



- runMCMCbtadjust
- (Projet de ) package R pour faciliter l'utilisation de « Monte-Carlo Markov Chains » (MCMC) en statistique Bayésienne

Frédéric Gosselin, UR EFNO, INRAE, Nogent-sur-Vernisson  
*frederic.gosselin@inrae.fr*

# Différence entre **Bayésien** et **fréquentiste**

## Fréquentiste

- \* Les observations  $y$  suivent une loi de probabilité  $f(y|\theta, x)$  (incertitude aléatoire)
- \* Les paramètres  $\theta$  sont considérés comme fixes ; on ne cherche pas à décrire notre incertitude sur leurs valeurs (incertitude épistémique)

## Bayésien

- \* Les observations  $y$  suivent une loi de probabilité  $f(y|\theta, x)$  (incertitude aléatoire)
- \* On cherche à décrire notre incertitude sur les valeurs des paramètres  $\theta$  (incertitude épistémique)



# Différence entre Bayésien et fréquentiste

Une des grosses différences entre Bayésien et fréquentiste est donc qu'en Bayésien, on définit une distribution de probabilité  $p(\theta)$  sur les paramètres, avant d'observer les données  $x$  :

C'est la **distribution de probabilités a priori** (« prior distribution » ou « prior ») des paramètres.



# Différence entre **Bayésien** et **fréquentiste**

## Fréquentiste

- \* Les observations  $y$  suivent une loi de probabilité  $f(y|\theta, x)$  (incertitude aléatoire)
- \* Les paramètres  $\theta$  sont considérés comme fixes ; on ne cherche pas à décrire notre incertitude sur leurs valeurs (incertitude épistémique)
- \* L'incertitude fréquentiste fait intervenir des fréquences de résultats/décisions/tests à  $\theta$  constant mais jeux de données  $y$  répliqués suivant la loi  $f(y|\theta, x)$

## Bayésien

- \* Les observations  $y$  suivent une loi de probabilité  $f(y|\theta, x)$  (incertitude aléatoire)
- \* On cherche à décrire notre incertitude sur les valeurs des paramètres  $\theta$  (incertitude épistémique)
- \* On cherche à mettre à jour notre incertitude sur les valeurs des paramètres  $\theta$  en tenant compte des données collectées  $y$  et du modèle probabiliste  $f(y|\theta, x)$

# Différence entre **Bayésien** et **fréquentiste**

En Bayésien, on cherche à mettre à jour notre incertitude sur les paramètres, une fois les données *x observées* :

C'est la ***distribution de probabilités a posteriori***

(« *posterior distribution* » ou « *posterior* ») des paramètres :

$$p(\theta|y, x)$$

obtenue à partir du **prior** et du modèle probabiliste ou **vraisemblance** en appliquant le **théorème de Bayes**.



# MCMC: outil pour estimer asymptotiquement la distribution a posteriori

- MCMC = « Monte-Carlo Markov Chain »
- C'est une ou plusieurs (K) chaîne(s) de Markov (suite(s) de variables aléatoires)  $(X_{t,k})_{t \in N, 1 \leq k \leq K}$  prenant comme valeurs les paramètres statistiques du modèle  $\theta$  et construite de façon à converger vers la distribution postérieure des paramètres  $p(\theta|y, x)$

↳ Il faut donc prêter attention à ce que la convergence ait bien lieu vers la distribution asymptotique, pour chaque  $k$ , i.e. pour chacune des chaînes de Markov

# MCMC: rappel de la nature de la sortie

!!! Attention toutefois: ce qu'on obtient ce n'est **pas l'expression**

**mathématique de**  $p(\theta|y, x)$  ,

**mais  $N$  valeurs dans  $K$  trajectoires**  $(\theta_{j,k})_{1 \leq j \leq N, 1 \leq k \leq K} \sim p(\theta|y, x)$  .

