Mise en oeuvre de l'inférence bayésienne avec **JAGS** et `rjags`

M. L. Delignette-Muller
VetAgro Sup - LBBE

5 décembre 2018

Exemple support

Modélisation de la loi dose-réponse associée à l'ingestion de Clostridium perfringens.

- **Partie déterministe** du modèle, probabilité que l'hôte soit malade :

$$p = 1 - (1 - r)^{dose}$$

avec *dose* le nombre de cellules ingérées

- **Partie stochastique** du modèle, le nombre de malades *Nmal* pour N hôtes exposés :

$$Nmal \sim Binomiale(n = N, p = 1 - (1 - r)^{dose})$$

Formalisation classique d'un modèle à l'aide d'un DAG - Directed Acyclic Graph ?

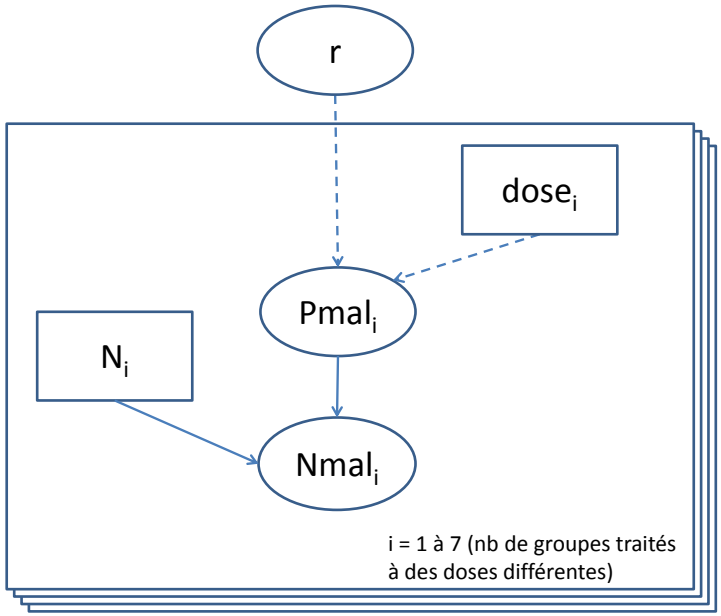
Qu'est-ce qu'un DAG ?

- **un graphe dirigé**
  (les liens ont un sens)
- **sans cycle**
  (partant d'un noeud et suivant les liens on ne peut revenir à ce noeud)
- qu'on utilise en statistique bayésienne pour représenter les **relations de dépendances conditionnelles** entre les noeuds
  Description mécaniste de l'acquisition des données : noeuds sortants

Formalisme des DAG

- **Noeuds du graphe**
  - covariable (rectangle)
  - variable (ellipse)
    variable observée, variable latente ou variable intermédiaire
- **Liens du graphe**
  - lien déterministe (ou logique - flèche en pointillés - lien dont on pourrait se passer en écrivant le modèle autrement)
  - lien stochastique (flèche en trait plein - lien indispensable)

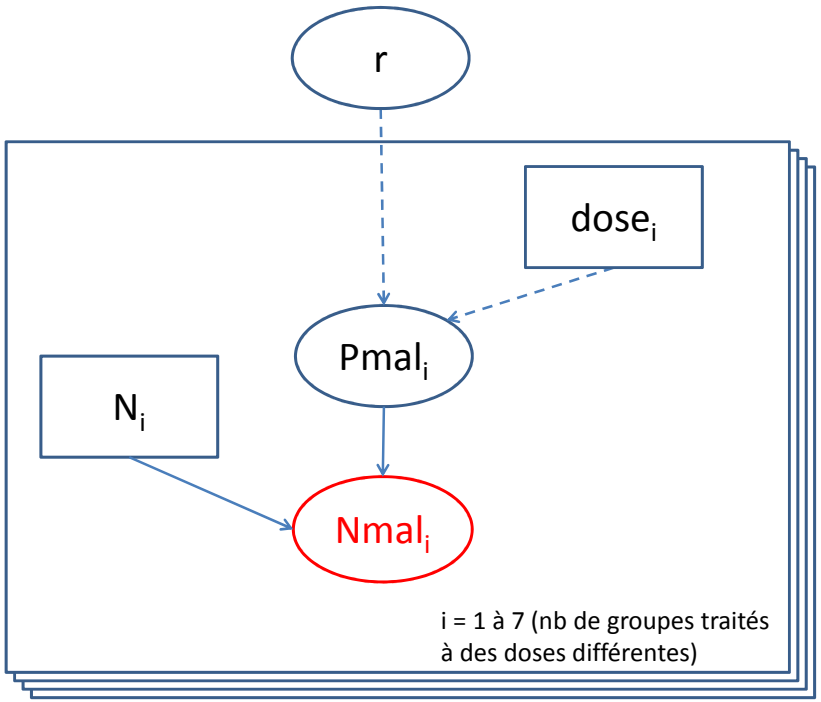
DAG du modèle de l'exemple support



Définition mathématique des liens

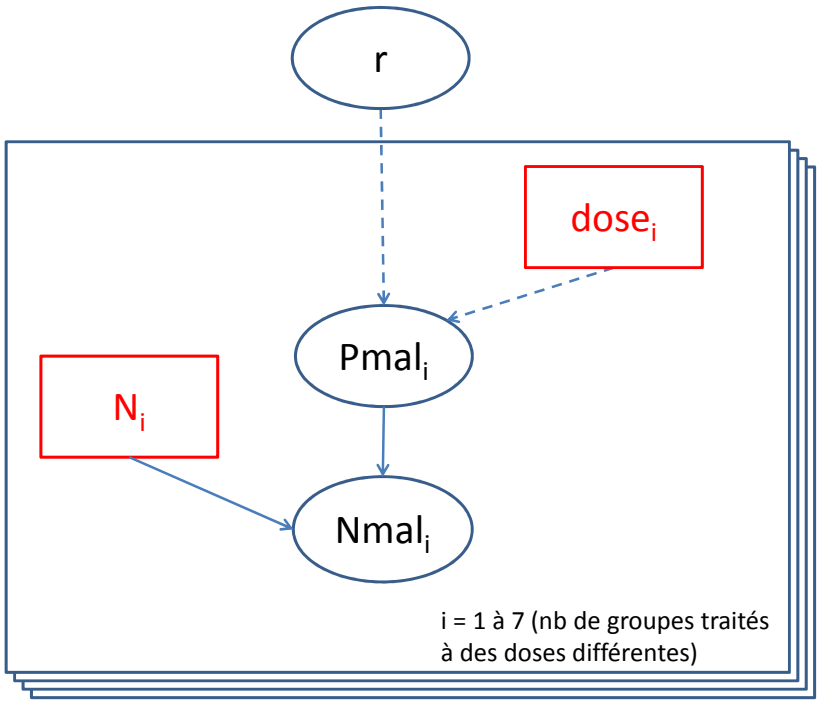
- liens déterministes
  $Pmal_i = 1 - (1 - r)^{dose_i}$
- liens stochastiques
  $Nmal_i \sim Binomiale(N, Pmal_i)$

DAG du modèle - données (vraisemblance)



