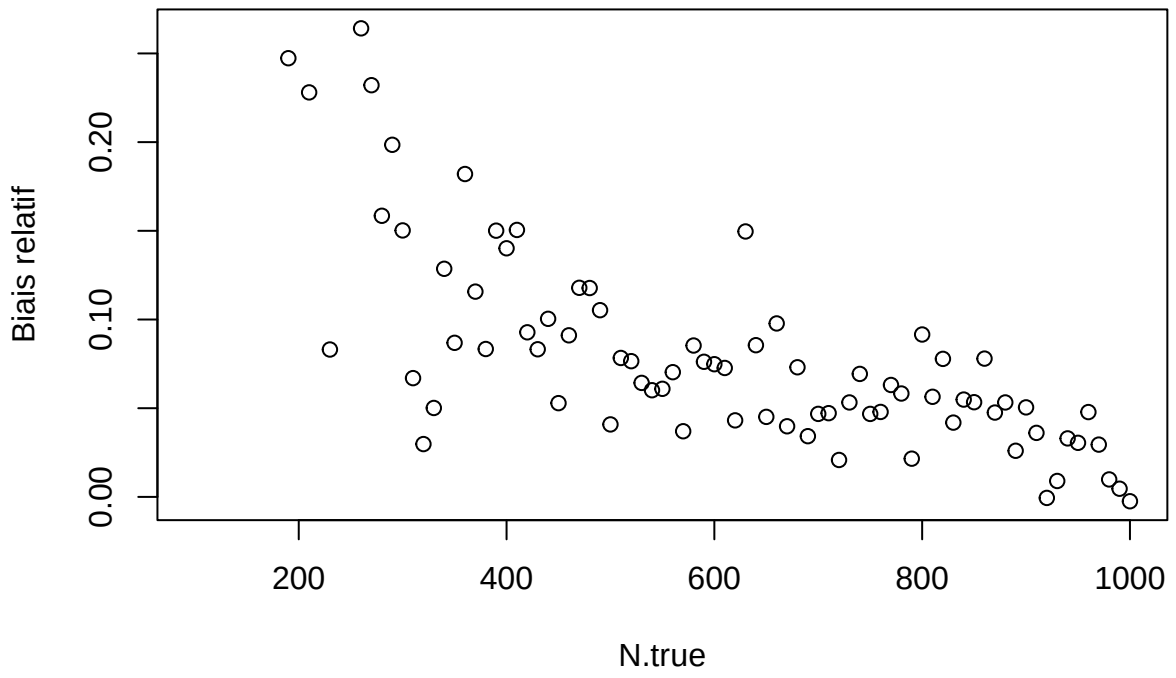

Figure 4: Evolution du biais sur N, en fonction de N.true

On remarque que le biais diminue avec une tendance linéaire jusqu'à atteindre 0.

3.2 Approche bayésienne

3.2.1 Loi conditionnelle complète de N

On choisit une loi *à priori* uniforme sur l'ensemble discret $\{1, ..., 2000\}$ pour N. Sa densité à priori est donc $f(n) = \frac{1}{2000}$, $\forall n \in \{1, ..., 2000\}$.

On rappelle que la vraissemblance du modèle $\mathcal{M}$, dont la log a été calculée précédemment est la suivante :

$$[y|N, \pi] = C_N^{c_1} C_{N-c_1}^{c_{20}} C_{c_1}^{c_{21}} \pi^{c_1+c_2} (1-\pi)^{2N-c_1-c_2}$$

On en déduit donc la forme de la loi à postériori.

$$
\begin{aligned}
[N|y,\pi] &\propto f(n)[y|N,\pi] \\
&\propto \frac{1}{2000} C_N^{c_1} C_{N-c_1}^{c_{20}} C_{c_1}^{c_{21}} \pi^{c_1+c_2}(1-\pi)^{2N-c_1-c_2} \\
&\propto C_N^{c_1} C_{N-c_1}^{c_{20}} (1-\pi)^{2N-c_1-c_2} \\
&\propto C_N^{c_1} C_{N-c_1}^{c_{20}} (1-\pi)^{2N} \\
&\propto \frac{N!}{c_1!(N-c_1)!} \frac{(N-c_1)!}{(c_{20})!(N-c_1-c_{20})!} (1-\pi)^{2N} \\
&\propto \frac{N!}{(N-c_1-c_{20})!} (1-\pi)^{2N} \\
[N|y,\pi] &\propto C_N^{c1+c_{20}} (1-\pi)^{2N}
\end{aligned}
$$

Bien qu'elle ressemble à une loi binomiale, cette loi ne fait pas partie des lois analytiques simples connues. Il sera donc nécessaire d'utiliser un algorithme ne dépendant pas de la simulation de cette loi, lorsque l'on souhaitera simuler sa densité.