# MCMC: les objectifs

On souhaite:

$N$  valeurs dans  $K$  trajectoires  $\left( \theta_{j,k} \right)_{1 \leq j \leq N, 1 \leq k \leq K} \underset{i}{\sim} p(\theta|y, x)$  ,  
qu'on peut considérer comme tirées au sort de manière  
indépendante dans la loi postérieure



# MCMC: objectif 1: convergence (burn-in)

- *Il faut donc prêter attention à ce que la convergence ait bien lieu*
  - ↳ *on ne va conserver les valeurs de la chaîne de Markov qu'après un certain nombre d'itérations  $v_{burn}$  de la chaîne, correspondant à la phase supposée transitoire (ou phase de « burn-in »)*

$$(X_{t,k})_{t \geq v_{burn}, 1 \leq k \leq K}$$

# MCMC: objectif 2: faible autocorrélation (thin)

- Par ailleurs, on ne va pas garder toutes les valeurs de la chaîne de Markov après la phase transitoire mais les valeurs avec *une certaine* « période » (dans les itérations), notée *Thin*, de façon à obtenir des valeurs des paramètres peu autocorrélées

⇒ on ne va conserver que

$$(X_{t,k})_{t \geq v_{burn} \text{ et } t \% \% Thin == 0, 1 \leq k \leq K}$$



**Notation R: signifie:**  
**t est un multiple de *Thin***

# MCMC: analyse des résultats

Pour avoir confiance en ses résultats, il faut commencer par **vérifier** plusieurs aspects du modèle :

(i) Vérification de la convergence : Traceplot + gelman.diag (ou autre)

(ii) *Conditionnellement à la convergence*, calcul du nombre de valeurs efficaces : comparaison SE + neff + acfplot (ou autocorrplot)

Une fois ces deux aspects évalués on peut s'intéresser aux résultats du modèle :

(iii) Résultats : summary + density plot



# Procédure habituelle n°1\*

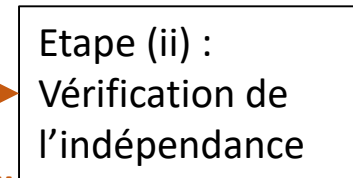
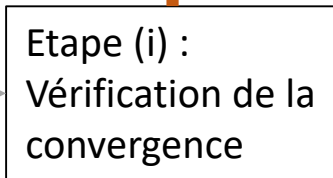
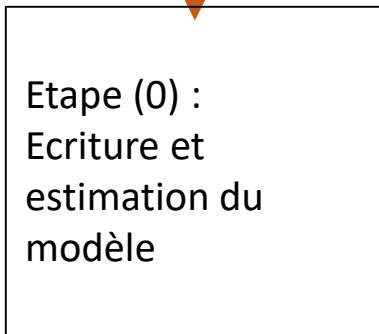
## Etape 1 : Vérification de la convergence

\*: avec Winbugs, openbugs, Stan, runMCMC dans Nimble

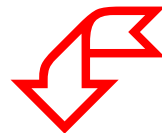


Convergence  
insatisfaisante

Reparamétrisation ou  
modification du nu.burn



Eventuelle  
modification avec  
window(..., start=...)  
et baisse N



INRAE

runMCMC\_btadjust

15 décembre 2022 / Applibugs / Supélec / F. Gosselin



# Procédure habituelle n°2\*

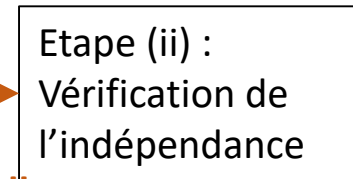
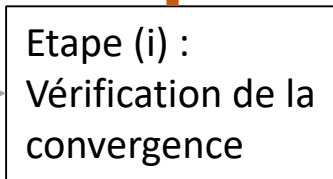
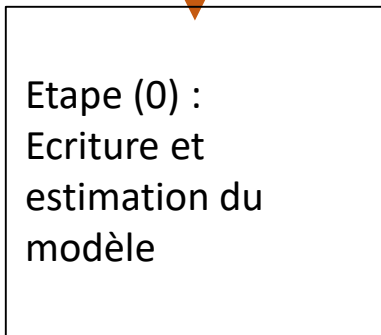
## Etape 1 : Vérification de la convergence

\*: avec Jags, reste Nimble, Greta...



