Introduction à la modélisation statistique bayésienne

Un cours en R et Stan avec brms

Ladislas Nalborczyk (CNRS, LPL, Aix-Marseille Univ)

Planning

Cours n°01 : Introduction à l'inférence bayésienne
Cours n°02 : Modèle Beta-Binomial
Cours n°03 : Introduction à brms, modèle de régression linéaire
Cours n°04 : Modèle de régression linéaire (suite)
Cours n°05 : Markov Chain Monte Carlo
Cours n°06 : Modèle linéaire généralisé
Cours n°07 : Comparaison de modèles
Cours n°08 : Modèles multi-niveaux (généralisés)
Cours n°09 : Examen final

Rappels de notation

La notation $p(y \mid \theta)$ peut faire référence à deux choses selon le contexte : la fonction de vraisemblance et le modèle d'observation. De plus, on trouve de nombreuses notations ambigues en statistique. Essayons de clarifier ci-dessous.

- $\Pr(Y = y \mid \Theta = \theta)$ désigne une **probabilité** (e.g., `dbinom(x = 2, size = 10, prob = 0.5)`).
- $p(Y = y \mid \Theta = \theta)$ désigne une **densité** de probabilité (e.g., `dbeta(x = 0.4, shape1 = 2, shape2 = 3)`).
- $p(Y = y \mid \Theta)$ désigne une fonction de vraisemblance (likelihood) discrète ou continue, y est connu/fixé, Θ est une variable aléatoire, la somme (ou l'intégrale) de cette distribution **n'est pas égale à 1** (e.g., `dbinom(x = 2, size = 10, prob = seq(0, 1, 0.1) )`).
- $p(Y \mid \Theta = \theta)$ désigne une fonction de masse (ou densité) de probabilité (dont la somme ou l'intégrale est égale à 1), qu'on appelle aussi "modèle d'observation" (observation model) ou "distribution d'échantillonnage" (sampling distribution), Y est une variable aléatoire, θ est connu/fixé (e.g., `dbinom(x = 0:10, size = 10, prob = 0.5)`)

Le but de l'analyse bayésienne (i.e., ce qu'on obtient à la fin d'une telle analyse) est la distribution postérieure $p(\theta \mid y)$. On peut la résumer pour faciliter la communication des résultats, mais toute l'information souhaitée est contenue dans **la distribution toute entière** (pas seulement sa moyenne, son mode, ou autre).