3.2.2 Algorithme de Metropolis within Gibbs

On se propose maintenant d'échantilloner dans la loi jointe à postériori du couple (N, π) sachant $y = (c_1, c_{20}, c_{21})$ à l'aide de l'algoritheme de Meetropolis within Gibbs dont l'itération k est la suivante :

- Mise à jour du paramètre π en tirant dans sa loi conditionnelle complète.
- Mise à jour du paramètre N avec l'algorithme de Metropolis-Hastings (MH), en utilisant comme loi de proposition une loi uniforme (discrète) sur $\{N^{curr} - k, N^{curr} + k\}$ où N^{curr} désigne la valeur courrante du paramètre N à une itération donnée et k est un paramètre de saut.

La loi de proposition $Q(x, y) := 1_{\{x-k,x+k\}}(y)$ étant symétrique, le ratio de Métropolis-Hastings se simplifie pour devenir, à l'ittération i :

$$
r_i = \frac{[N^{cand}|\pi, y]}{[N^{i-1}|y]}
$$

où N^{cand} a été tiré selon la loi $Q(N^{i-1}, \cdot)$

```
MH.ratio <- function(N, N.curr, pi){
  #return(ifelse(N>=N.curr, prod((N.curr+1):N), 1/prod((N+1):N.curr))*(1-pi)^(2*(N-N.curr)))
  seq1 <- (N-c1-c20):N
  seq2 <- (N.curr-c1-c20):N.curr
  ratio <- seq1/seq2
  r <- prod(ratio)*((1-pi)^(2*(N-N.curr)))
  return(r)
  }
```

```
MCMC <- function(theta.0, y, n.iter, a, b, k){
  c1 <- y[1]
  c20 <- y[2]
  c21 <- y[3]
  c2 <- c20 + c21
  pi.0 <- theta.0[1]
  N.0 <- theta.0[2]
```

```
  chain <- matrix(0, nrow=n.iter, ncol=2, dimnames=list(NULL,c("pi","N")))
  accept <- vector("numeric", n.iter)

  chain[1, 1:2] <- theta.0
  accept[1] <- 1
  N.curr <- N.0
  pi.curr <- pi.0
  for (i in 2:n.iter){
    #on tire dans la loi conditionnelle complète de pi
    pi.curr <- rbeta(n=1, shape1=c1+c2+a, shape2=2*N.curr-c1-c2+b)

    #loi uniforme discrète
    N.cand <- sample(x=(N.curr-k):(N.curr+k), size=1)

    r <- MH.ratio(N.cand, N.curr, pi.curr)
    u <- runif(n=1, min=0, max=1)

    if (u<min(1,r)){
      N.curr <- N.cand
      accept[i] <- 1
    }

     chain[i,] <- c(pi.curr, N.curr)
  }
  return(list(chain=matrix(chain, nrow=n.iter, ncol=2, byrow=FALSE), taux.accept=mean(accept)))
}
```

```
set.seed(50)
a <- 1
b <- 3
k <- 30
n.iter <- 10000
c1 <- 125
c20 <- 134
c21 <- 21
theta.0=c(pi.0=rbeta(n=1, shape1=a, shape2=b), N.0=sample(134:2000, size=1))
y=c(c1=125, c20=134, c21=21)
MCMC(theta.0, y, n.iter,a,b,k)$taux.accept
```

```
## [1] 0.8836
```

```
k.accept.rate.evolution <- function(k.seq, theta.0,pi.0, n.iter, y, a, b){
  accept.rate.evolution <- sapply(k.seq, FUN=function(x){
    return(MCMC(theta.0, y, n.iter, a, b, x)$taux.accept)})
  df <- as.data.frame(x=cbind(k=k.seq, accept.rate=accept.rate.evolution))

  plt <- ggplot(df, aes(x=k, y=accept.rate), plot=FALSE) + geom_point()

  return(list(plt=plt, df=df))
}
```

```r
set.seed(30)
k.seq <- seq(1, 301, 10)
n.iter <- 10000
a <- 1
b <- 3
y <- c(C1=125, C20=134, C21=21)
pi.0 <- rbeta(n=1, shape1=a, shape2=b)
N.0 <- sample(134:2000, size=1)

theta.0 <- c(pi.0, N.0)
print(theta.0)
```

```
## [1]    0.3680187 1426.0000000
```

```r
list <- k.accept.rate.evolution(k.seq, theta.0,pi.0, n.iter, y, a, b)
plt <- list$plt
df <- list$df

#plot
plt + geom_hline(yintercept = 0.4)
```