Convergence  
insatisfaisante

Reparamétrisation



Eventuelle  
modification avec  
`window(..., start=...)` et  
augmentation niter



INRAE

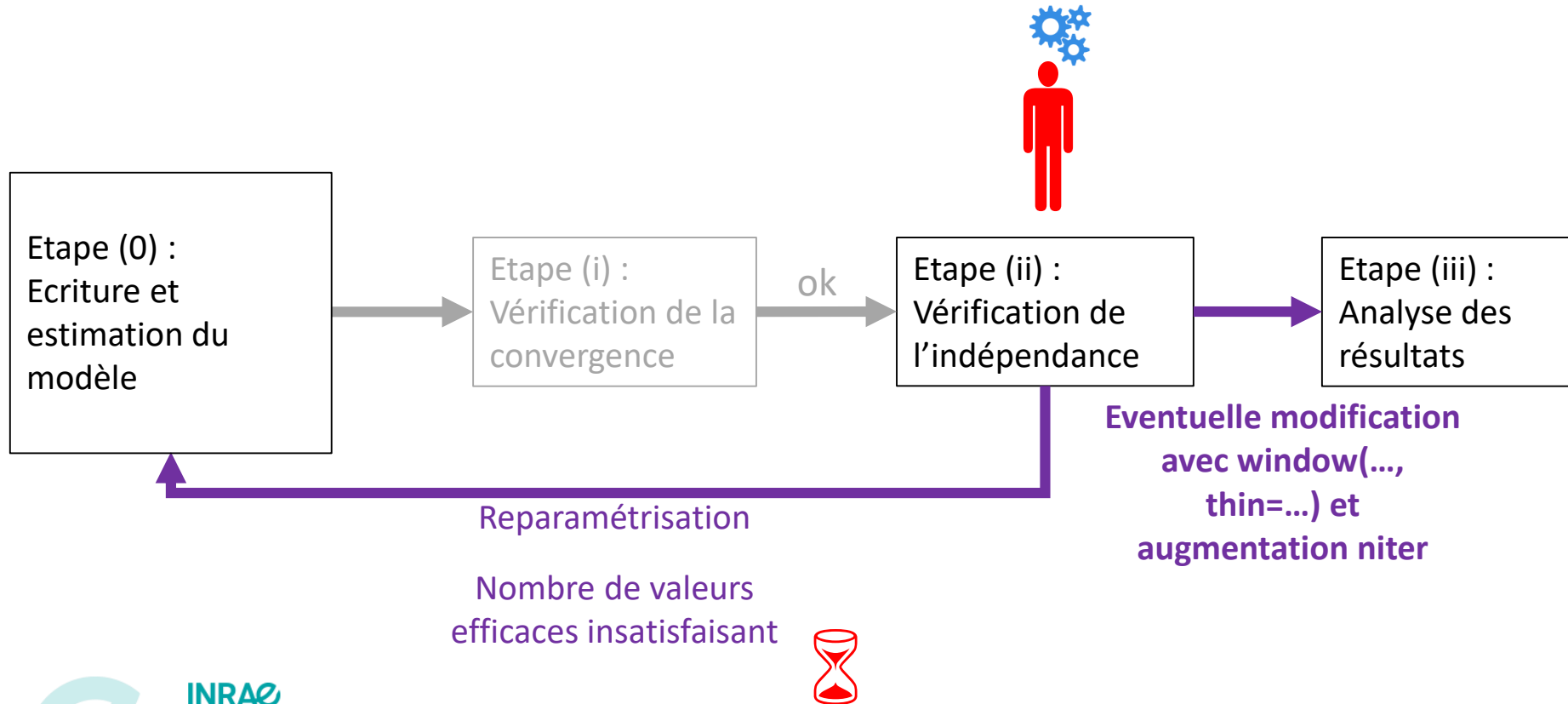
runMCMC\_btadjust

15 décembre 2022 / Applibugs / Supélec / F. Gosselin

# Procédure habituelle n°2\*

## Etape 2 : Vérification de l'indépendance

\*: avec Jags, reste Nimble, Greta...