Rappels de notation

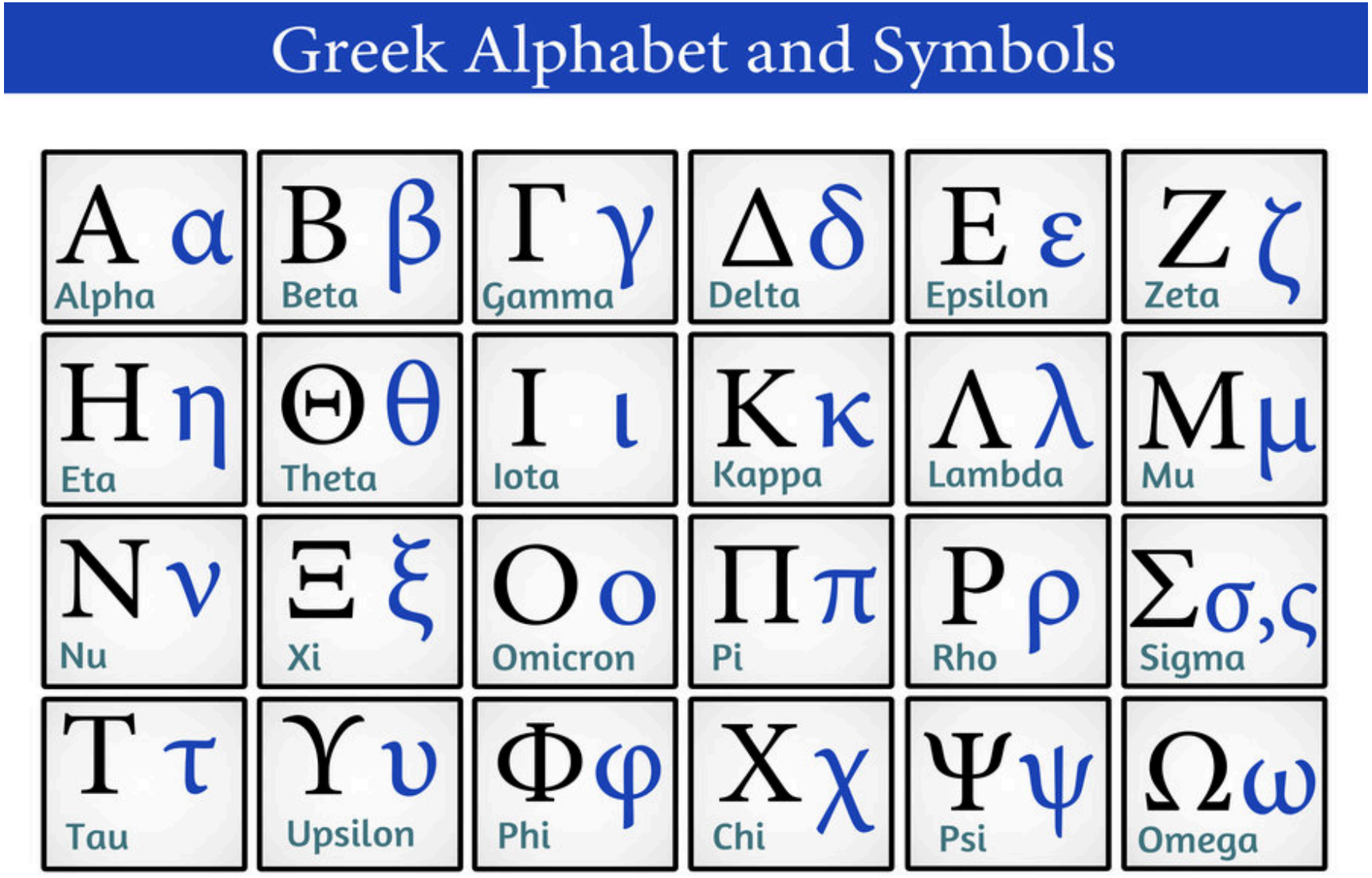
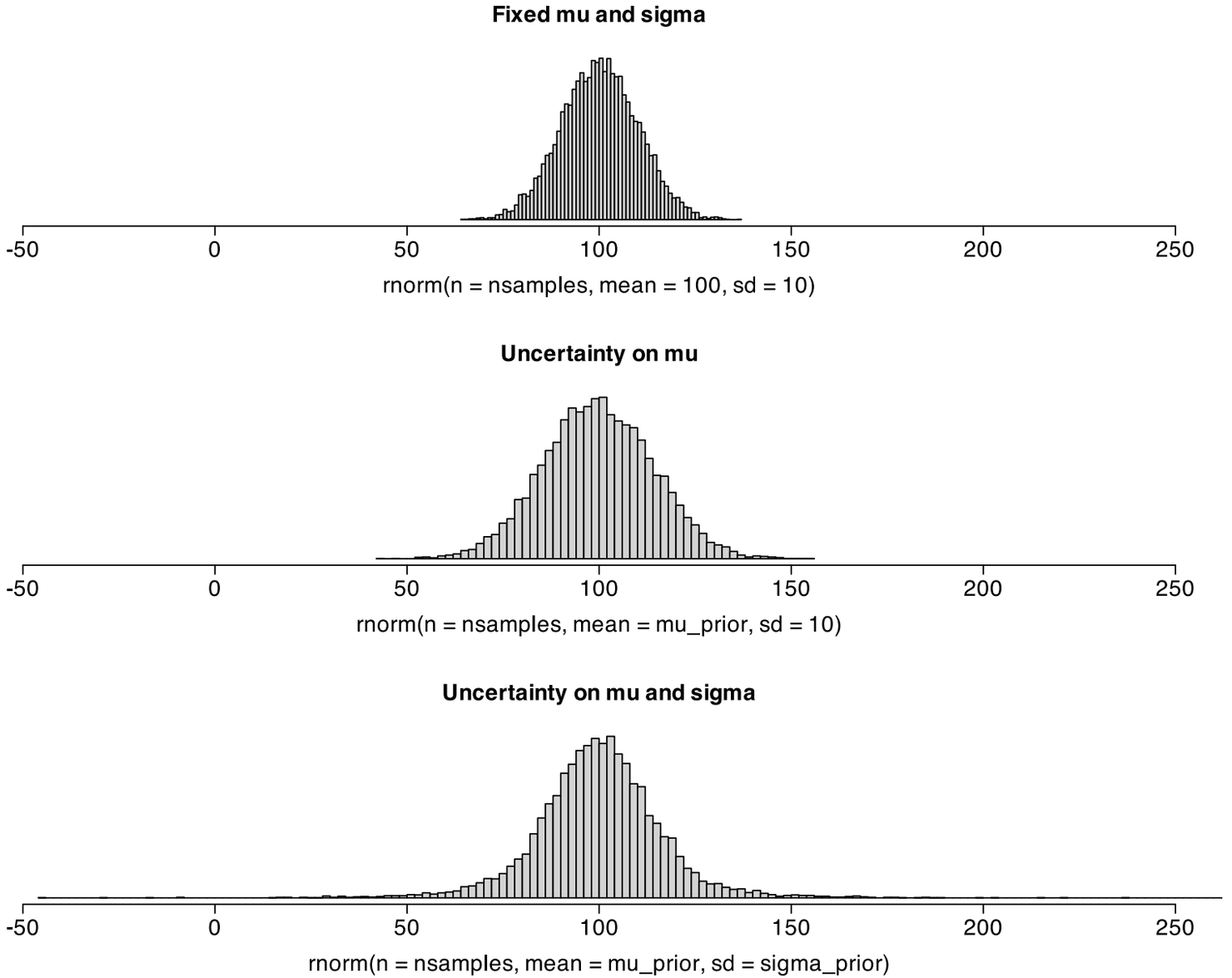


Illustration tirée de https://masterofmemory.com/mmem-0333-learn-the-greek-alphabet/.