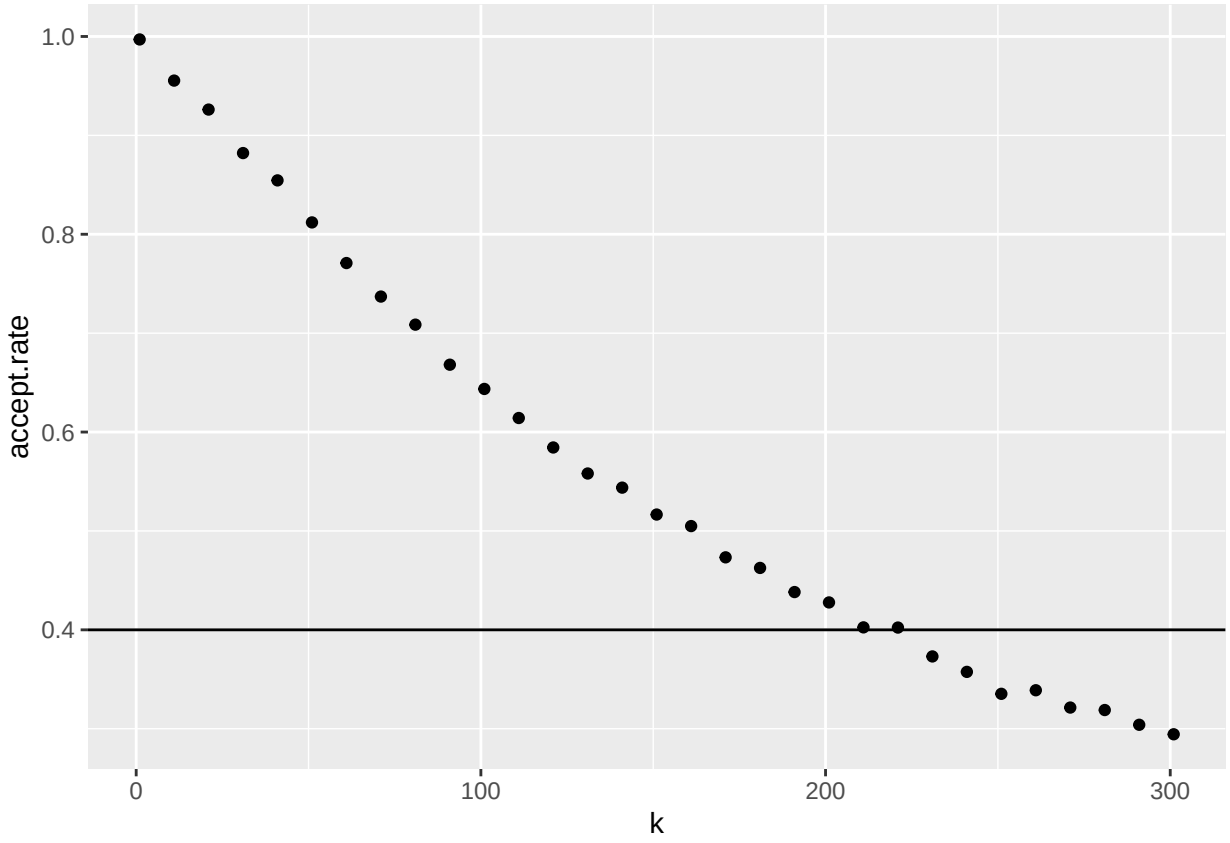

Figure 5: Evolution du taux d'acceptation en fonction de k

```
k.opt <- df$k[which.max(which(df$accept.rate>0.4))]
#221
```

La figure montre une valeur de k optimale valant 221.

