runMCMCbtadjust: Un package R pour faciliter l'utilisation de " Monte-Carlo Markov Chains " (MCMC) en statistique Bayésienne

Frédéric Gosselin

runMCMCbtadjust
Un 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

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

INRAE

Différence entre Bayésien et fréquentiste

Fréquentiste

* Les observations y suivent une loi de probabilité $f(y|\theta, x)$ (incertitude aléatoire)

* Les paramètres θ 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 θ (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 θ 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 θ 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 θ (incertitude épistémique)

*** On cherche à mettre à jour notre incertitude sur les valeurs des paramètres θ 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) $\left(X_{\iota,k}\right)_{\iota \in N, 1 \leq k \leq K}$ prenant comme valeurs les paramètres statistiques du modèle θ 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: nature de la sortie

Ce qu'on obtient ce n'est pas l'expression mathématique de

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

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

MCMC: les objectifs

On souhaite:

N valeurs dans K trajectoires $\left(\theta_{j,k}\right)_{1\le j\le N, 1\le k\le K} \underset{\text{i}}{\sim} p(\theta|y,x)$,

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* ν_{burn} *de la chaîne, correspondant à la phase supposée transitoire (ou phase de « burn-in »)*

$$\left(X_{\iota,k}\right)_{\iota \geq \nu_{burn}, 1 \leq k \leq K}$$

MCMC: objectif 2: faible autocorrélation (thin)

- *Par ailleurs, on ne va pas garder toutes les valeurs de la chaîne de Markov après la phrase 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*

$$\left(X_{\iota,k}\right)_{\iota \geq \nu_{burn} \text{ et } \iota \%\% Thin == 0, 1 \leq k \leq K}$$

Notation R: signifie:
ι 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

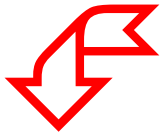
Etape (0) : Ecriture et estimation du modèle