Rappels : prior predictive checking

```r
##########################################################################
# On définit un modèle avec :                                            #
# Une fonction de vraisemblance Gaussienne : y ~ Normal(mu, sigma)       #
# Un prior Gaussien pour la moyenne : mu ~ Normal(100, 10)               #
# Et un prior Exponentiel pour l'écart-type : sigma ~ Exponential(0.1)   #
##########################################################################

# on simule 10.000 observations issues d'une distribution Gaussienne sans incertitude (épistémique)
rnorm(n = 1e4, mean = 100, sd = 10) |> hist(breaks = "FD")

# on tire 10.000 échantillons issus du prior Gaussien pour mu (i.e., p(mu))
mu_prior <- rnorm(n = 1e4, mean = 100, sd = 10)

# 10.000 observations issues d'une distribution Gaussienne avec incertitude sur mu
rnorm(n = 1e4, mean = mu_prior, sd = 10) |> hist(breaks = "FD")

# on tire 10.000 échantillons issus du prior Exponentiel pour sigma (i.e., p(sigma))
sigma_prior <- rexp(n = 1e4, rate = 0.1)

# 10.000 observations issues d'une distribution Gaussienne avec incertitude sur mu ET sigma
# ce que le modèle suppose à propos de y sur la base de nos priors pour mu et sigma...
```

Rappels : prior predictive checking



Le problème avec la distribution postérieure

$$p(\mu, \sigma \mid h) = \frac{\prod_i \text{Normal}(h_i \mid \mu, \sigma)\text{Normal}(\mu \mid 178, 20)\text{Uniform}(\sigma \mid 0, 50)}{\int\int \prod_i \text{Normal}(h_i \mid \mu, \sigma)\text{Normal}(\mu \mid 178, 20)\text{Uniform}(\sigma \mid 0, 50)\text{d}\mu\text{d}\sigma}$$

Petit problème : La constante de normalisation (en vert) s'obtient en calculant la somme (pour des variables discrètes) ou l'intégrale (pour des variables continues) de la densité conjointe $p(\text{data}, \theta)$ sur toutes les valeurs possibles de θ. Cela se complique lorsque le modèle comprend plusieurs paramètres et/ou que la forme de la distribution postérieure est complexe...

Le problème avec la distribution postérieure

WebGL is not supported by your browser -
visit https://get.webgl.org for more info

Rappels Cours n°02

Trois méthodes pour résoudre (contourner) ce problème :

- La distribution a priori est un **prior conjugué** de la fonction de vraisemblance (e.g., modèle Beta-Binomial). Dans ce cas, il existe une solution analytique (i.e., qu'on peut calculer de manière exacte) pour la distribution postérieure.
- Autrement, pour des modèles simples, on peut utiliser la méthode par grille. On calcule la valeur exacte de la probabilité postérieure en un nombre fini de points dans l'espace des paramètres.
- Pour les modèles plus complexes, explorer tout l'espace des paramètres n'est pas tractable. On va plutôt échantillonner **intelligemment** un grand nombre de points dans l'espace des paramètres.

Objectifs du cours

- ⟶ Présenter le principe de base de l'échantillonnage : Markov Chain Monte Carlo
- ⟶ Présenter deux algorithmes (Metropolis-Hastings et HMC)
- ⟶ Montrer les forces mais aussi les faiblesses de ces méthodes
- ⟶ Donner des outils de contrôle sur ces méthodes
- ⟶ Appliquer ces méthodes à un cas simple