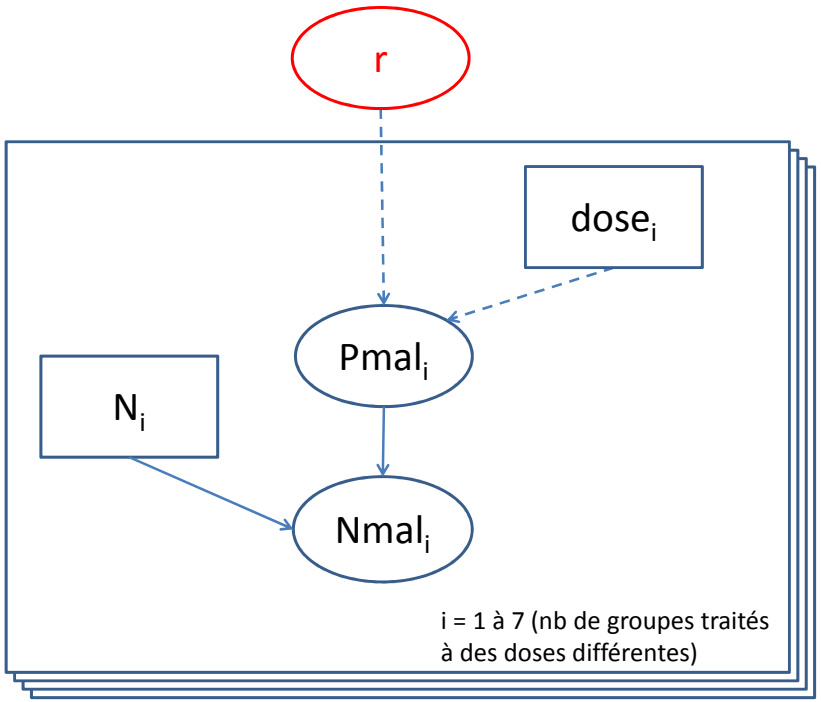
Données

DAG du modèle - covariables (variables explicatives)



Covariables

DAG du modèle - paramètres (à estimer)



Paramètres

Information *a priori*

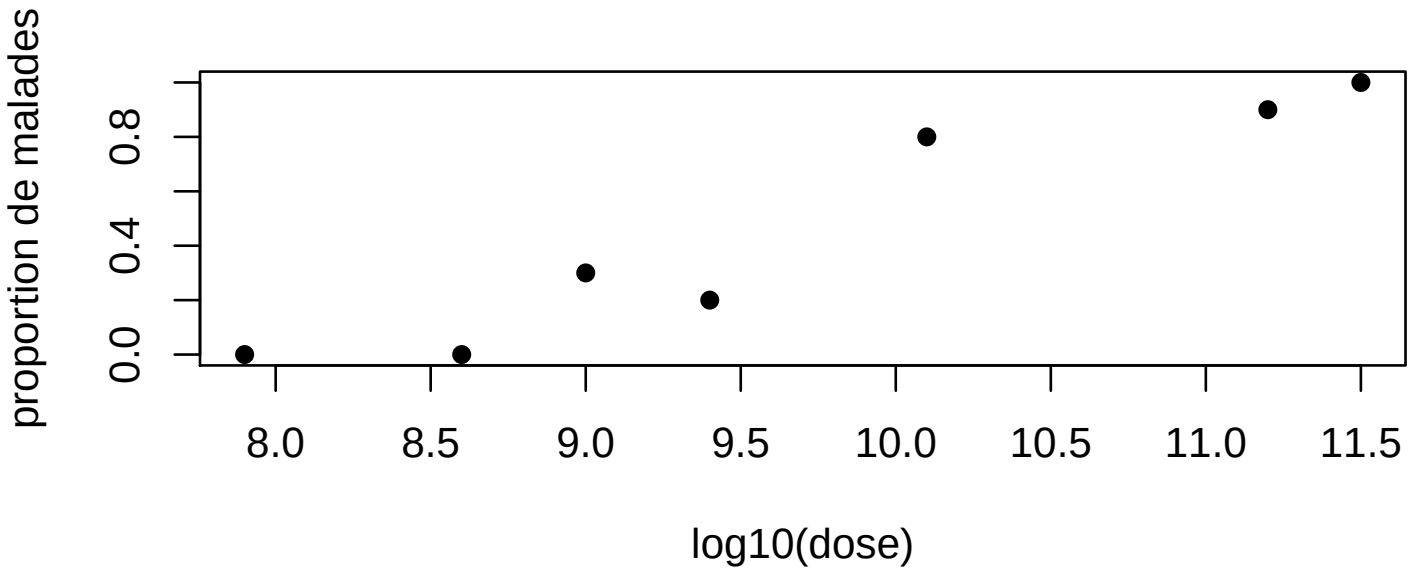
Dans cet exemple, on considérera qu'il est raisonnable de définir comme loi *a priori* sur l'unique paramètre du modèle :

- une loi uniforme entre -15 et -5 sur $log_{10}(r)$,

Données associées à l'exemple

Nombre de malades $Nmal_i$ pour chaque groupe de taille N_i exposé à la dose $dose_i$

```
> plot(Nmal/N ~ doselog10, data = d, pch = 16,
+ xlab = "log10(dose)", ylab = "proportion de malades")
```



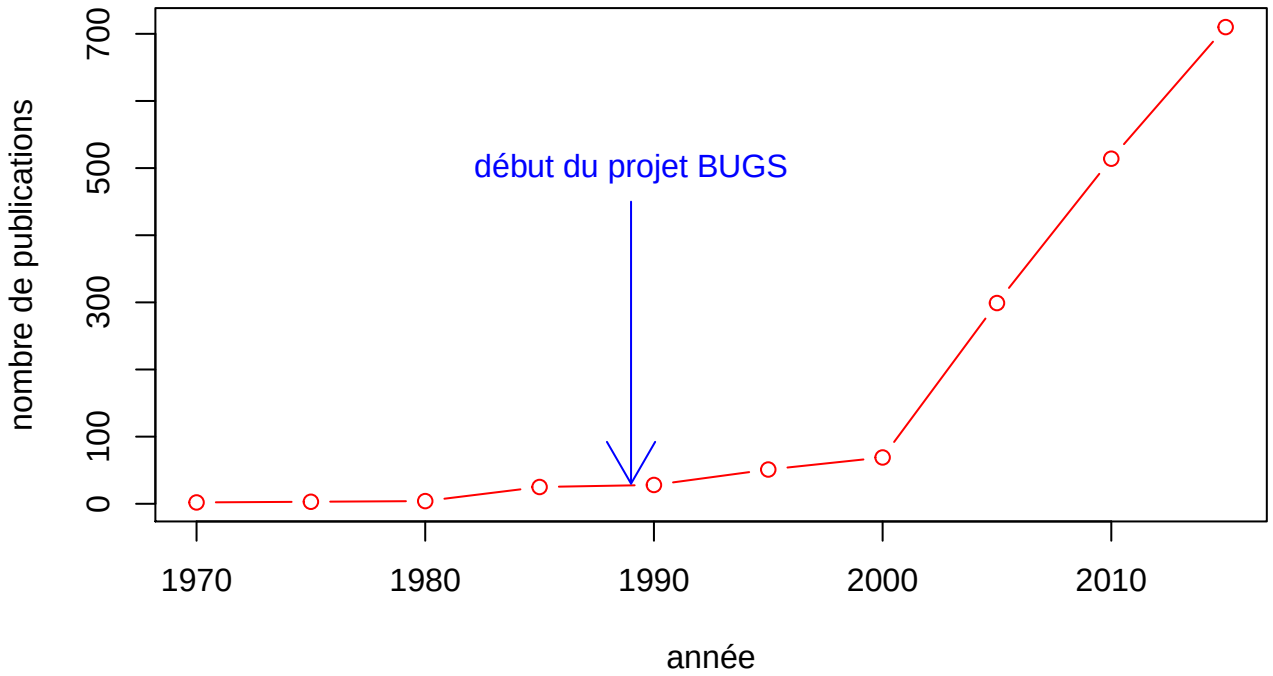
Projet BUGS (depuis 1989)

Bayesian inference **U**sing **G**ibbs **S**ampling
Développement et mise à disposition de logiciels flexibles permettant de mettre en oeuvre l'inférence bayésienne sur des modèles complexes, en utilisant les MCMC.
Quelques outils disponibles :

- WinBUGS and OpenBUGS
- **JAGS (Just Another Gibbs sampler - Martyn Plummer)**
- Stan et Nimble (nouveaux algorithmes en plus des MCMC)
- RevBayes (pour la phylogénie)
- multiples outils spécifiques à une famille de modèles

Evolution de l'utilisation de la statistique bayésienne

Recherche dans *Pubmed* d'articles contenant **bayesian** dans le titre



Codification d'un modèle en langage de type BUGS

Langage déclaratif
(l'ordre des lignes de commande n'a pas d'importance)
qui ressemble à **R**

- **déclaration d'un noeud déterministe**

```
noeud <- fonction(certains autres noeuds)
```

- **déclaration d'un noeud stochastique**
  y compris noeuds entrants (lois *a priori*)

```
noeud ~ distributions(certains autres noeuds)
```

ATTENTION : un noeud sur lequel on a des données doit toujours être codé par un lien stochastique !