Etape (i) : Vérification de la convergence

Etape (ii) : Vérification de l'indépendance

Etape (iii) : Analyse des résultats

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



Procédure habituelle n°1*

Etape 2 : Vérification de l'indépendance

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

Etape (0) : Ecriture et estimation du modèle

Etape (i) : Vérification de la convergence

ok

Etape (ii) : Vérification de l'indépendance

Etape (iii) : Analyse des résultats

Reparamétrisation ou modification du Thin

Eventuelle modification avec window(…, thin=…) et baisse *N*

Nombre de valeurs efficaces insatisfaisant

Procédure habituelle n°2*

Etape 1 : Vérification de la convergence

*: avec Jags, reste Nimble, Greta…

Convergence insatisfaisante

Reparamétrisation

Etape (0) : Ecriture et estimation du modèle

Etape (i) : Vérification de la convergence

Etape (ii) : Vérification de l'indépendance

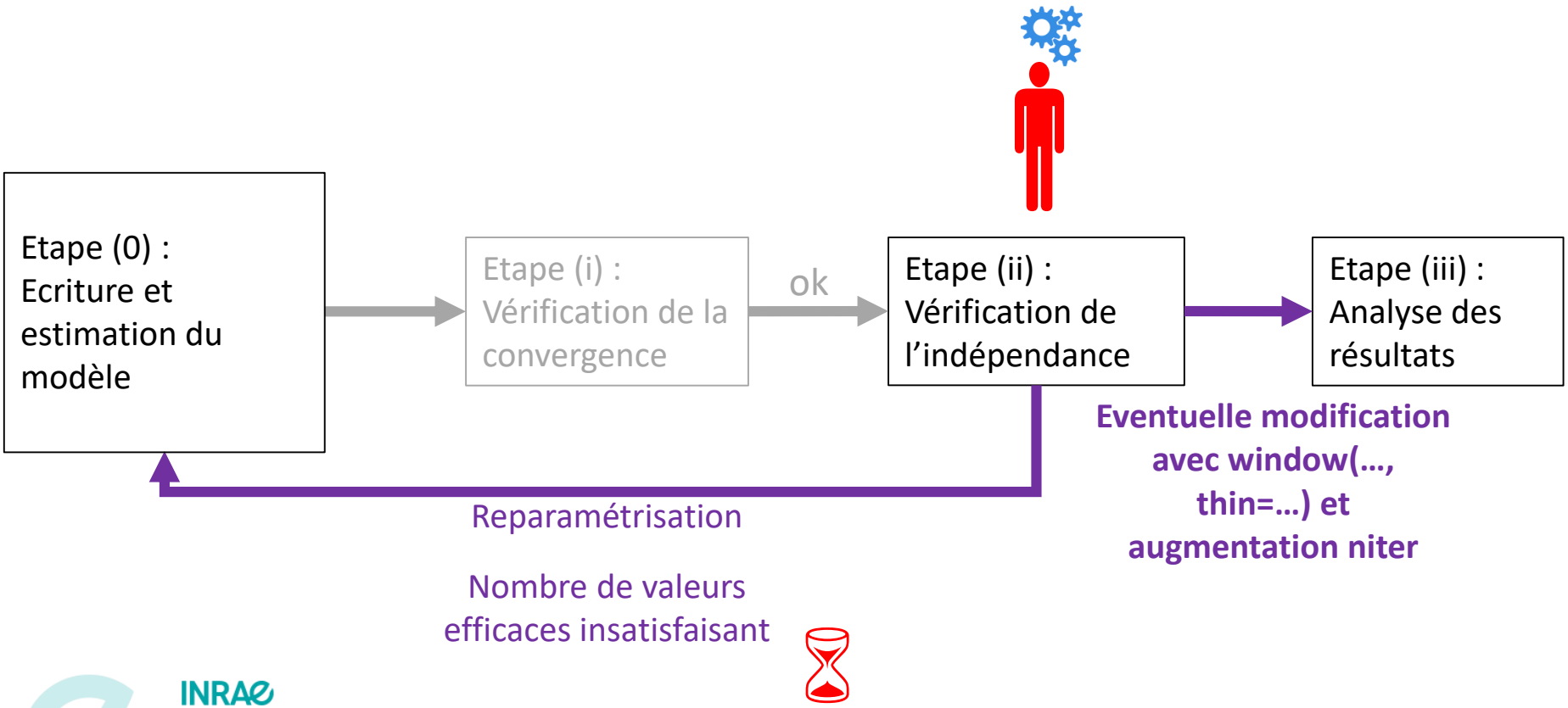
Etape (iii) : Analyse des résultats

Eventuelle modification avec window(…, start=…) et/ou augmentation niter

Procédure habituelle n°2*

Etape 2 : Vérification de l'indépendance

*: avec Jags, reste Nimble, Greta…

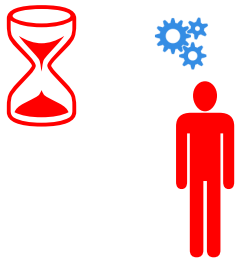


Inconvénients de la procédure habituelle n°1&2

- 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)