3.2.3 Examen de convergence

```
set.seed(92)
G <- 20000
theta.0.list <- sapply(X=1:3, FUN=function(x) c(rbeta(n=1, shape1=a, shape2=b),
            sample(134:2000, size=1)))

chain1 <- MCMC(theta.0=theta.0.list[,1], y, G, a, b, k.opt)$chain
chain2 <- MCMC(theta.0=theta.0.list[,2], y, G, a, b, k.opt)$chain
chain3 <- MCMC(theta.0=theta.0.list[,3], y, G, a, b, k.opt)$chain
```

Examen visuel

```
library(ggmcmc)
library(coda)

mcmc.chains <- mcmc.list(mcmc(chain1), mcmc(chain2), mcmc(chain3))
model.samples.gg <- ggs(mcmc.chains)
ggs_traceplot(model.samples.gg)
```

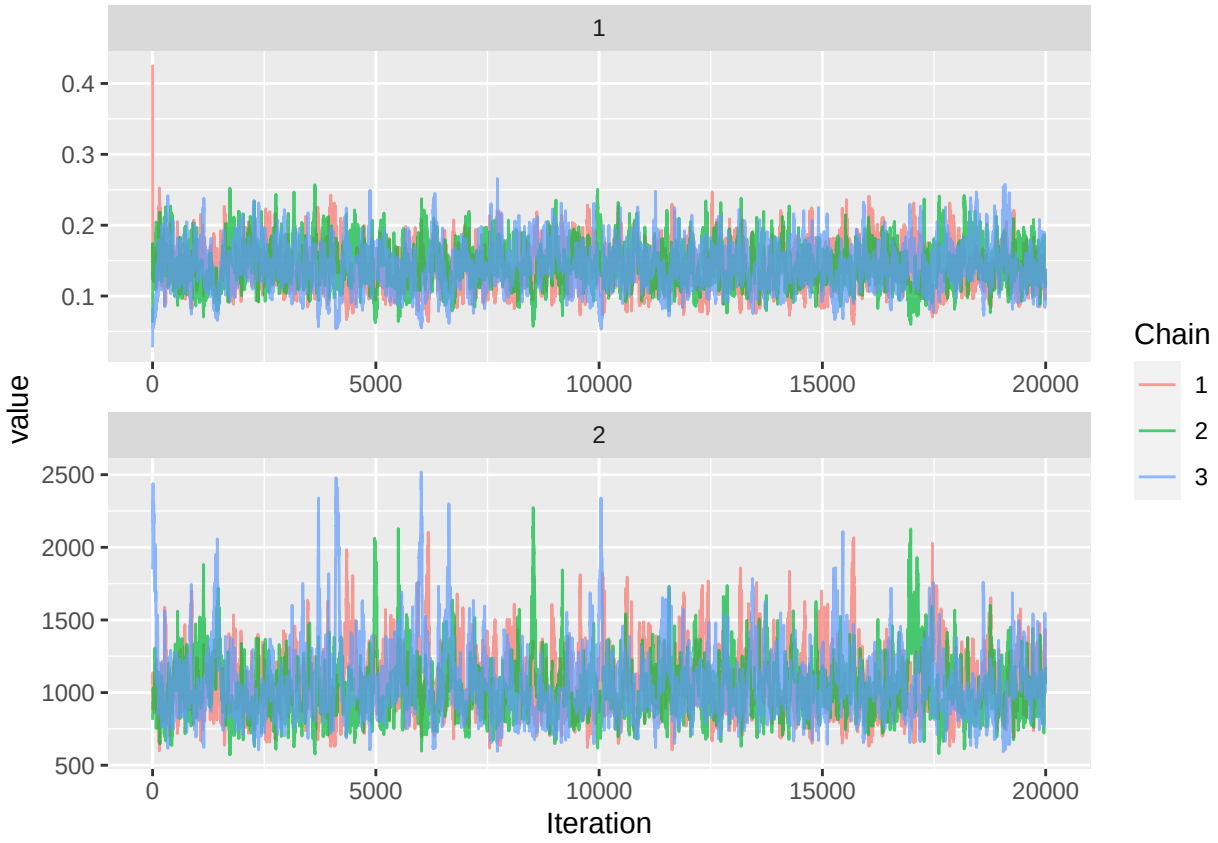

Figure 6: Tracé de l'évolution des chaines pour chaque paramètre

Les chaînes semblent se mélanger convenablement. Elles explorent le même support de valeurs pour les différents paramètres de *theta.0.list*.

```
gelman.diag(mcmc.chains)
```

critère de Gelman-Rubin

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]       1.01       1.02
## [2,]       1.01       1.03
##
## Multivariate psrf
##
## 1.01
```