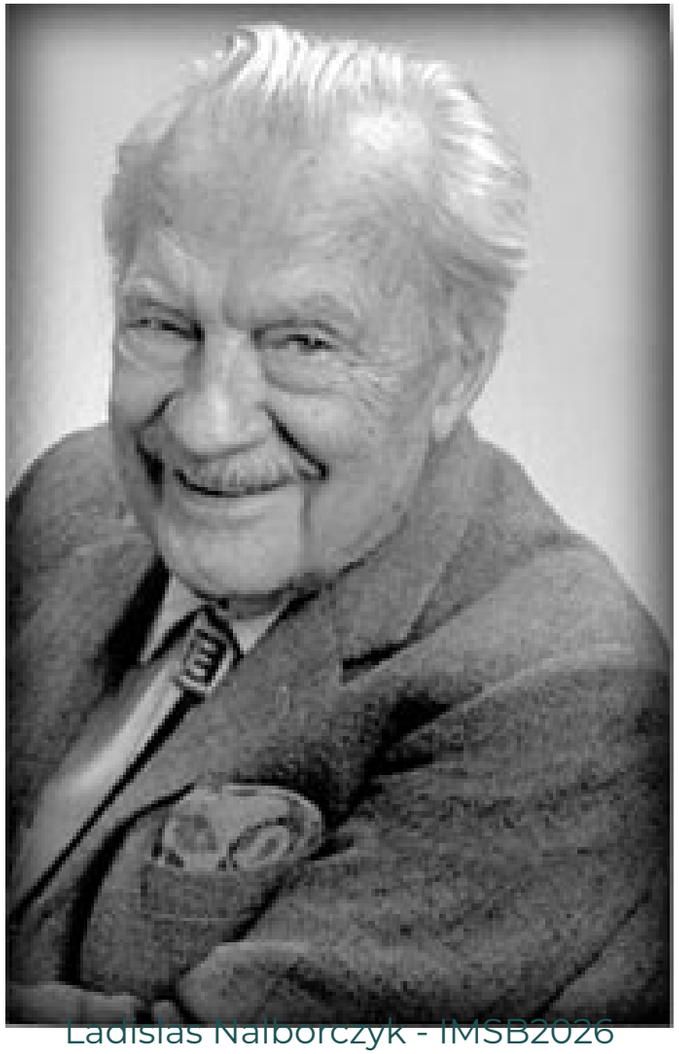
Markov Chain Monte Carlo

- Markov chain **Monte Carlo**
  - ⟶ Échantillonnage aléatoire
  - ⟶ Le résultat est un ensemble de valeurs du paramètre
- Markov **chain** Monte Carlo
  - ⟶ Les valeurs sont générées sous forme de séquences (liaison de dépendance)
  - ⟶ Indice temporel pour identifier la place dans la chaîne
  - ⟶ Le résultat est de la forme : $\theta^1, \theta^2, \theta^3, \ldots, \theta^t$
- **Markov** chain Monte Carlo
  - ⟶ La valeur de paramètre générée ne dépend que de la valeur du paramètre précédent

$$\Pr(\theta^{t+1} \mid \theta^t, \theta^{t-1}, \ldots, \theta^1) = \Pr(\theta^{t+1} \mid \theta^t)$$

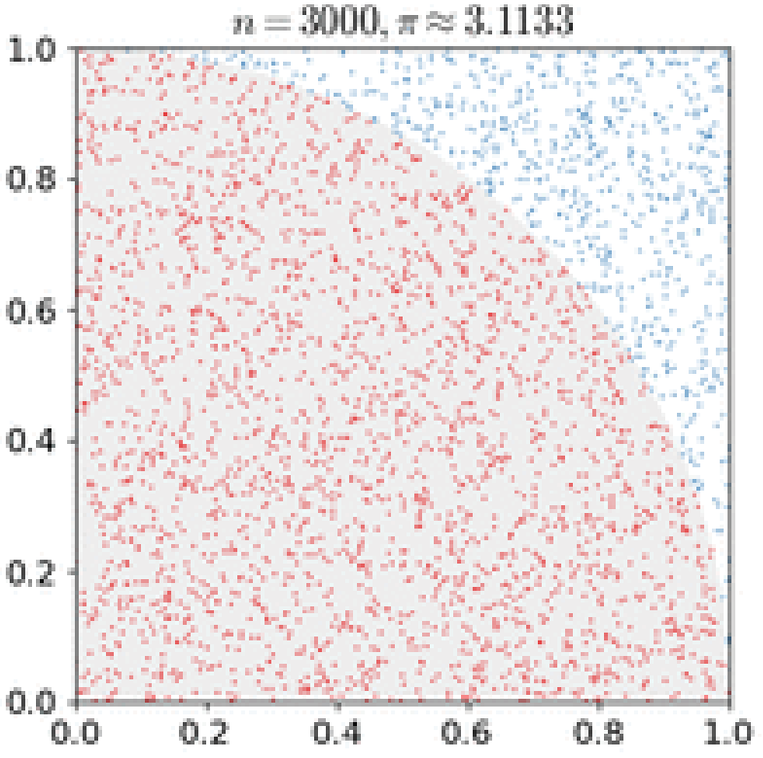
Méthodes Monte Carlo

Le terme de **méthode de Monte-Carlo** désigne une famille d'algorithmes visant à calculer (ou approcher) une valeur numérique en utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes. Cette méthode a été formalisée en 1947 par Nicholas Metropolis, et publiée pour la première fois en 1949 dans un article co-écrit avec Stanislaw Ulam.