# Inconvénients de la procédure habituelle n°1&2

- Nécessite de multiples (re-)lancements/analyses du modèle:

↳ **1: temps de calcul long (RSE)**



↳ **temps de travail important (RSE)**

↳ **1: possibilité de perte en termes de nombre de valeurs efficaces par rapport à ce qui était prévu**



- Particulièrement **problématique quand on a besoin d'itérer** le modèle N fois (e.g. validation croisée)



# Souhait d'une procédure davantage automatisée

- **Incitations à automatiser la procédure dans mes projets de recherche:**

- GNB (Gestion, Naturalité & Biodiversité) (2014-...) : écriture d'une procédure un peu automatique de ce type pour faire de la **comparaison de modèles** (avec Winbugs)
- PASSIFOR2 – M2 Pierre Bouchet (2020) : écriture d'une procédure de ce type pour la **partie convergence uniquement** (avec Jags)
- GNB-IBP (effet observateur) (2022): évolution de la **comparaison de modèles vers de la validation croisée** qui multiplierait le travail par N (ici 14) au moins (avec Nimble)
- GAMBAS (2022): travail sur la **convergence** des Joint Species Distributions Models (JSDMs)



- Souhait: **une seule ligne de commande** qui permette de lancer le modèle, de diagnostiquer/attendre la convergence, et d'obtenir en sortie un échantillon avec un nombre de valeurs efficaces souhaité ...
- ... et ce pour des modèles Bayésiens écrits avec **Jags, Nimble ou Greta** (tous les langages que je connaisse qui permettent de continuer le MCMC).



# Développement de runMCMCbtadjust

## Etape 1: pseudo-code convergence

```
Do first MCMC run with nburnin & niter
  converged<-check.convergence(nburnin,niter)
  i<-1
  while (!converged & ...)
    {converged<-
  check.convergence(nburnin+probs.conv[i]*(niter-nburnin),
  niter.min)
    i<-i+1}
Update thin, neffs.reached, nburnin & niter

While (!converged|!neffs.reached)
  {Do extra MCMC run with  $\Delta$ niter
  if (!converged|recheck.convergence)
  {converged<-check.convergence(nburnin,niter)
  while (!converged & ...)
    {converged<-
  check.convergence(nburnin+probs.conv[i]*(niter- nburnin),
  niter.min)
    i<-i+1}
  Update thin, neffs.reached, nburnin & niter}
```

# Développement de runMCMCbtadjust

## Etape 2 : pseudo-code « indépendance »

```
Do first MCMC run with nburnin & niter
  ///convergence section///
  if (converged)
    {Calculate thinmult
    thin<-min(thin*thinmult,thin.max) }
  Update neffs.reached, nburnin & niter

While (!converged|!neffs.reached)
  {Do extra MCMC run with  $\Delta$ niter
  ///convergence section///
  if (converged)
    {Calculate thinmult
    thin<-min(thin*thinmult,thin.max) }
  Update neffs.reached, nburnin & niter}
```



# Fonction runMCMC\_btadjust

## Principaux paramètres (I)

**MCMC\_language**: designates the `MCMC_language` used to write & fit the Bayesian model in R. Either "Nimble" - the default-, "Greta" or "Jags".

**code**: code for the model if `MCMC_language` is "Nimble" or "Jags".

**data, constants**: list for data if `MCMC_language` is "Nimble" or "Jags".

**model**: R model object if `MCMC_language` is "Greta".

*green*: arguments specific to `runMCMC_btadjust`

*blue*: arguments that are anyway needed for MCMC



# Fonction runMCMC\_btadjust

## Principaux paramètres (II)

**Nchains**: number of Markov chains.

**params**, **params.conv**, **params.save**: parameter names, monitored parameter names, saved parameter names.

**inits**: lists of initial values – one per chain.

**nburnin.min**, **nburnin.max**, **niter.min**, **niter.max**, **thin.min**, **thin.max**: in terms of iterations of Markov chains.