```
gelman.plot(mcmc.chains)
```

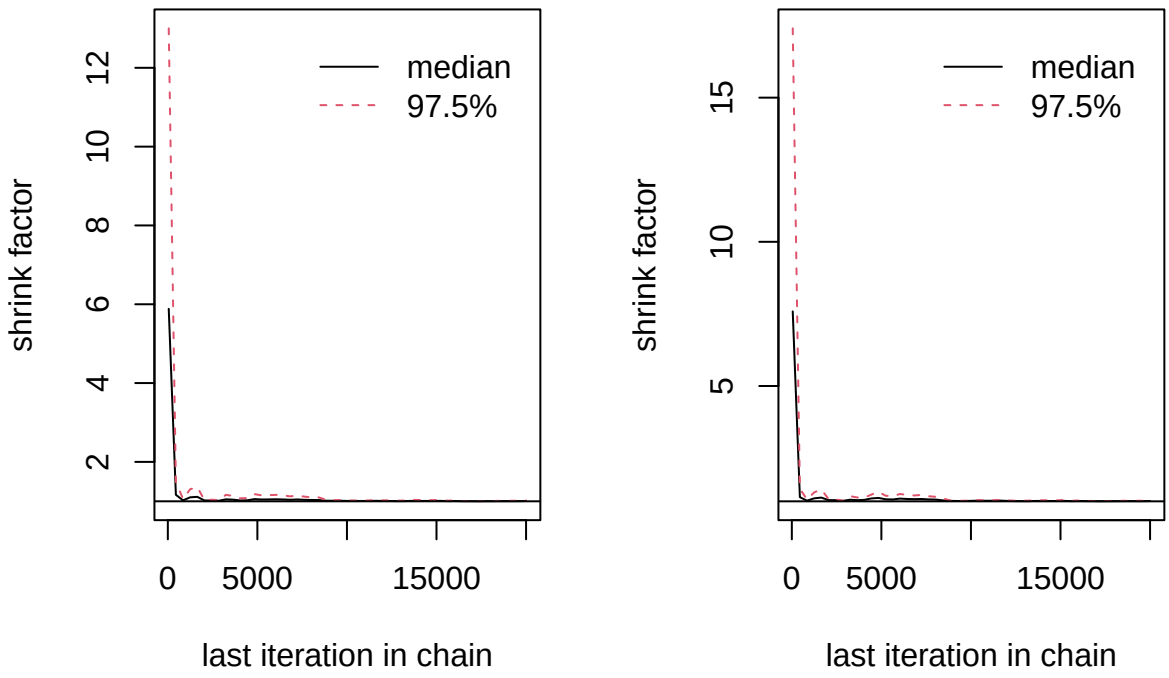

Figure 7: Critère de Gelman-Rubin

Le critère de gelman-Rubin s'utilise ainsi :

Règle : $\hat{R} < 1.05$ et stabilité $\Rightarrow$ pas de problème de convergence majeur diagnostiqué.

Dans notre cas, on voit une nette stabilité du facteur à partir de 10000 itérations.

On identifie donc le temps de chauffe aux 10000 premières itérations, qu'il faudra retirer de la chaîne.

3.2.4 Autocorrélations intra-chaînes