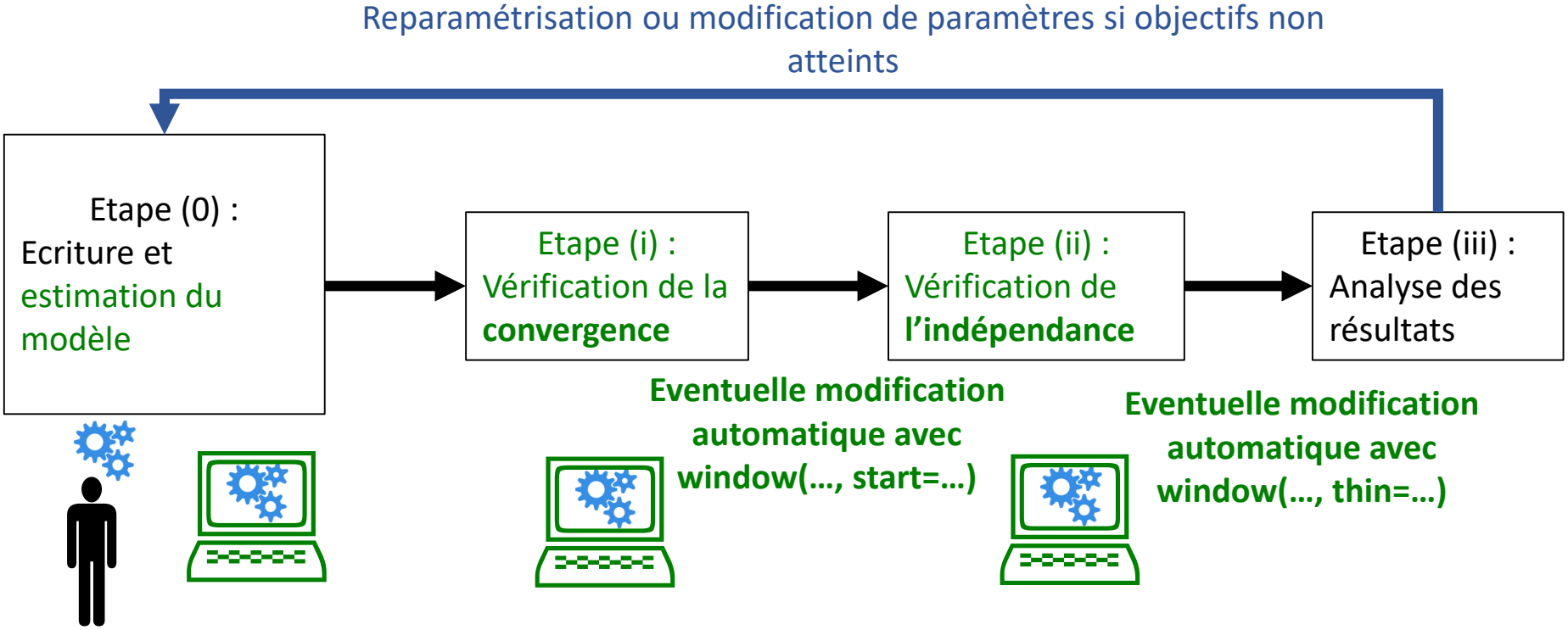
Souhait d'une procédure davantage automatisée

- **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 d'un nouveau package R

Etapes 1 & 2 faites par le package runMCMCbtadjust



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 Δniter
        if (!converged|recheck.convergence)
        {converged<-check.convergence(nburnin,niter)
        while (!converged & …)
            {converged<-
        check.convergence(nburnin+probs.conv[i]*(nite
    r-  nburnin),  niter.min)
            i<-i+1}
        Update thin, neffs.reached, nburnin & niter}
```

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 Δniter
        if (!converged|recheck.convergence)
        {converged<-check.convergence(nburnin,niter)
        while (!converged & …)
            {converged<-
        check.convergence(nburnin+probs.conv[i]*(nite
    r-  nburnin),  niter.min)
            i<-i+1}
        Update thin, neffs.reached, nburnin & niter}
```

Vérification de la convergence à intervalles réguliers entre nburnin et niter

Développement de runMCMCbtadjust

Etape 2 : pseudo-code « indépendance »

```
Do first MCMC run with nburnin & niter
    ///convergence section///
    if (converged)
    {Calculate thinmult (>1)
    thin<-min(thin*thinmult,thin.max)}
    Update neffs.reached, nburnin & niter
    Keep past samples only every thin interations

While (!converged|!neffs.reached)
    {Do extra MCMC run with Δniter
        ///convergence section///
        if (converged)
            {Calculate thinmult (>1)
            thin<-min(thin*thinmult,thin.max)}
        Update neffs.reached, nburnin & niter
        Keep past samples only every thin interations }
```

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 de la partie btadjust

```
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 = TRUE,
identifier.to.print = "", print.diagnostics = FALSE,
print.thinmult = TRUE, innerprint = FALSE, seed = 1,
remove.fixedchains = TRUE, check.installation = TRUE)
```

green: *arguments specific to runMCMC_btadjust*
blue*: arguments that are anyway needed for MCMC*

Fonction runMCMC_btadjust

Paramètres de contrôle du MCMC

```
control.MCMC = list(confModel.expression.toadd = NULL,
sampler = expression(hmc), warmup = 1000, n.adapt = 1000,
n_cores = NULL, RNG.names = c("base::Wichmann-Hill",
"base::Marsaglia-Multicarry", "base::Super-Duper",
"base::Mersenne-Twister"), n_cores = NULL,
showCompilerOutput = TRUE)
```