**neff.min**, **neff.med**, **neff.mean**: pre-specified targets in terms of minimum, median, mean effective sample size.

**conv.max**, **conv.med**, **conv.mean**: pre-specified targets in terms of maximum, median, mean convergence diagnostic.

***green**: arguments specific to runMCMC\_btadjust*

***blue**: arguments that are anyway needed for MCMC*



# Fonction runMCMC\_btadjust

## Paramètres de contrôle

```
control = list(time.max = NULL, innerprint = FALSE,  
check.convergence = TRUE, check.convergence.firstrun =  
NULL, recheck.convergence = TRUE, convtype = "Gelman",  
convtype.Gelman = 2, convtype.Geweke = c(0.1, 0.5),  
neff.method = "Stan", Ncycles.target = 2, probs.conv =  
c(0.25, 0.5, 0.75), min.Nvalues = 300, min.thinmult = 1.1,  
safemultiplier.Nvals = 1.2, round.thinmult = FALSE,  
identifiant.to.print = "", print.diagnostics = FALSE,  
print.thinmult = TRUE, seed = 1)
```

```
control.MCMC = list(confModel.expression.toadd = NULL,  
sampler = expression(hmc), warmup = 1000, n.adapt = 1000,  
n_cores = NULL)
```

***green**: arguments specific to `runMCMC_btadjust`*

***blue**: arguments that are anyway needed for MCMC*



# Fonction runMCMC\_btadjust

## Valeur

A **mcmc.list** object giving the values for `params.save` from iterations `nburnin` to `niter` with `thin`,

with **attributes** that include:

- **call.params** : reminder of call parameters
- **final.params**: value of some of the parameters at the en of the MCMC: `burnin`, `thin`, `niter.tot`, different kinds of durations (MCMC & not MCMC, transient & asymptotic...)
- **final.diags**: both synthetic & extended final diagnostics (convergence & ESS)
- **warnings**: warning messages

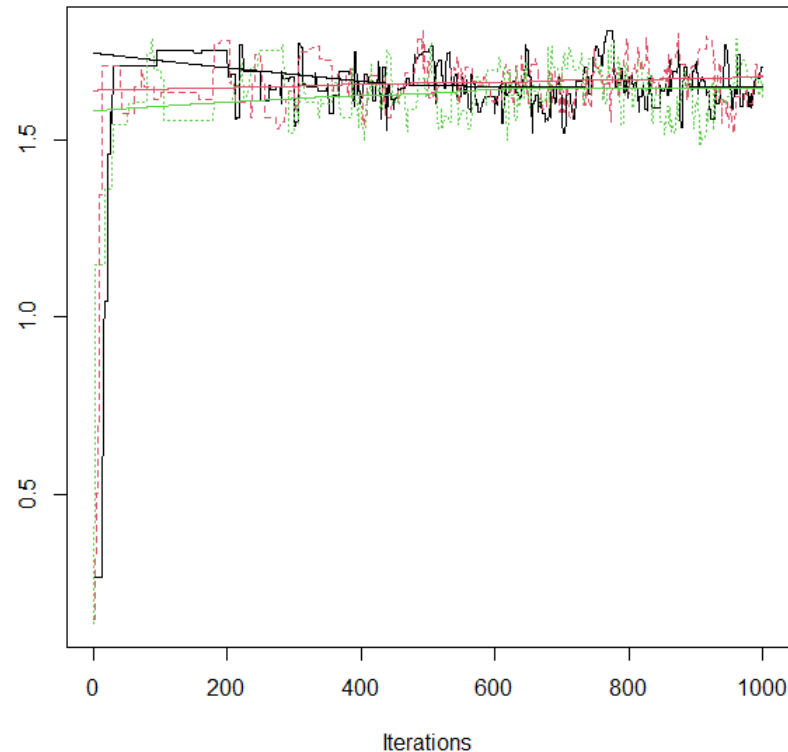




## ➤ Exemple: Gosselin et al. (2017) Forest Ecology & Management

Richesse spécifique des bryophytes terricoles en fonction de l'essence dominante en interaction avec le caractère mélangé