Code du modèle de l'exemple

A écrire dans un fichier texte ou dans une chaîne de caractères comme ci-dessous.

```
> model <-
+ "model
+ {
+   for (i in 1:Ndose)
+   {
+     pmal[i] <- 1 - (1 - r)^dose[i]
+     Nmal[i] ~ dbin(pmal[i], N[i])
+   }
+   log10r ~ dunif(-15, -5)
+   r <- 10^log10r
+ }
+ "
```

Quelques principes des langages de type BUGS

Un noeud est univarié.
Il faut spécifier les dimensions, les indices,
et **écrire des boucles** pour définir des vecteurs ou matrices ou tableaux multidimensionnels (`array`)
On peut par exemple écrire :

```
v[]       v[i]
M[,]      M[i,j]
A[,,,]    A[i,j,k,l]
M[,j]     v[n:m]
x[y[i]]   x[2*j-1]
```

Code du modèle expliqué pas à pas - données

Utilisation ici d'une boucle pour définir toutes les observations

```
model
{
   for(i in 1:Ndose)
   {
      Nmal[i] ~ dbin(pmal[i], N[i])
   }
}
```

Code du modèle expliqué pas à pas - variables intermédiaires

Tous les noeuds doivent être définis sauf les covariables (l'ordre des lignes n'importent pas).

```
model
{
   for(i in 1:Ndose)
   {
      Nmal[i] ~ dbin(pmal[i], N[i])
      pmal[i] <- 1 - (1 - r)^dose[i]
   }
}
```

Code du modèle expliqué pas à pas - lois *a priori*

A priori défini en dehors de la boucle car ne change pas d'une observation à l'autre ici.

```
model
{
   for(i in 1:Ndose)
   {
      Nmal[i] ~ dbin(pmal[i], N[i])
      pmal[i] <- 1 - (1 - r)^dose[i]
   }
  log10r ~ dunif(-15, -5)
  r <- 10^log10r
}
```

Formalisme parfois différent de celui utilisé dans **R**

ATTENTION,
le formalisme est parfois différent de celui utilisé dans **R**, notamment concernant le nom des distributions et leur paramétrisation.
Se référer au manuel d'utilisation de JAGS ou des autres langages de type BUGS pour avoir une liste complète et à jour des fonctions et distributions disponibles.
Manuel de JAGS :
`http://sourceforge.net/projects/mcmc-jags/files/Manuals/`

Codification des données

La codification est logicielle-dépendante.
Nous utiliserons ici **JAGS** (MCMC) et **rjags** (préparation des données, examen des MCMC via **coda**)

```
> require(rjags)
```

```
> data4jags <- list(dose = 10^d$doselog10,
+                   N = d$N,
+                   Nmal = d$Nmal,
+                   Ndose = nrow(d))
```

ATTENTION à la cohérence entre les noms utilisés dans le modèle et dans les données

Tous les noeuds apparaissant dans le modèle mais non définis dans le modèle (ici *dose*, *N*)
(donc apparaissant uniquement à droite d'un opérateur) ainsi que les indices max de boucle (*Ndose*)
doivent être définis dans les données.
Pour que l'inférence puisse se faire,
les sorties du modèle doivent aussi être définies dans les données.
ATTENTION à utiliser les mêmes noms dans les données et le code du modèle !

Définition des valeurs initiales

Là encore la codification est logicielle-dépendante.
(ici **JAGS** et **rjags**)
Codification des valeurs initiales nécessaire pour chaque noeud entrant et chaque chaîne si l'on veut utiliser ensuite la statistique de Gelman et Rubin pour juger de la convergence (sinon les chaînes partent toutes d'une même valeur définie par défaut à partir de la loi *a priori*).

Ex.

```
> ini <- list(list(log10r = -12),
+                  list(log10r = -11),
+                  list(log10r = -10))
```

Simulations

- **Construction du modèle ici avec 3 chaînes** incluant une phase d'adaptation de l'algorithme de `n.adapt` = 1000 itérations par défaut

```
> m <- jags.model(file = textConnection(model),
+                 data = data4jags, inits = ini,
+                 n.chains = 3, n.adapt = 1000)
```

n.adapt correspond au nombre d'itérations de la phase d'adaptation de l'algorithme durant laquelle les valeurs simulées ne sont pas des MCMC.

- **Phase de chauffe (burnin)**

```
> update(m, 3000)
```

- **Simulations monitorées**

```
> mc <- coda.samples(m, c("r"), n.iter = 1000)
> # en pratique on démarre plutôt avec n.iter = 5000 environ
```

Simulations

- **Construction du modèle ici avec 3 chaînes**
  incluant une phase d'adaptation de l'algorithme de `n.adapt` = 1000 itérations par défaut

```
> m <- jags.model(file = textConnection(model),
+                 data = data4jags, inits = ini,
+                 n.chains = 3, n.adapt = 1000)
```

  n.adapt correspond au nombre d'itérations de la phase d'adaptation de l'algorithme durant laquelle les valeurs simulées ne sont pas des MCMC.

- **Phase de chauffe (burnin)**

```
> update(m, 3000)
```

- **Simulations monitorées**

```
> mc <- coda.samples(m, c("r"), n.iter = 1000)
> # en pratique on démarre plutôt avec n.iter = 5000 environ
```

Simulations

- **Construction du modèle ici avec 3 chaînes**
  incluant une phase d'adaptation de l'algorithme de `n.adapt` = 1000 itérations par défaut

```
> m <- jags.model(file = textConnection(model),
+                 data = data4jags, inits = ini,
+                 n.chains = 3, n.adapt = 1000)
```

  n.adapt correspond au nombre d'itérations de la phase d'adaptation de l'algorithme durant laquelle les valeurs simulées ne sont pas des MCMC.