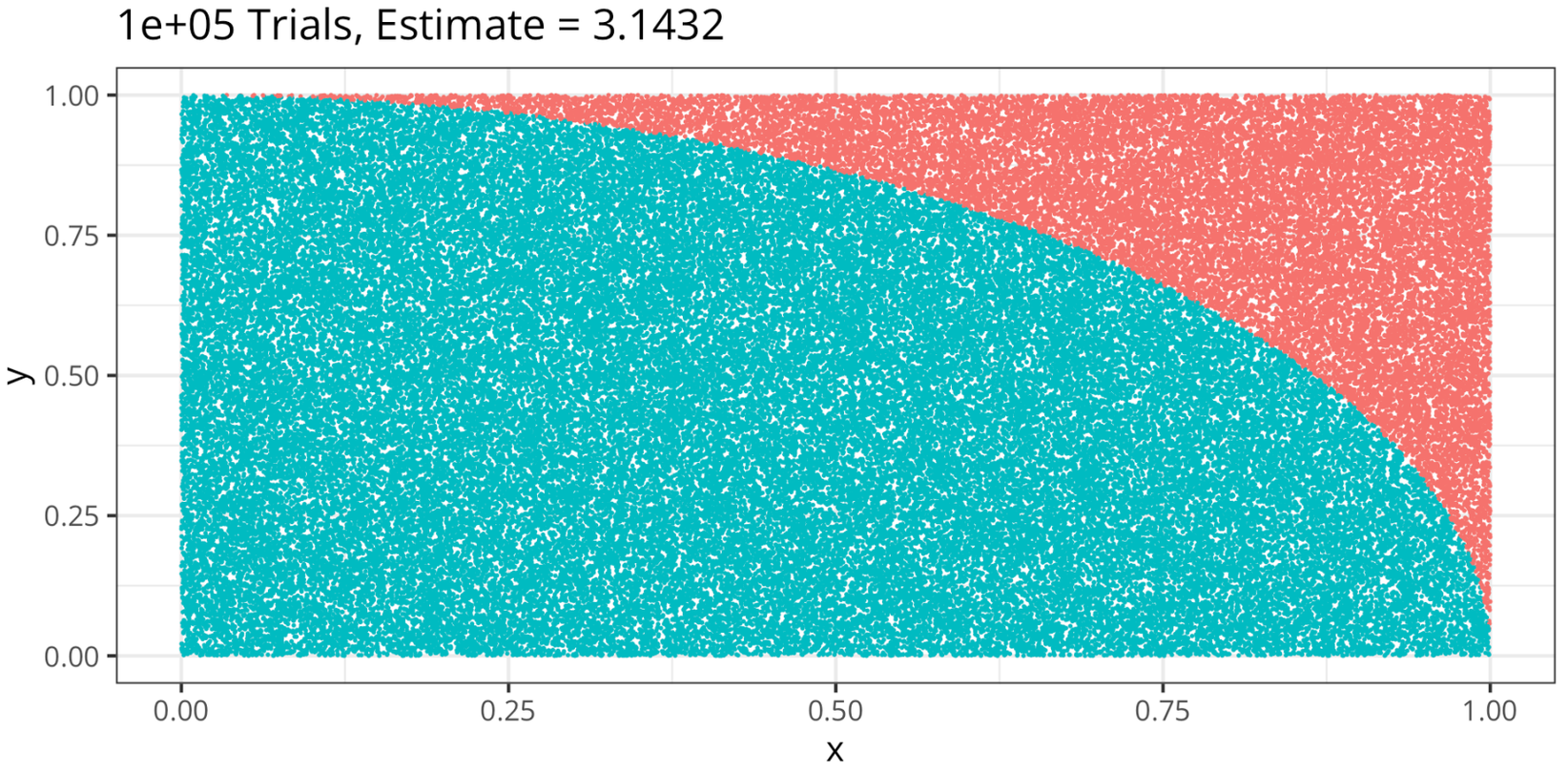
Méthodes Monte Carlo : Estimation de π

Soit un point M de coordonnées (x, y), où $0 < x < 1$ et $0 < y < 1$. On tire aléatoirement les valeurs de x et y entre 0 et 1 suivant une loi uniforme. Le point M appartient au disque de centre $(0, 0)$ de rayon $r = 1$ si et seulement si $\sqrt{x^2 + y^2} \leqslant 1$. On sait que le quart de disque est de surface $\sigma = \pi r^2/4 = \pi/4$ et que le carré qui le contient est de surface $s = r^2 = 1$. Si la loi de probabilité du tirage de point est uniforme, la probabilité que le point M appartienne au disque est donc de $\sigma/s = \pi/4$. En faisant le rapport du nombre de points dans le disque au nombre de tirages $\frac{N_{\text{inner}}}{N_{\text{total}}}$, on obtient alors une approximation de $\pi/4$.