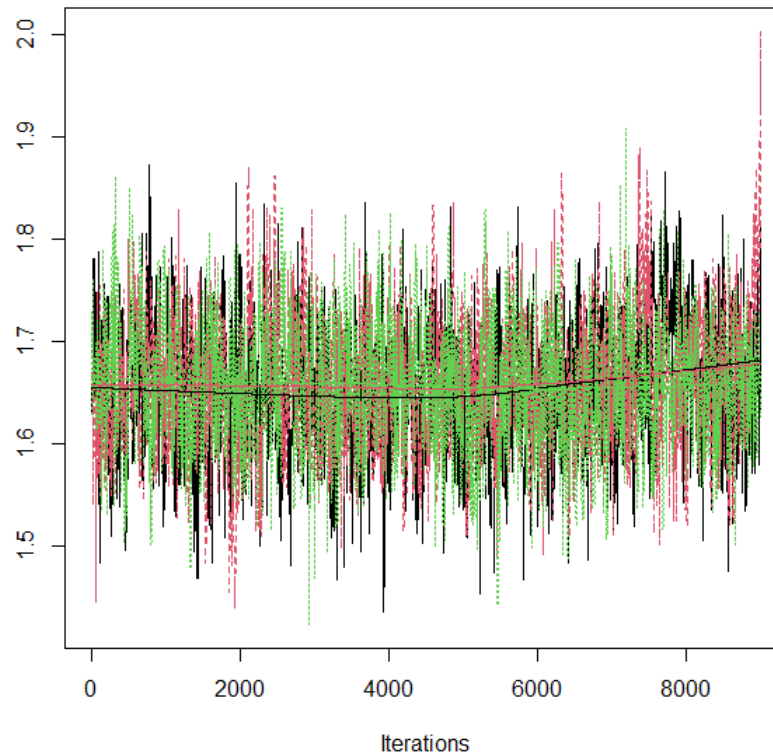
Modèle avec burn-in=1, thin=1, 1<sup>er</sup> coefficient



## ➤ Exemple: Gosselin et al. (2017) Forest Ecology & Management

Richesse spécifique des bryophytes terricoles en fonction de l'essence dominante en interaction avec le caractère mélangé

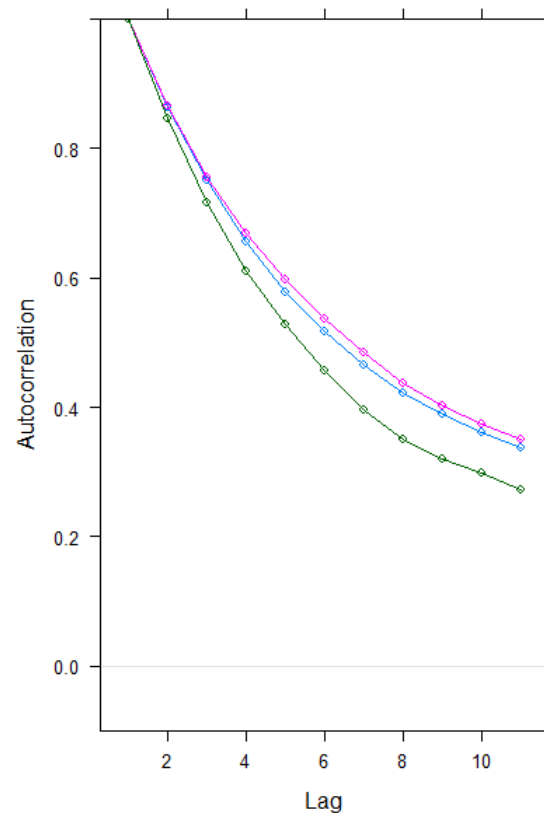
2<sup>nd</sup> Modèle avec burn-in=500, thin=9, 1<sup>er</sup> coefficient