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` every `thin` iteration,

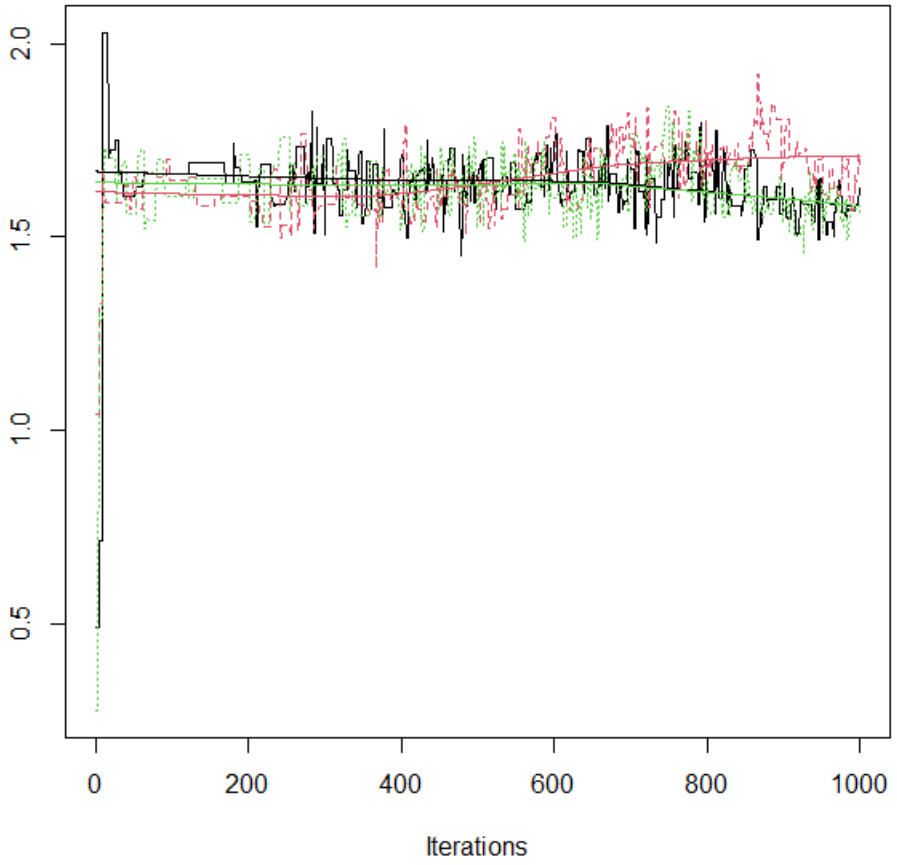
with **`attributes`** that include (among other things):

- **`call.params`** : reminder of call parameters
- **`final.params`**: value of some of the parameters at the end 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)
- **`package.versions & R.version`** : storing package & R versions used when fiting MCMC
- **`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

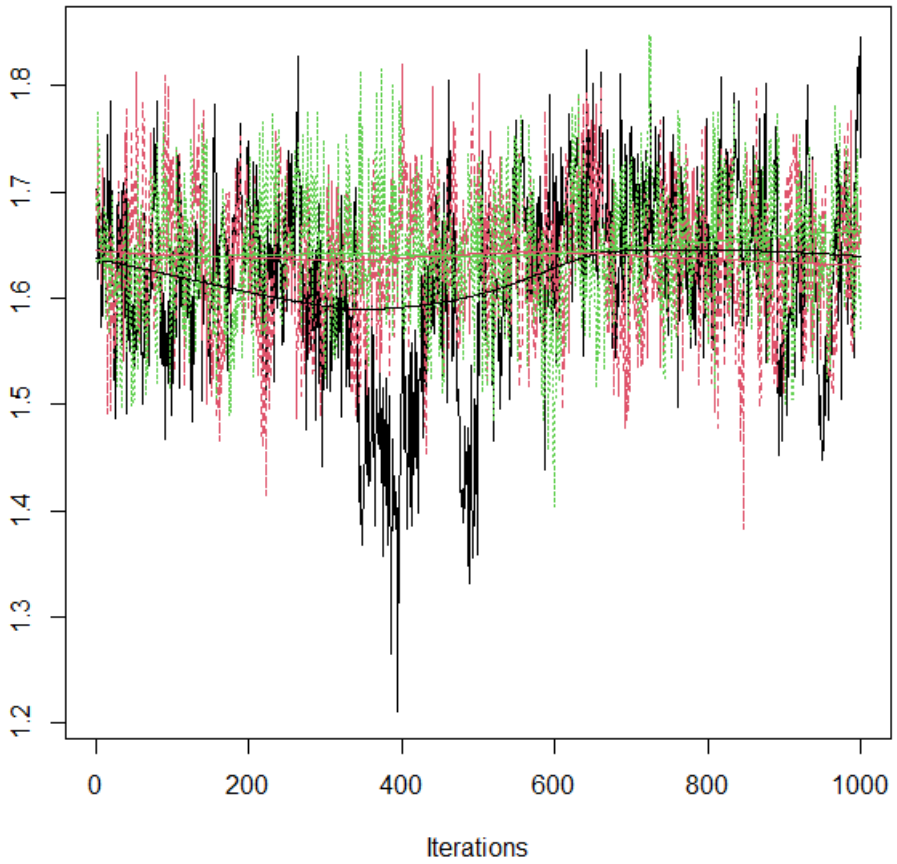
Modèle avec burn-in=1. thin=1. 1er coefficient