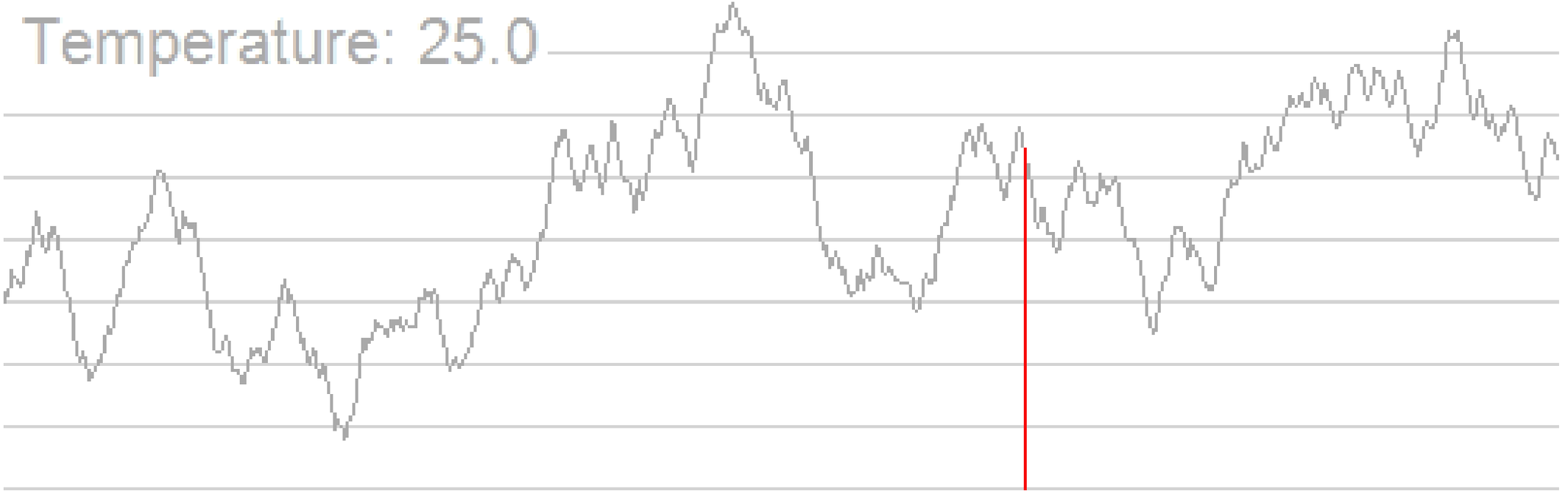
Méthodes Monte Carlo : Estimation de π

```r
trials <- 1e5 # nombre d'échantillons
radius <- 1 # rayon du cercle
x <- runif(n = trials, min = 0, max = radius) # tirages pour x
y <- runif(n = trials, min = 0, max = radius) # tirages pour y
distance <- sqrt(x^2 + y^2) # distance à l'origine
inside <- distance < radius # à l'intérieur (ou pas) du quart de cercle ?
pi_estimate <- 4 * sum(inside) / trials # estimation de pi
```



Méthodes Monte Carlo

Autre exemple : déterminer la superficie d'un lac ou encore déterminer le maximum d'une fonction (optimisation) via **recuit simulé** (simulated annealing, voir Wikipedia).



Méthodes Monte Carlo

Monte Carlo désigne une famille d'algorithmes qui ont pour but d'approcher des valeurs numériques à partir de procédés aléatoires. Pourrait-on s'en servir pour obtenir une approximation de la distribution postérieure ?

On connaît les priors $p(\theta_1)$ et $p(\theta_2)$
On connaît la fonction de vraisemblance $p(\text{data} \mid \theta_1, \theta_2)$

Mais on ne sait pas calculer la distribution postérieure... $p(\theta_1, \theta_2 \mid \text{data}) = \frac{p(\text{data} \mid \theta_1, \theta_2)p(\theta_1)p(\theta_2)}{p(\text{data})}$

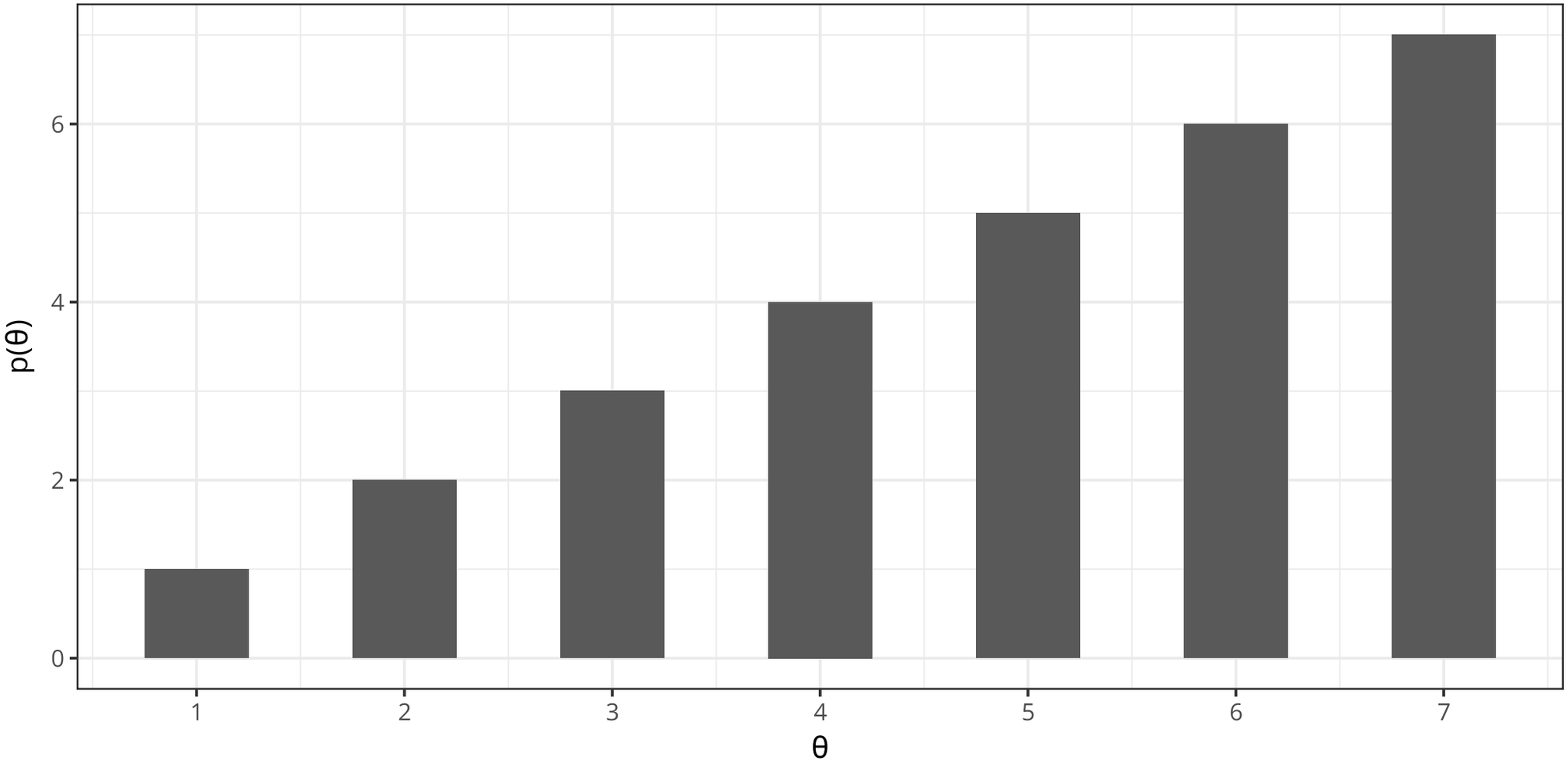
Ou plutôt, on ne sait pas calculer $p(\text{data})$... ! Mais on sait calculer la distribution postérieure à une constante près. Or, comme $p(\text{data})$ est une constante, elle ne change pas la forme de la distribution postérieure... ! On va donc explorer l'espace des paramètres et produire des échantillons proportionnellement à leur (densité de) probablité relative.