## ➤ Exemple: Gosselin et al. (2017) Forest Ecology & Management

Richesse spécifique des bryophytes terricoles en fonction de l'essence dominante en interaction avec le caractère mélangé

2<sup>nd</sup> Modèle avec burn-in=500, thin=9, sans ajustement, 1<sup>er</sup> coefficient



## ➤ Exemple: Gosselin et al. (2017) Forest Ecology & Management

Richesse spécifique des bryophytes terricoles en fonction de l'essence dominante en interaction avec le caractère mélangé

3<sup>ème</sup> Modèle avec ajustement de burnin et thin: commande d'appel:

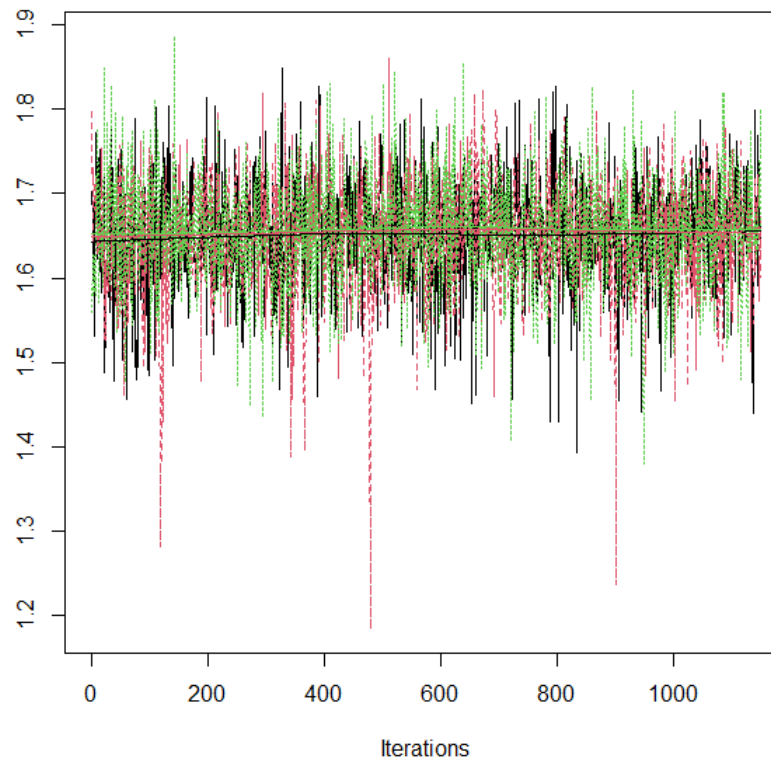
```
out.BPNB.RS.H5.Iscar.wbtadujsted<-  
runMCMC_btadjust(MCMC_language="Nimble", code=modelCode,  
constants = ModelConsts, data = ModelData,  
  
params.save=params, params.conv=params, Nchains=Nchains,  
inits = Inits, niter.min=10000, niter.max=3*10^6,  
nburnin.min=10, nburnin.max=10^6, thin.min=1, thin.max=1200,  
conv.max=1.05,neff.min=1000,neff.med=2500)
```



## ➤ Exemple: Gosselin et al. (2017) Forest Ecology & Management

Richesse spécifique des bryophytes terricoles en fonction de l'essence dominante en interaction avec le caractère mélangé