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

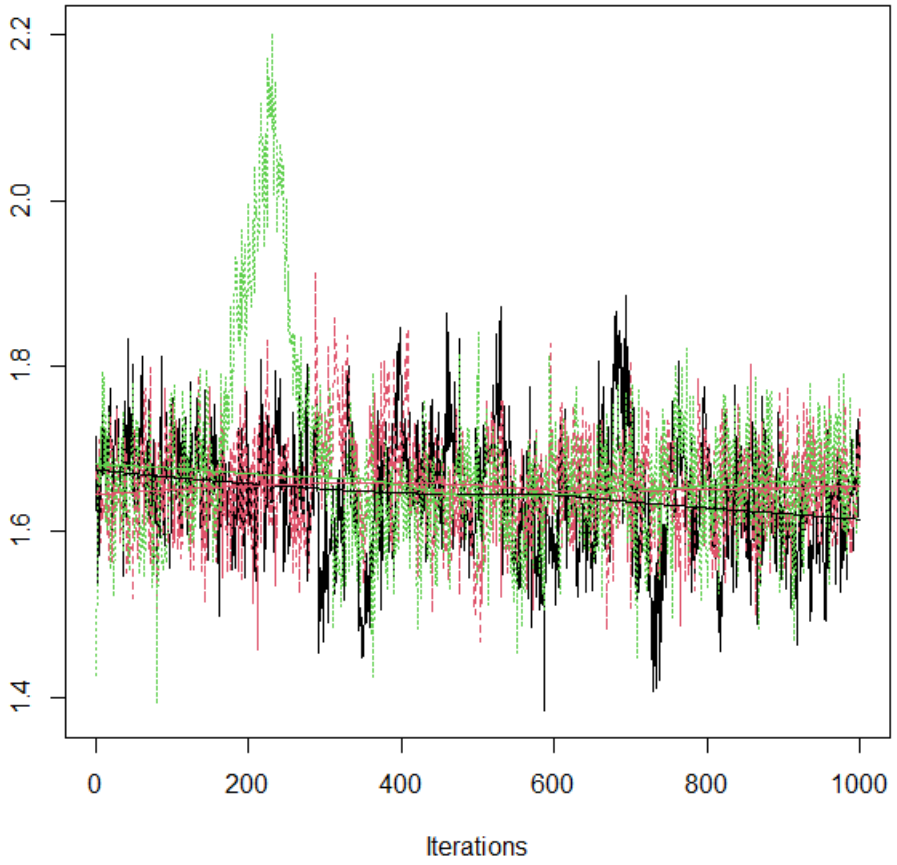
2nd Modèle avec burn-in=500, thin=9, 1er coefficient