Influence de la constante de normalisation

WebGL is not supported by your browser -
visit https://get.webgl.org for more info

Méthodes Monte Carlo : Exemple

Considérons un exemple simple : Soit un paramètre θ avec 7 valeurs possibles et la fonction de répartition suivante, où $p(\theta) = \theta$.

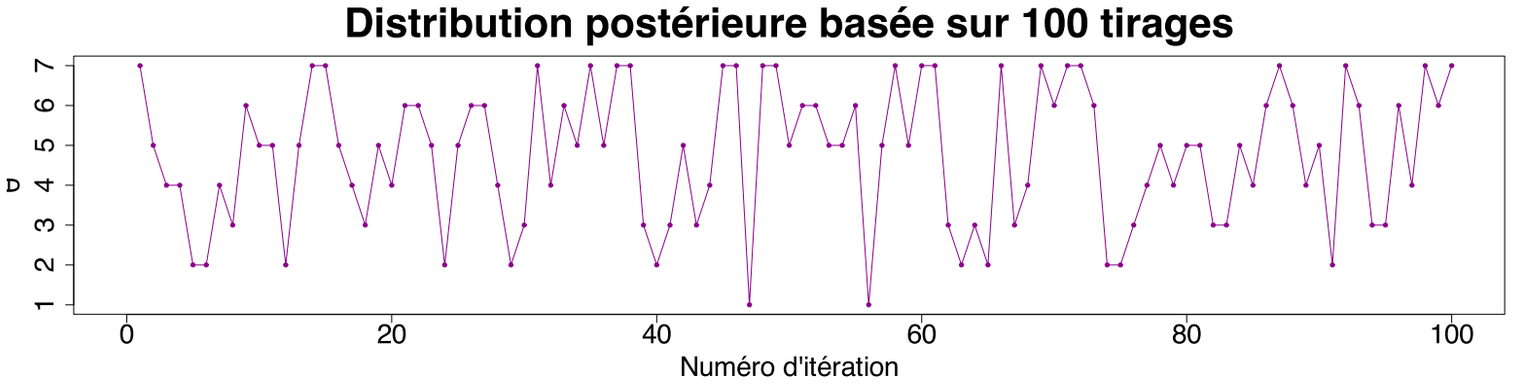


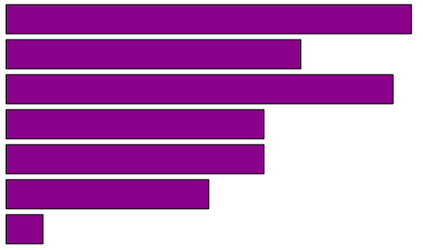
Méthodes Monte Carlo : Exemple

Approximation de cette distribution par tirage aléatoire : Cela revient à tirer aléatoirement un grand nombre de points “au hasard” parmi ces 28 cases (comme pour le calcul de π) !

Méthodes Monte Carlo : Exemple

```r
niter <- 100 # nombre d'itérations
theta <- 1:7 # valeurs possibles de theta
ptheta <- theta # densité de probabilité de theta
samples <- sample(x = theta, prob = ptheta, size = niter, replace = TRUE) # échantillons
```





- La distribution des échantillons obtenus converge vers la “vraie” distribution.
- Mais, cela nécessite généralement beaucoup d’échantillons…
- Aucun contrôle sur la vitesse de convergence…
- Et si on abandonnait l’échantillonnage indépendant ?