```
ggs_autocorrelation(model.samples.gg)
```

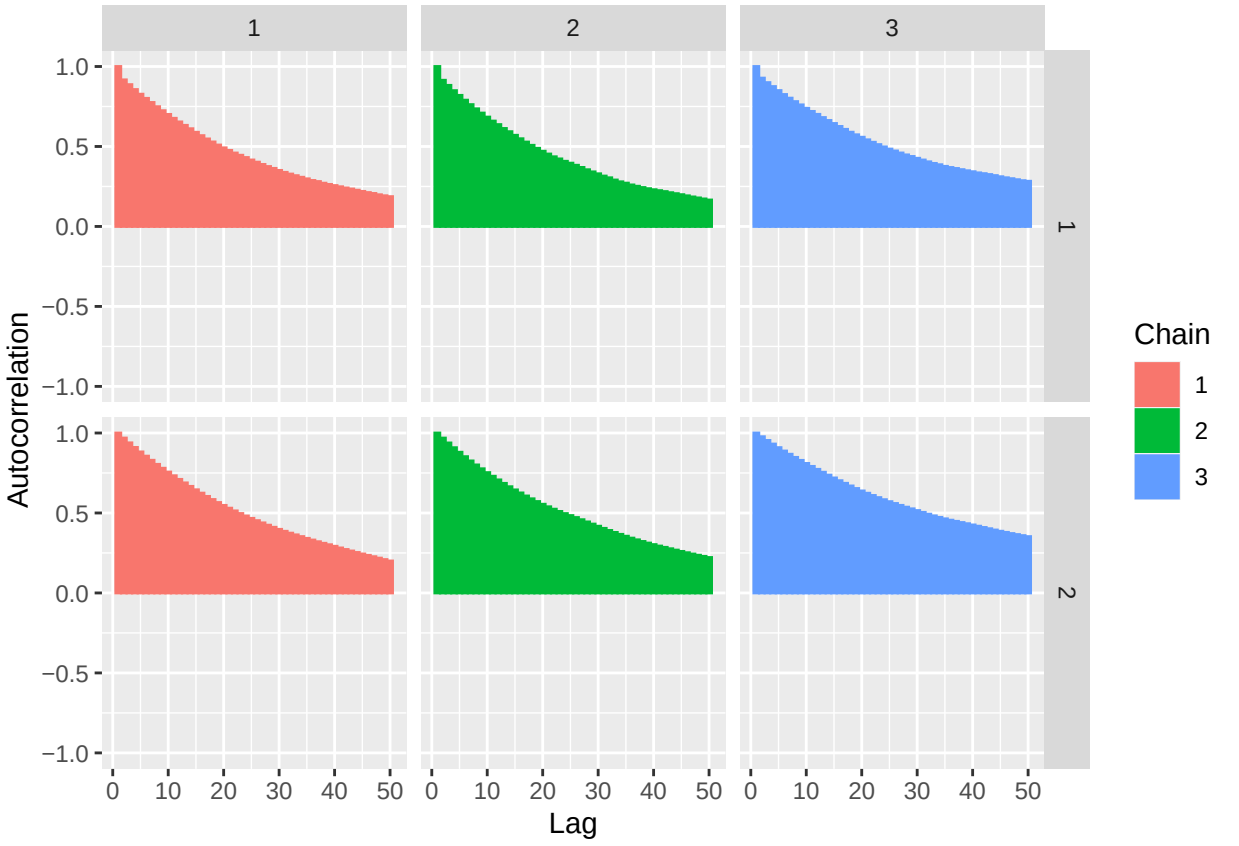

Figure 8: Tracé des autocorrélogrammes des 3 chaines pour les variables pi et N

Les autocorrélogrammes montrent des corrélations encore élevées jusqu'au lag de 40. Il nous faudra à priori un grand nombre de tirages pour avoir un échantillon non corrélé de taille satisfaisante.

3.2.5 Echantillon effectif

```
G0 <- 10000
model.samples <- mcmc.list(mcmc(chain1[-c(1:G0),]), mcmc(chain2[-c(1:G0),]),
                           mcmc(chain3[-c(1:G0),]))
model.samples.gg <- ggs(model.samples)

effectiveSize(model.samples)
```

```
##     var1     var2
## 556.2586 483.5010
```

La taille d'échantillon effective est relativement faible. Il nous faudrait au moins 5000 échantillons. Nous pouvons augmenter le nombre d'itérations et réeffectuer les calculs pour l'augmenter.