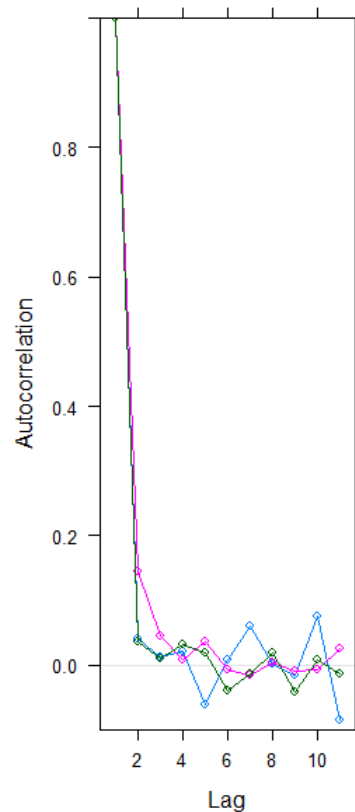
3<sup>ème</sup> Modèle avec ajustement de burnin et thin, 1<sup>er</sup> coefficient



## ➤ Exemple: Gosselin et al. (2017) Forest Ecology & Management

Richesse spécifique des bryophytes terricoles en fonction de l'essence dominante en interaction avec le caractère mélangé

3<sup>ème</sup> Modèle avec ajustement de burnin et thin, 1<sup>er</sup> coefficient



## > Discussion

- Assez proche de la fonction `autorun.jags` du package `runjags` (développement parallèle)
- **Avantages `autorun.jags`:**
  - Davantage d'options pour `jags` (modules...)
- **Avantages `runMCMC_btadjust`:**
  - Accès à trois langages, pas juste `jags`
  - Ajustement du paramètre `thin`
  - Spécification de la méthode et du critère de convergence
  - Spécification de la méthode et du critère d'ESS (indépendance)
  - Autres options



## ➤ Discussion

- Connaissez-vous d'autres fonctions sous R qu'[autorun.jags](#) qui adaptent burn-in et/ou thin?
- Connaissez-vous [d'autres langages MCMC sous R](#) que Jags, Nimble & Greta qui permettent de produire des échantillons supplémentaires?
- Faut-il étendre ce type de code à des [fonctions ad-hoc](#), non génériques (ex: code « maison » MCMC)?
- Quelles [autres fonctionnalités](#) pas trop compliquées et utiles à embarquer?





## ➤ Principaux avantages de runMCMCbtadjust

- **Code simplifié (surtout Nimble)**
- **Rigueur**: « force » l'utilisateur à avoir un MCMC ayant convergé avec un niveau de convergence pré-spécifié et avec un nombre de valeurs efficaces pré-spécifié
- **RSE: Réduction de la dépense énergétique**: pas besoin de relancer N fois le même modèle pour atteindre la convergence ou avoir un nombre de valeurs efficaces spécifié
- **RPS: Réduction du temps de travail répétitif**: c'est la fonction qui s'occupe de régler la convergence et l'indépendance
- **Comparaison d'efficacité Jags/Nimble/Greta simplifiée**: accès à différents types de durées, à ESS & convergence contrôlées



## ➤ Principaux avantages de runMCMCbtadjust

- **Particulièrement utile pour des estimations répétées:**
  - ↳ répétition d'analyses statistiques de données simulées
  - ↳ comparaison de modèles multiples (WAIC, looIC, **CV**)

## > Perspectives

- publication sous CRAN (fin 2022-début 2023)
- autres publications (orales, écrite)
- formation: utilisation en cours M2 (Orléans), diffusion/formation auprès des collègues
- retours/suggestions d'améliorations bienvenus



## ➤ runMCMCbtadjust in a nutshell

*runMCMCbtadjust vous libère des tâches fastidieuses de la vérification de convergence et du choix de la période et devrait permettre d'être plus efficace sur un plan énergétique.*



## > runMCMCbtadjust in a nutshell

*runMCMCbtadjust frees the modeler in you*

(pour paraphraser Kéry 2010)



*Le temps retrouvé*

(en mémoire de M. Proust 1871-1922)



## ➤ Des questions ?



*Remerciements: F. Mortier (ANR Gambas), P. Bouchet, M. Gosselin, U. Godeau (INRAE)*



**INRAE**

runMCMC\_btadjust

15 décembre 2022 / Applibugs / Supélec / F. Gosselin