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

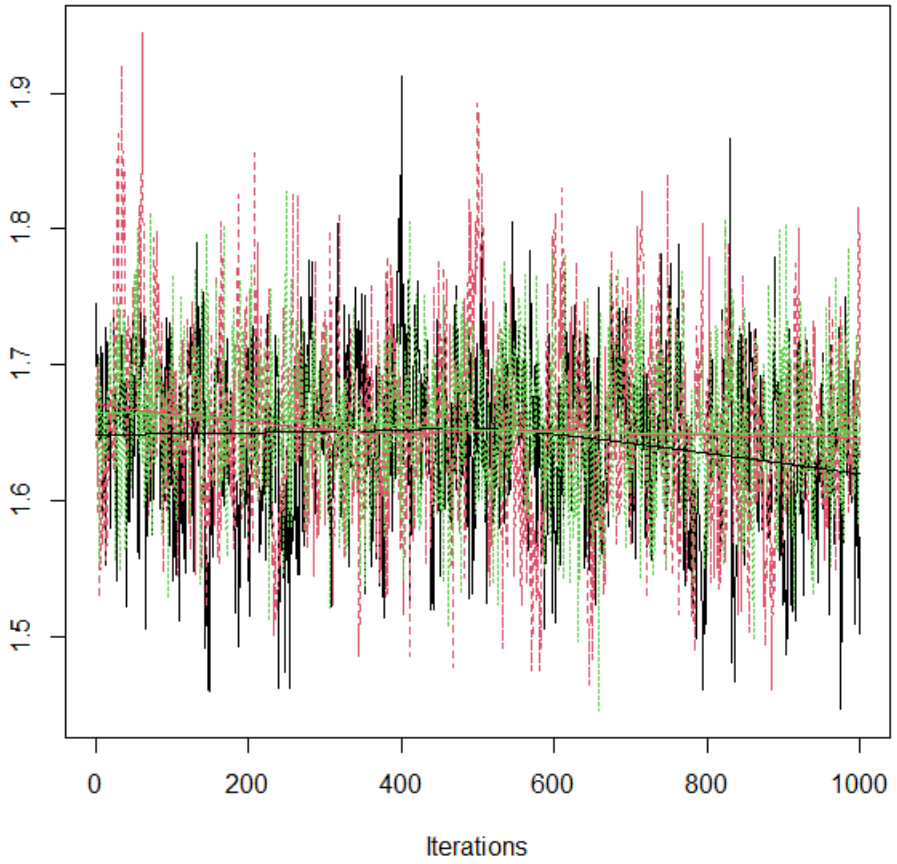
3ème Modèle avec burn-in=500+600*9, thin=9, 1er coefficient



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

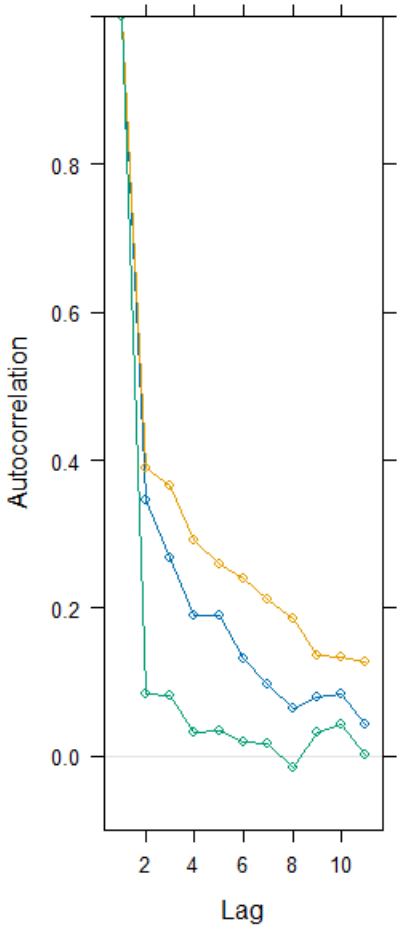
4ème Modèle avec burn-in=20000, thin=9, 1er coefficient