- **Phase de chauffe (burnin)**

```
> update(m, 3000)
```

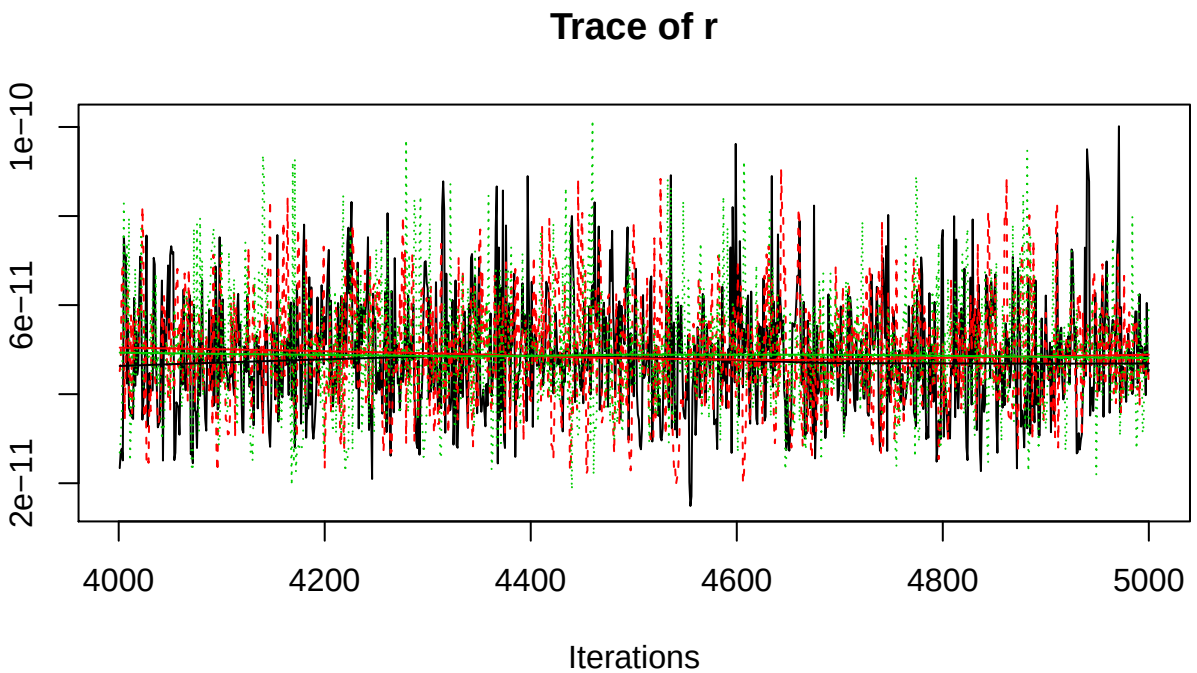
- **Simulations monitorées**

```
> mc <- coda.samples(m, c("r"), n.iter = 1000)
> # en pratique on démarre plutôt avec n.iter = 5000 environ
```

Tracé des chaînes MCMC

Toutes les chaînes doivent converger vers la même limite en distribution (stabilité et recouvrement des chaînes). Ici le recouvrement semble à peu près correct.

```
> plot(mc, density = FALSE)
```

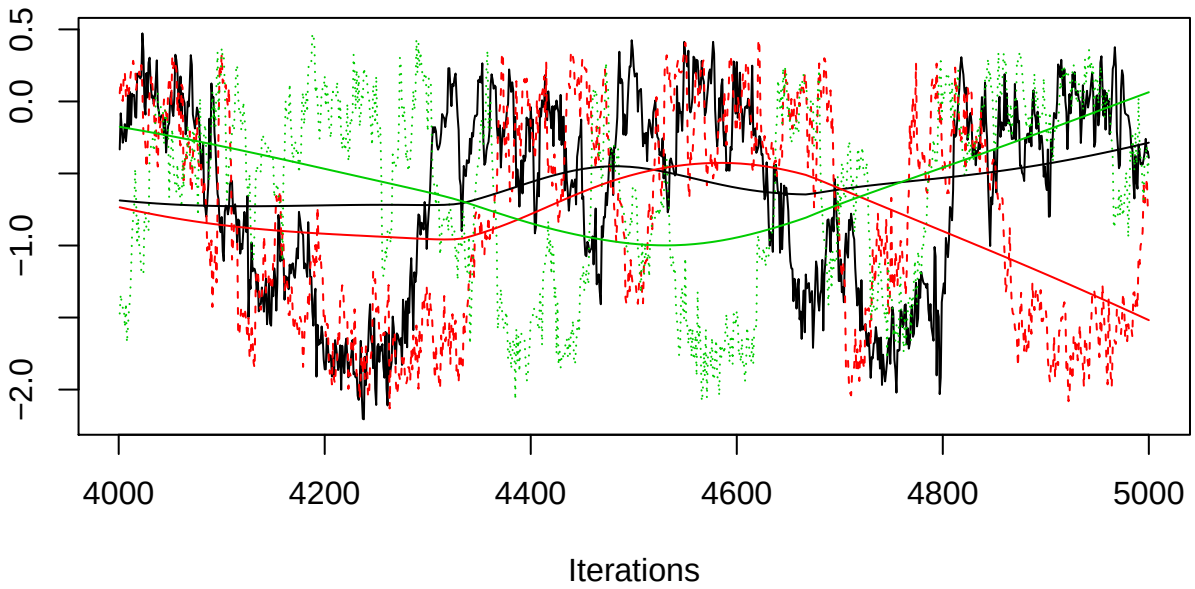


Calcul du critère de Gelman-Rubin

Pour chaque paramètre, racine carrée du rapport entre la variance de sa distribution *a posteriori* marginale et la variance intra-chaîne, qu'on attend à 1 lorsque la convergence est atteinte. Ici le recouvrement semble à peu près correct.
Gelman indique comme valeur max acceptable 1.1 pour tous les noeuds tout en indicant qu'il faut être plus strict pour atteindre un résultat final précis (le but est d'atteindre 1.00).

```
> gelman.diag(mc)

Potential scale reduction factors:

  Point est. Upper C.I.
r          1       1.01
```

Exemple de chaînes MCMC qui se recouvrent mal

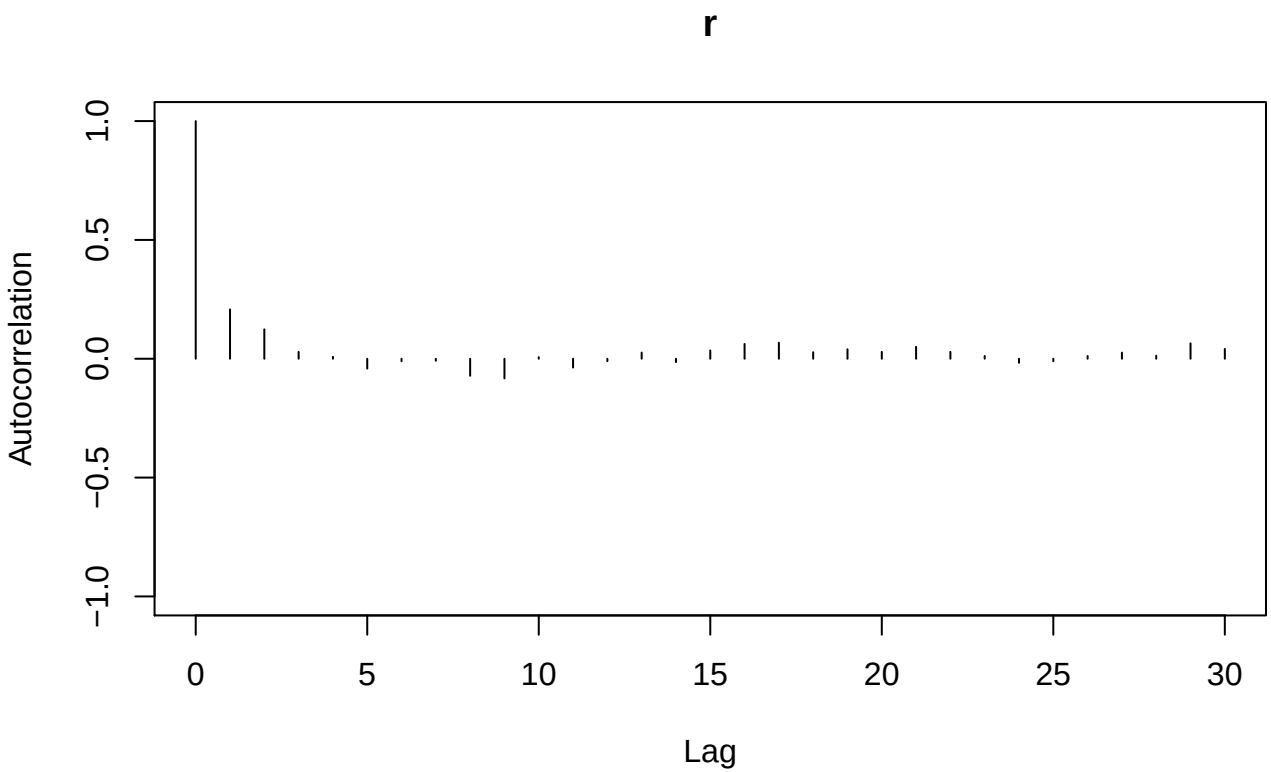



```
> gelman.diag(mc3.3c)

Potential scale reduction factors:

          Point est. Upper C.I.
l10alpha        1.02       1.07
```