```
set.seed(27)
G <- 200000
G0 <- 10000
```

```
chain1 <- MCMC(theta.0=theta.0.list[,1], y, G, a, b, k.opt)$chain
chain2 <- MCMC(theta.0=theta.0.list[,2], y, G, a, b, k.opt)$chain
chain3 <- MCMC(theta.0=theta.0.list[,3], y, G, a, b, k.opt)$chain

model.samples <- mcmc.list(list(mcmc(chain1[-c(1:G0),]), mcmc(chain2[-c(1:G0),]),
                           mcmc(chain3[-c(1:G0),])))

effectiveSize(model.samples)
```

```
##     var1     var2
## 9846.126 8886.420
```

```
Respost<- summary(model.samples)
Respost$statistics[,4]< 0.05*Respost$statistics[,2]
```

```
## [1] TRUE TRUE
```

L'erreur de Monte-Carlo est inférieure à 5% de l'ecart-type à posteriori empirique. On en déduit que la moyenne empirique de l'échantillon a posteriori de (N, π) est un bon estimateur de Monte-Carlo de l'espérance a posteriori $\mathbb{E}(N, \pi|y)$

```
print(Respost)
```

```
##
## Iterations = 1:190000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 190000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean        SD Naive SE Time-series SE
## [1,]    0.1401   0.02778 3.68e-05        0.00028
## [2,] 1039.3014 209.91377 2.78e-01        2.22751
##
## 2. Quantiles for each variable:
##
##           2.5%      25%       50%      75%     97.5%
## [1,]   0.09023   0.1205    0.1386    0.158    0.1986
## [2,] 715.00000 890.0000 1010.0000 1155.000 1532.0000
```

Les déviations standards sont nettement plus petites que les valeurs moyennes déterminées. Nous pouvons donc avoir confiance en les valeurs moyennes obtenues.

Nous obtenons finalement les estimations suivantes pour les valeurs de N et π :

$$\begin{cases} \hat{\pi} & = 0.14 \\ \hat{N} & = 1039 \end{cases}$$

```
plot(mcmc.chains, density=TRUE, trace=FALSE)
```

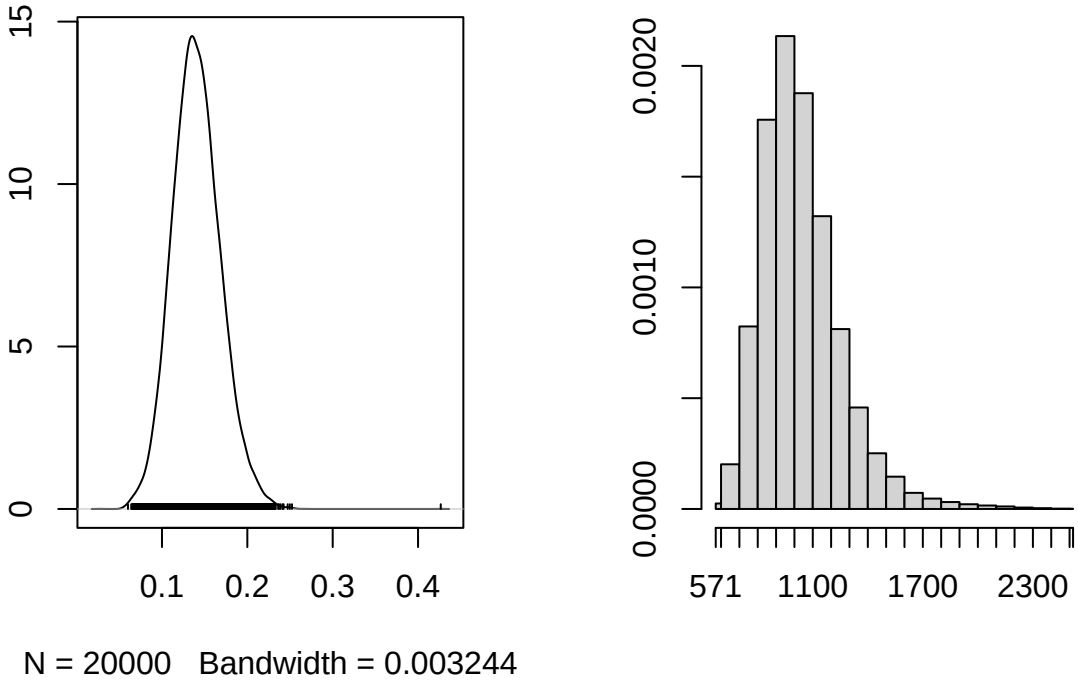

Figure 9: Lois à postériori approchées (pi à gauche, N à droite)