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

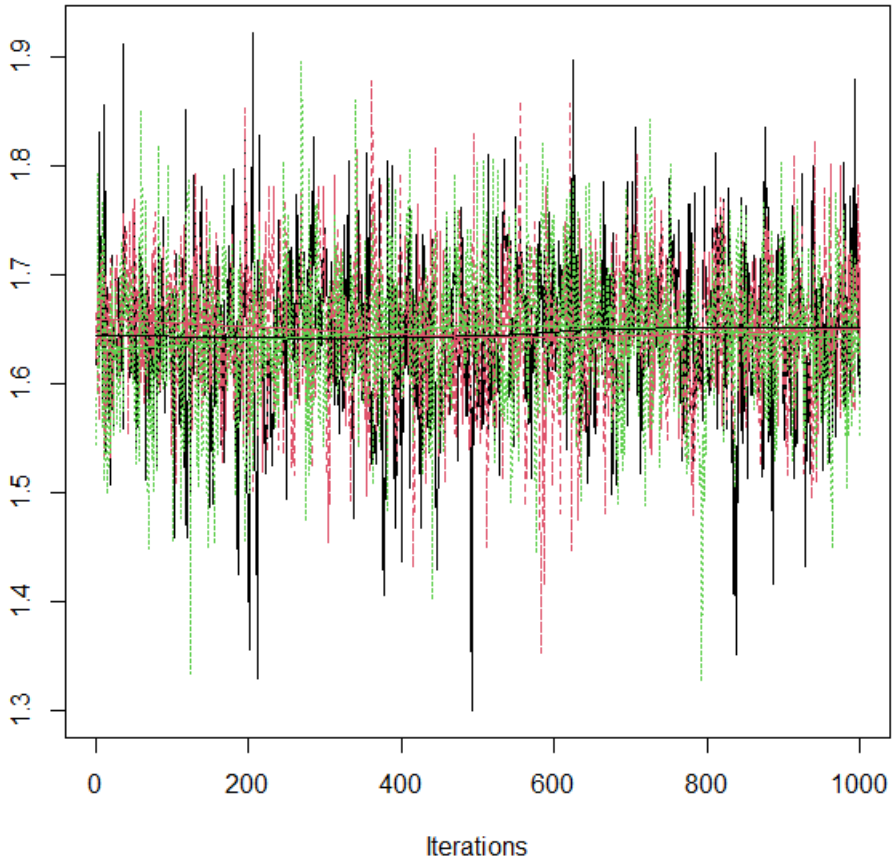
4ème Modèle avec burn-in=20000, thin=9, 1er coefficient



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

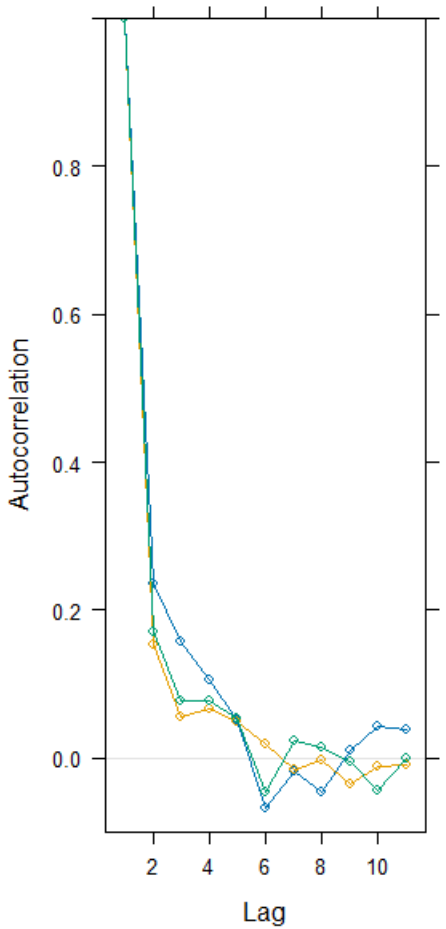
5ème Modèle avec burn-in=20000, thin=100, 1er coefficient



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

5ème Modèle avec burn-in=20000, thin=100, 1er 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

Lignes de commandes pour un de ces modèles:

```
K <-Nchains<- 3

N.sst <- 3000

Thin <- 100

nu.burn <- 20000

N.ssc <-round(N.sst/K)

Niter <- nu.tot <- nu.burn+N.ssc*Thin

Nburn<-nu.burn

Model <- nimbleModel(code = modelCode, name = 'Nimble', constants =
ModelConsts,  data = ModelData,check=FALSE,inits=Inits[[1]])

CModel <- compileNimble(Model)

ConfModel <- configureMCMC(Model, thin = Thin, monitors = params)

ModelMCMC <- buildMCMC(ConfModel)

CModelMCMC <- compileNimble(ModelMCMC, project = CModel, showCompilerOutput =
TRUE)

set.seed(2)

out.BPNB.RS.H5.Iscar.wb20000t100 <- runMCMC(CModelMCMC, niter=Niter,
nburnin=Nburn, nchains=Nchains, inits = Inits, progressBar = TRUE
,setSeed=TRUE, samplesAsCodaMCMC=TRUE)
```