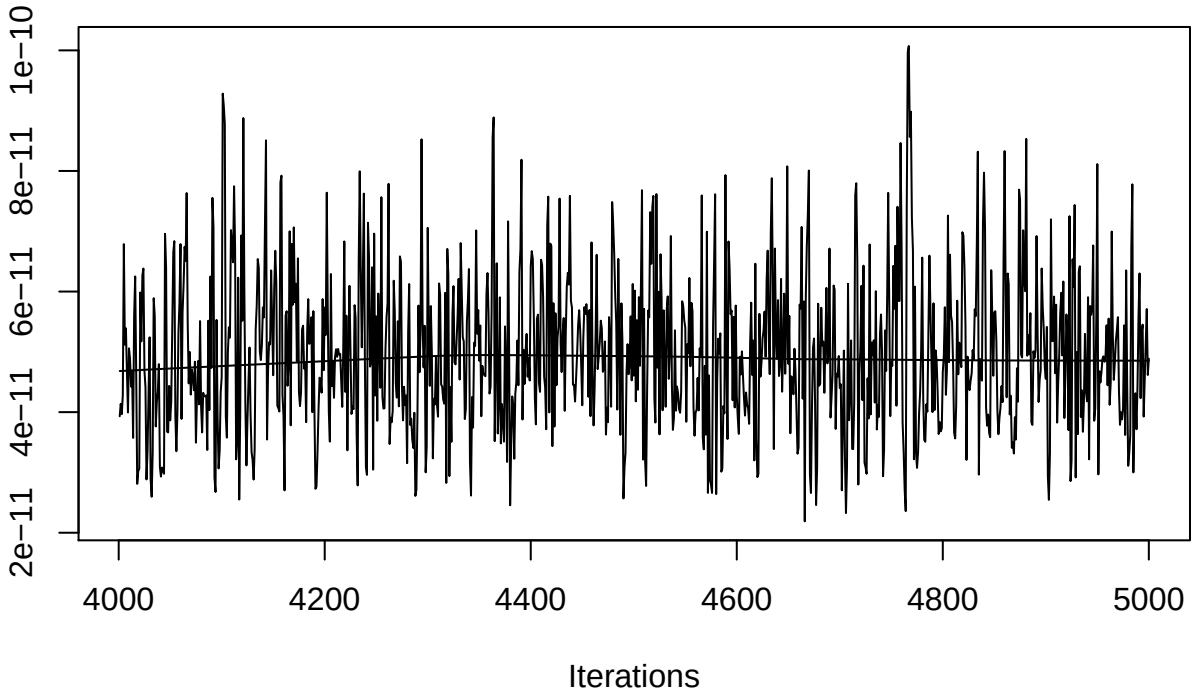
Visualisation de l'autocorrélation

Pour chaque chaîne, autocorrélation entre les itérations en fonction de l'écart entre les itérations. **Ici l'autocorrélation est faible**.