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

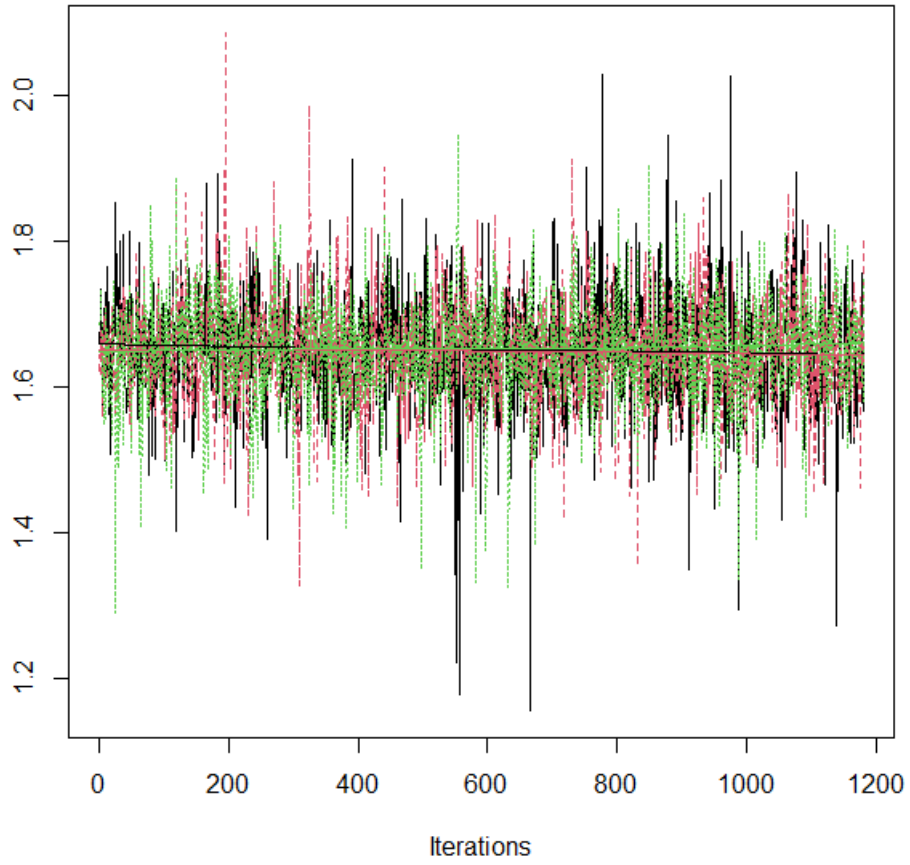
Modèle avec ajustement automatique de burnin et thin: commande d'appel:

```
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

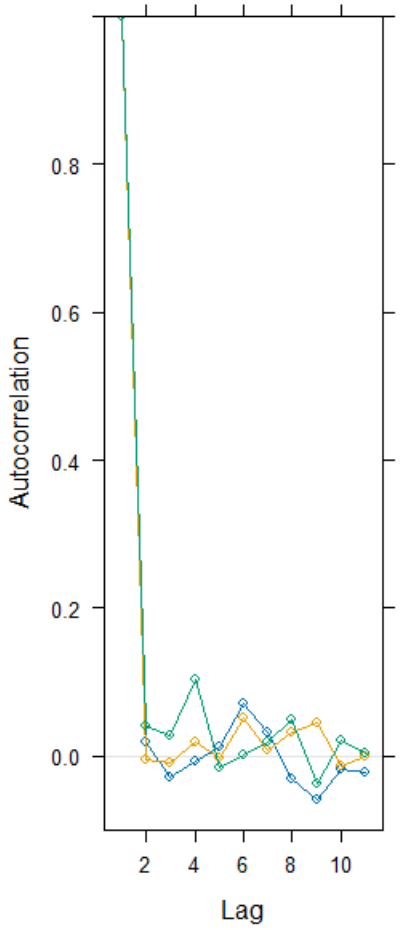
Modèle avec ajustement automatique de burnin et thin, 1er coefficient



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

Modèle avec ajustement automatique de burnin et thin, 1er 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 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
- **Capacité à changer de langage pour le même modèle facilitée** (presque les mêmes appels à runMCMC_btadjust)

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

- publications (orales, écrite)
- formation: utilisation en cours M2 (Orléans), diffusion/formation auprès des collègues
- quelques autres fonctionnalités programmées pour les versions suivantes
- retours/suggestions d'améliorations bienvenus

runbMCMCbtadjust 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.

runbMCMCbtadjust 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)