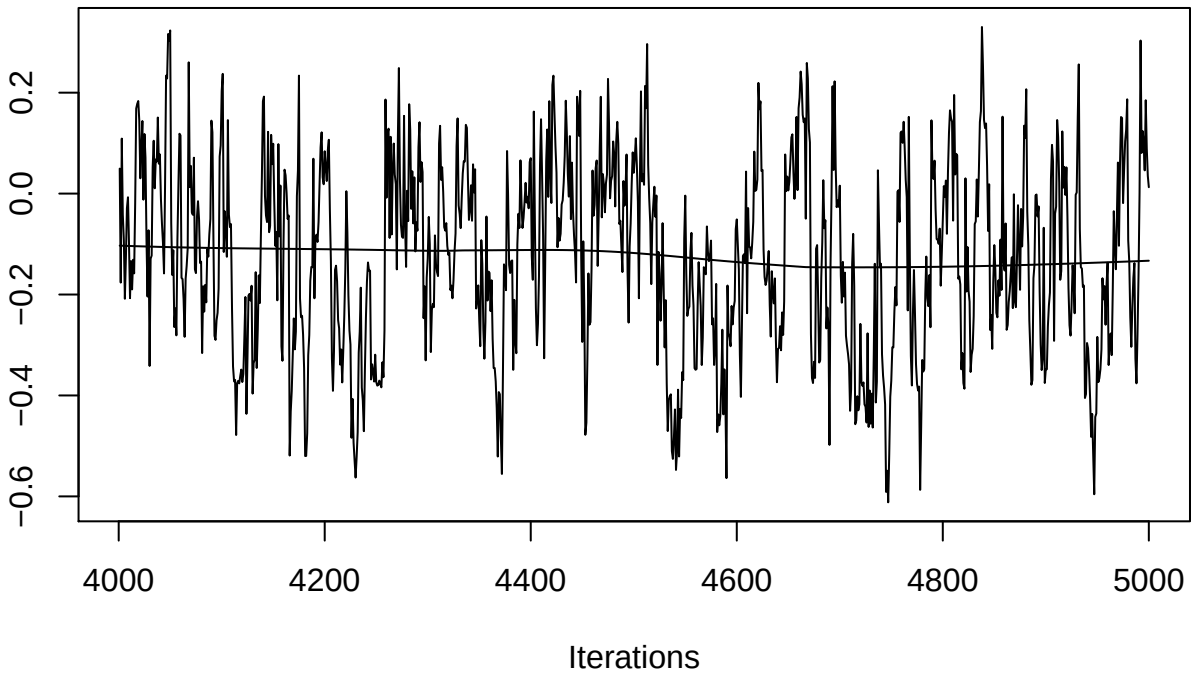
```
> autocorr.plot(mc[[1]])
```



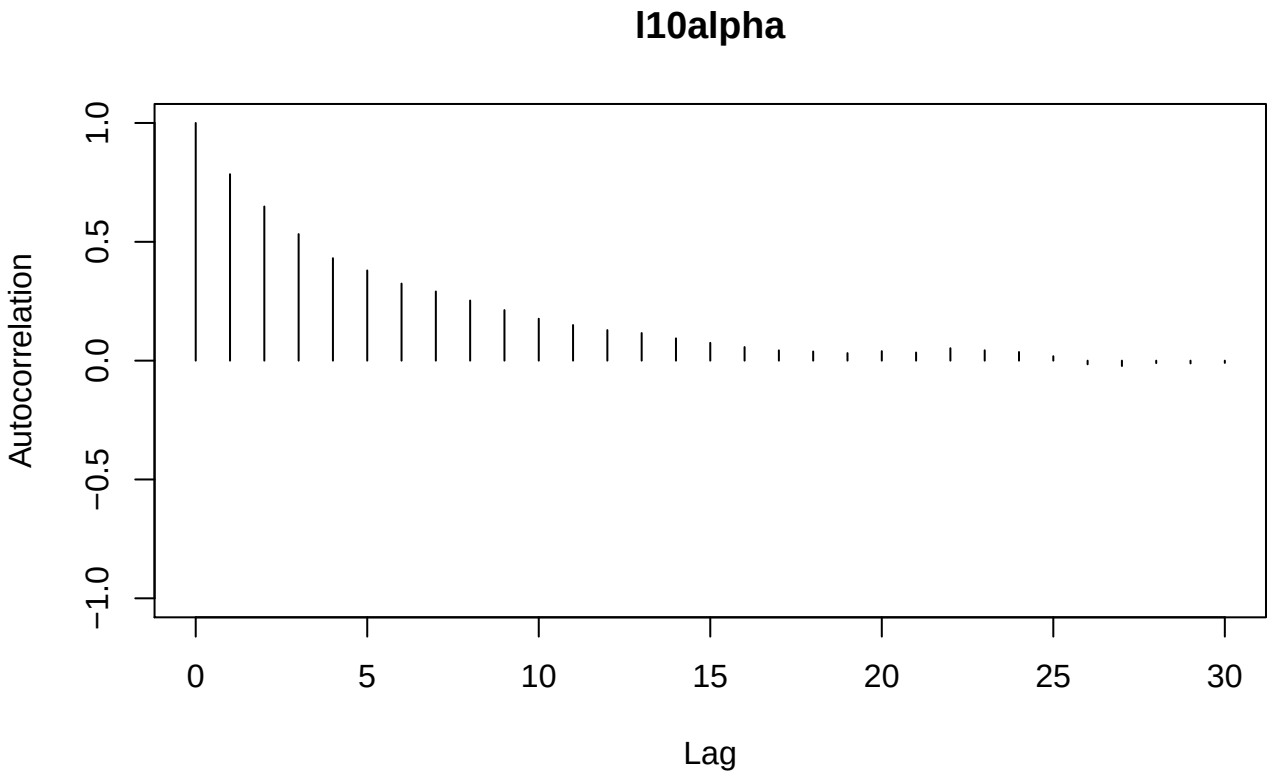
Trace d'une chaîne peu autocorrélée (satisfaisant)



Trace d'une chaîne plus autocorrélée (non satisfaisant)

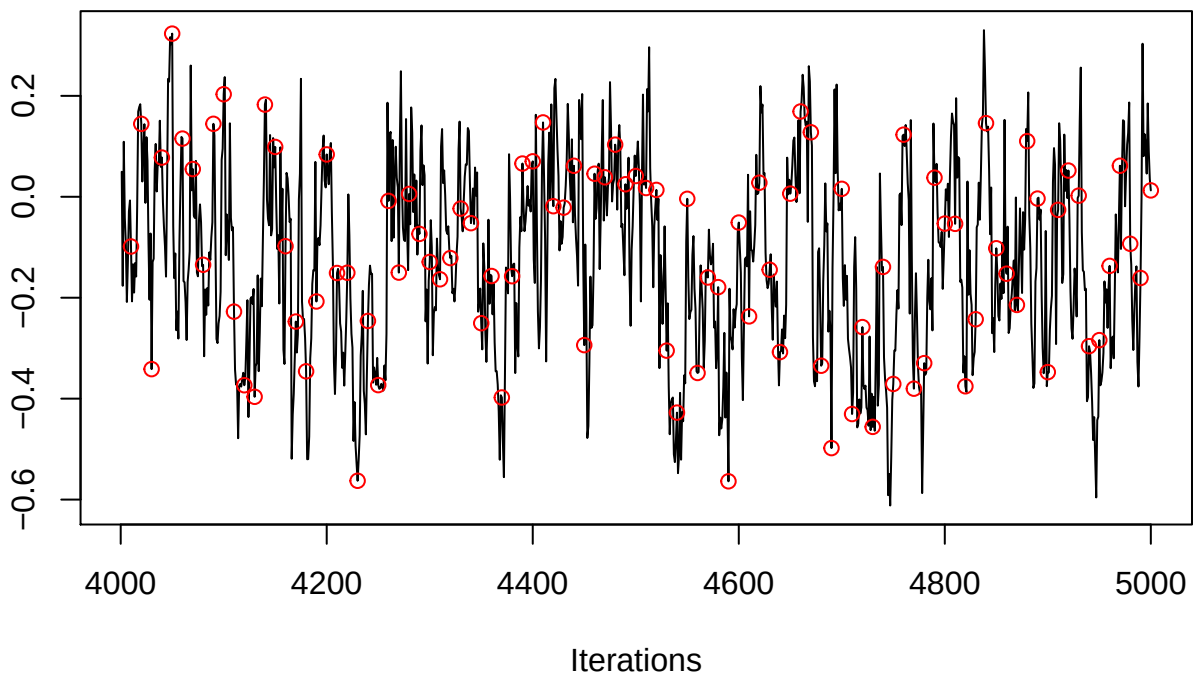


Graphe d'autocorrélation d'une chaîne très autocorrélée



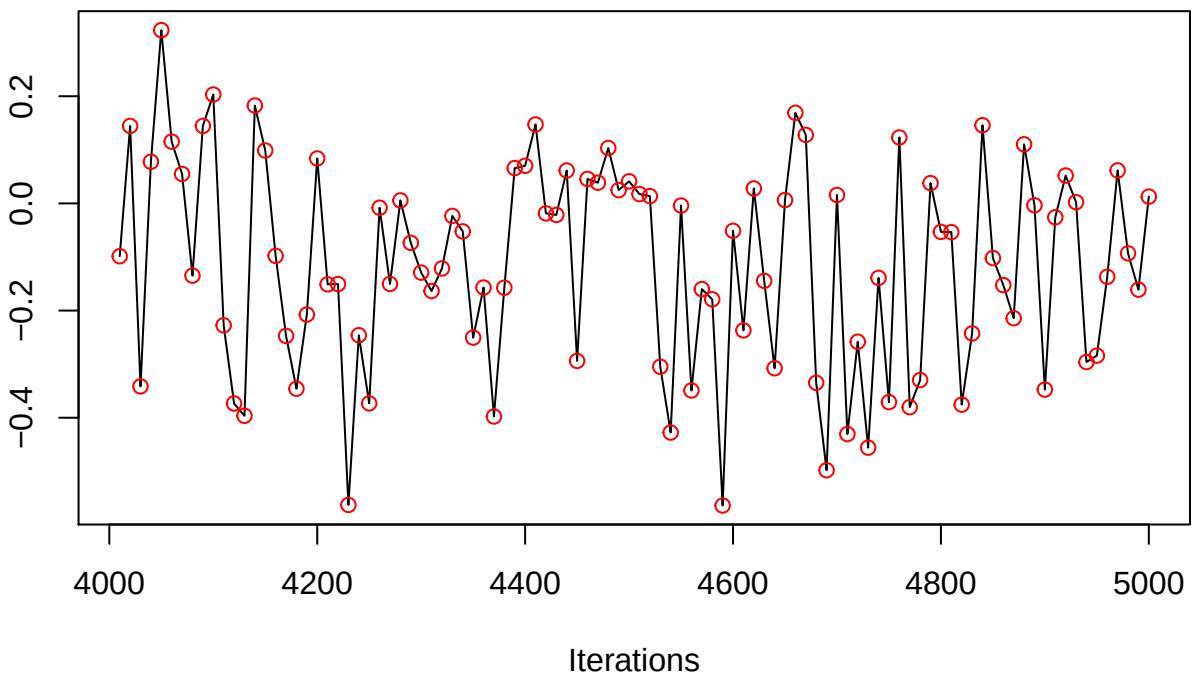
Traitement de l'autocorrélation par amincissement (thinning)

Avec un thin de 10 on garde une itération sur 10.



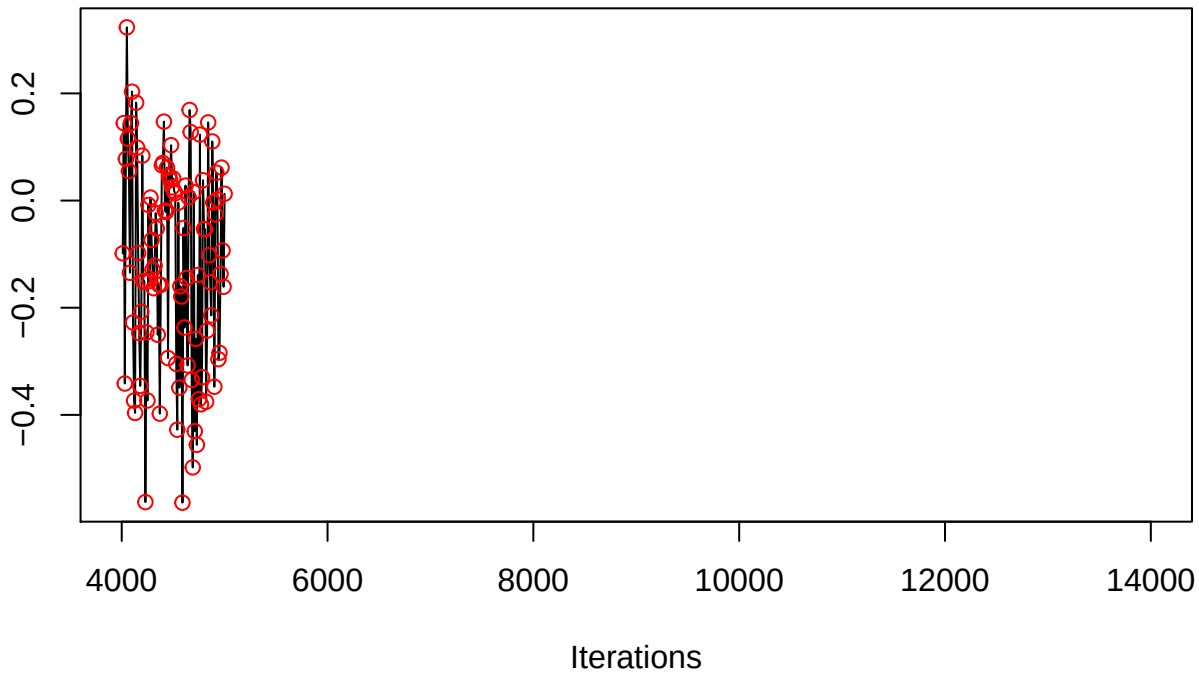
Traitement de l'autocorrélation par amincissement (thinning)

Itérations gardées : ici 100 sur les 1000 initiales.



Traitement de l'autocorrélation par amincissement (thinning)

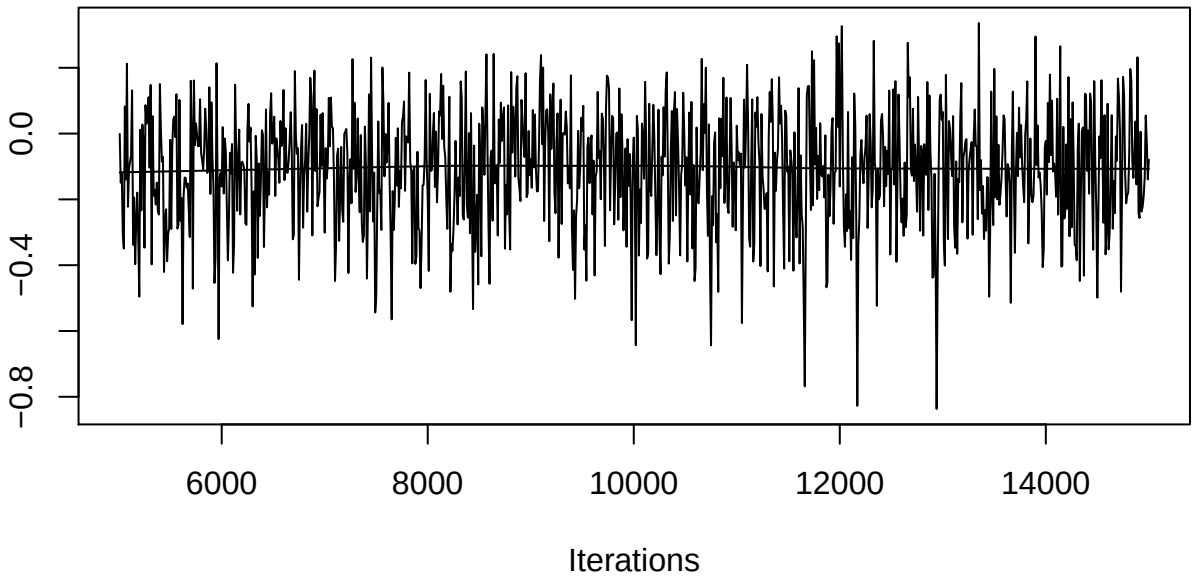
Du coup on a beaucoup moins d'itérations utilisables (ici 100 uniquement).



Traitement de l'autocorrélation par amincissement (thinning)

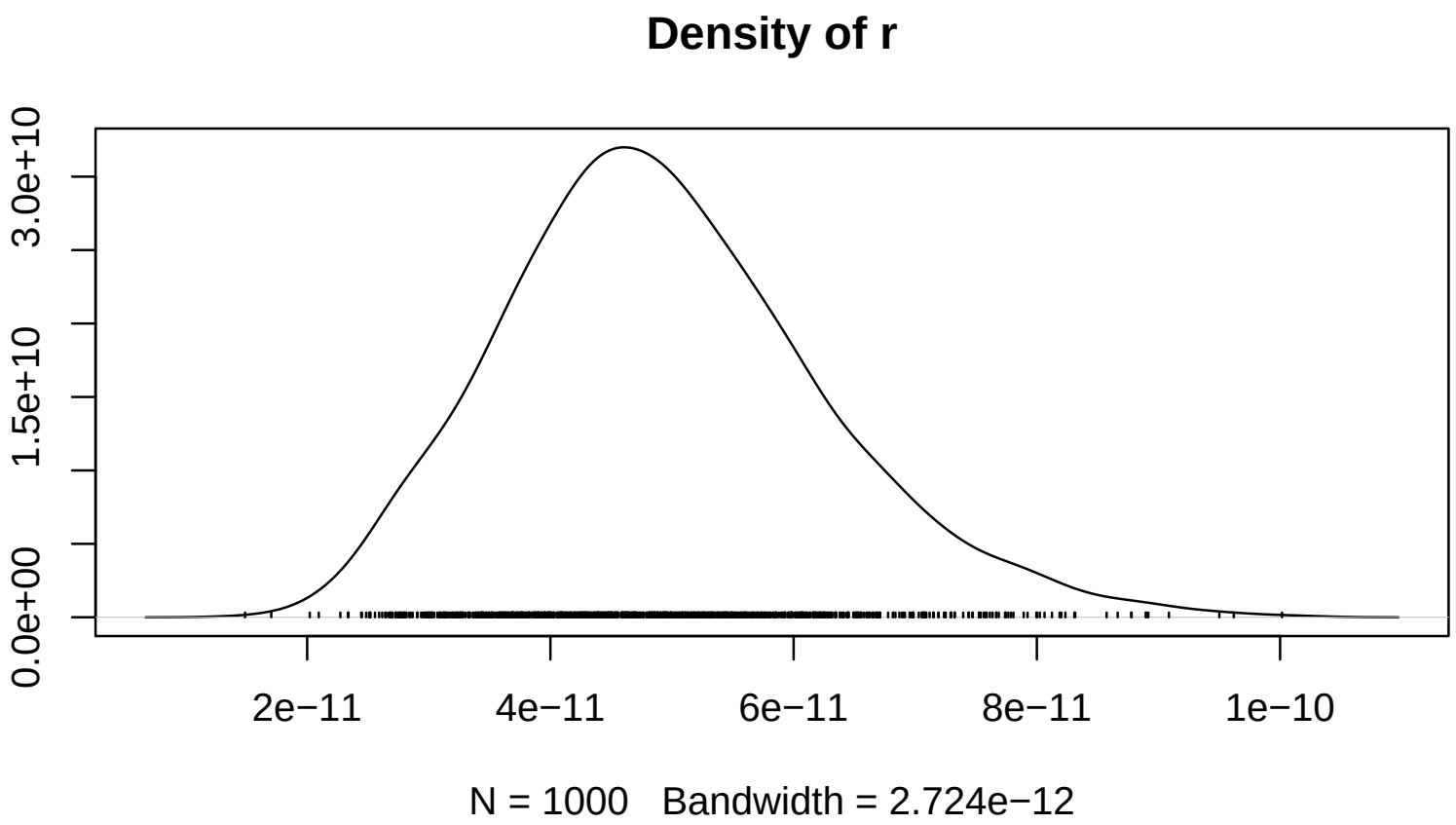
Il faut donc augmenter le nombre d'itérations au départ (ici on le multiplie par 10 $\rightarrow$ calcul plus long).

```
> mc3.1c <- coda.samples(m3.1c, c("l10alpha"), n.iter = 10000, thin = 10)
> plot(mc3.1c, density = FALSE, main = "")
```



Visualisation de la loi *a posteriori* en densité

```
> plot(mc, trace = FALSE)
```



Résumé statistique

```
> summary(mc)

Iterations = 4001:5000
Thinning interval = 1
Number of chains = 3
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean             SD       Naive SE Time-series SE
      4.92e-11       1.31e-11       2.39e-13       0.00e+00

2. Quantiles for each variable:

    2.5%      25%      50%      75%    97.5%
2.69e-11 4.00e-11 4.82e-11 5.71e-11 7.85e-11
```

Intervalles de crédibilité pour les paramètres

- **Intervalles classiquement basés sur quantiles à 2.5% et 97.5%**

```
> summary(mc)$quantiles
```

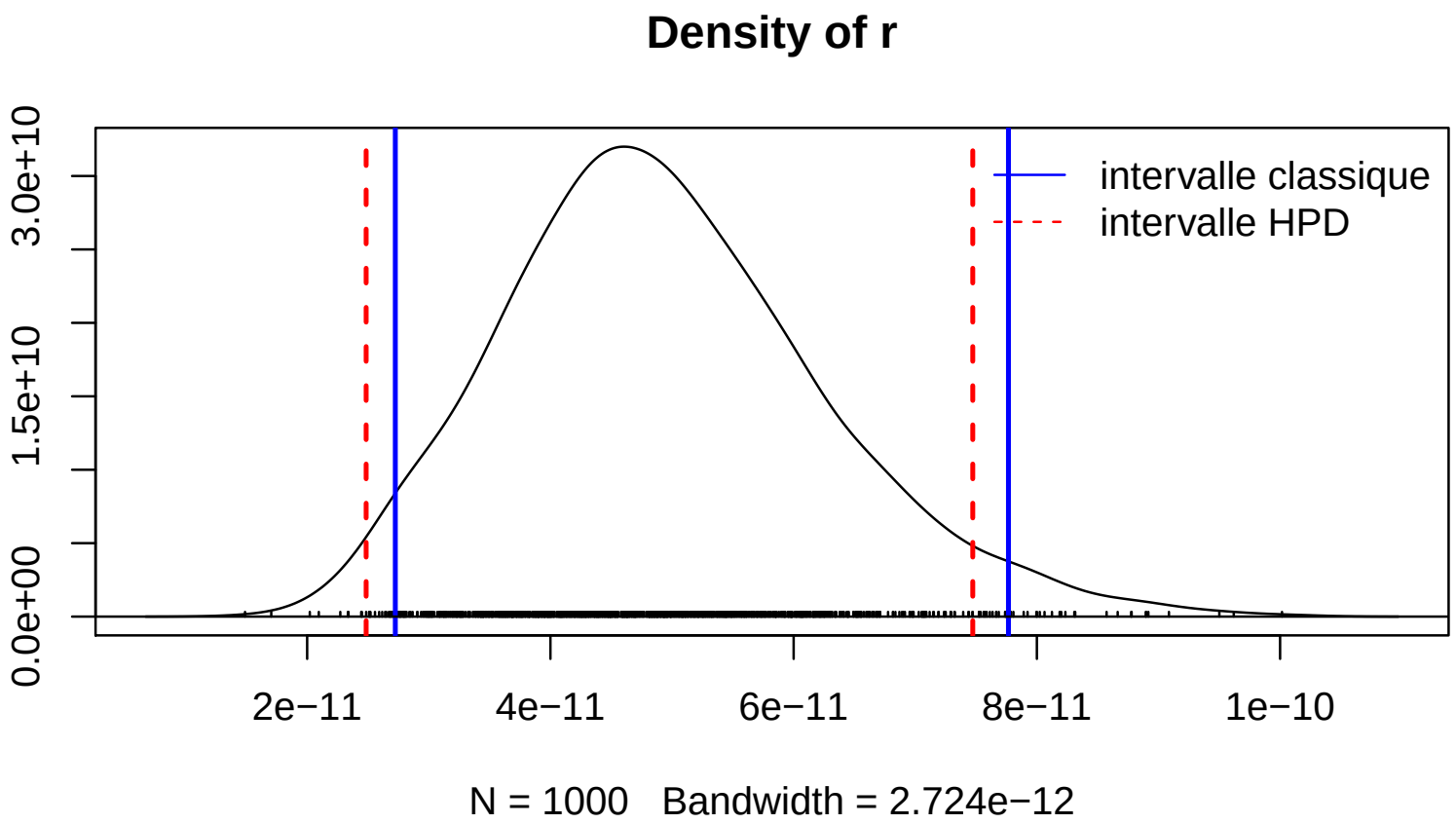
```
    2.5%       25%       50%       75%     97.5%
2.69e-11 4.00e-11 4.82e-11 5.71e-11 7.85e-11
```

- **Intervalles HPD (High Posterior Density)**

```
> HPDinterval(mc[[1]], prob = 0.95) # ici pour chaîne 1
```

```
     lower    upper
r 2.48e-11 7.47e-11
attr(,"Probability")
[1] 0.95
```

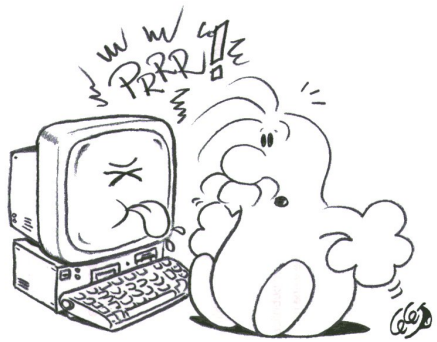
Deux types d'intervalles de crédibilité, un peu différents sur une distribution *a posteriori* assymétrique



Conclusion

Maintenant à vous de jouer !

Pour apprendre les aspects techniques, rien ne vaut la pratique (prochain TD) !



Vous disposerez du guide d'introduction à **JAGS** et `rjags` pour vous aider à démarrer et aller plus loin (notamment aborder les aspects prédiction et validation à partir des MCMC).