Algorithme Metropolis

Cet algorithme a été présenté pour la première fois en 1953 par Nicholas Metropolis, Arianna W. Rosenbluth, Marshall Rosenbluth, Augusta H. Teller, et Edward Teller. Le problème des algorithmes Monte-Carlo n'est pas la convergence, mais la vitesse à laquelle la méthode converge. Pour augmenter la vitesse de convergence, il faudrait **faciliter l'accès aux valeurs de paramètres les plus représentées**.

Principe :

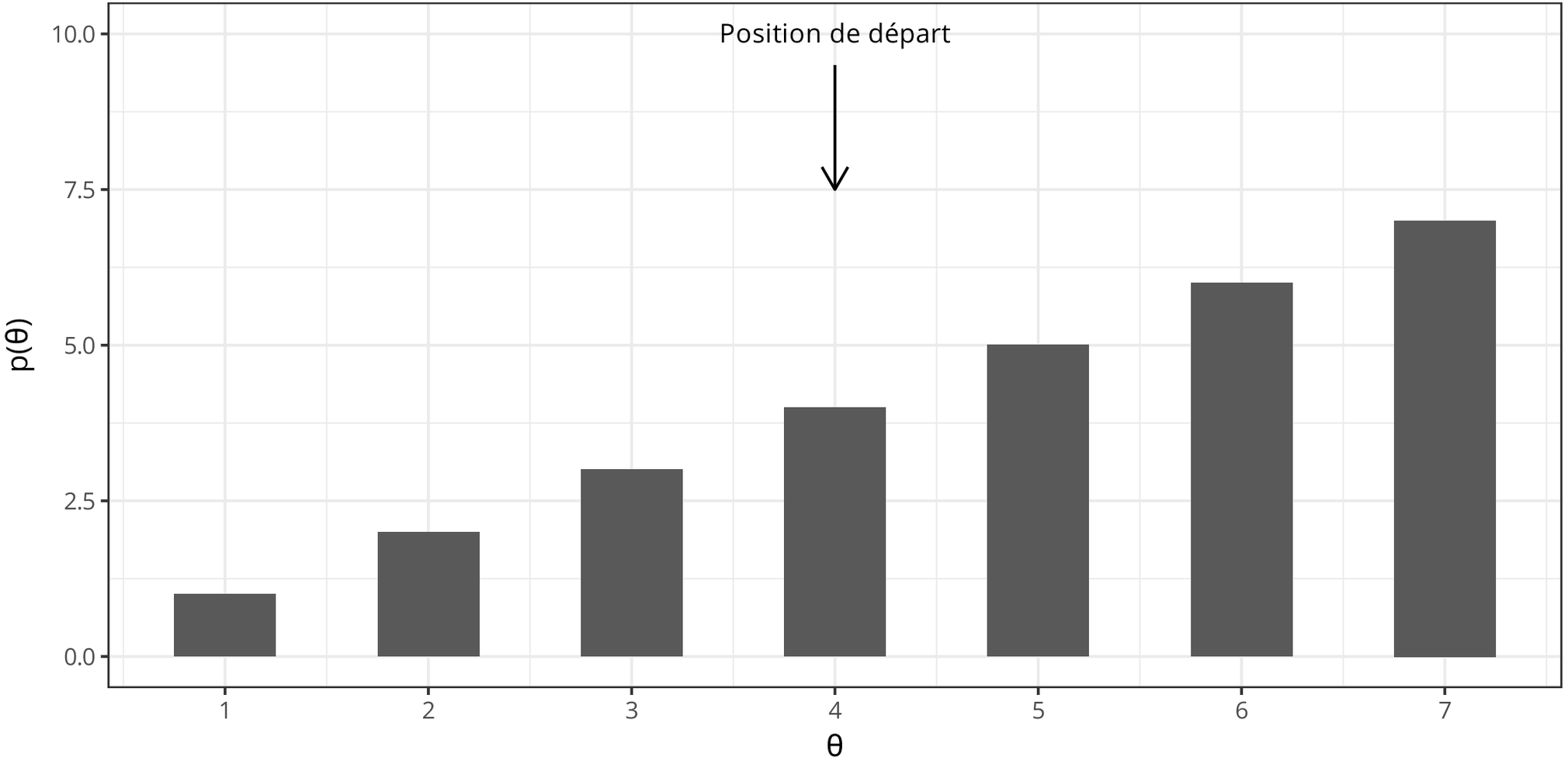
- On fait une proposition de déplacement sur la base de la valeur courante du paramètre.
- On réalise un tirage aléatoire pour accepter ou rejeter la nouvelle position.

Deux idées centrales :

- La proposition doit favoriser les valeurs de paramètre les plus probables : On parcourt plus souvent ces valeurs de paramètres.
- La proposition doit se limiter aux valeurs adjacentes au paramètre courant : On augmente la vitesse de convergence en restant là où se trouve l'information (i.e., en parcourant l'espace des paramètres de manière **locale** plutôt que **globale**).

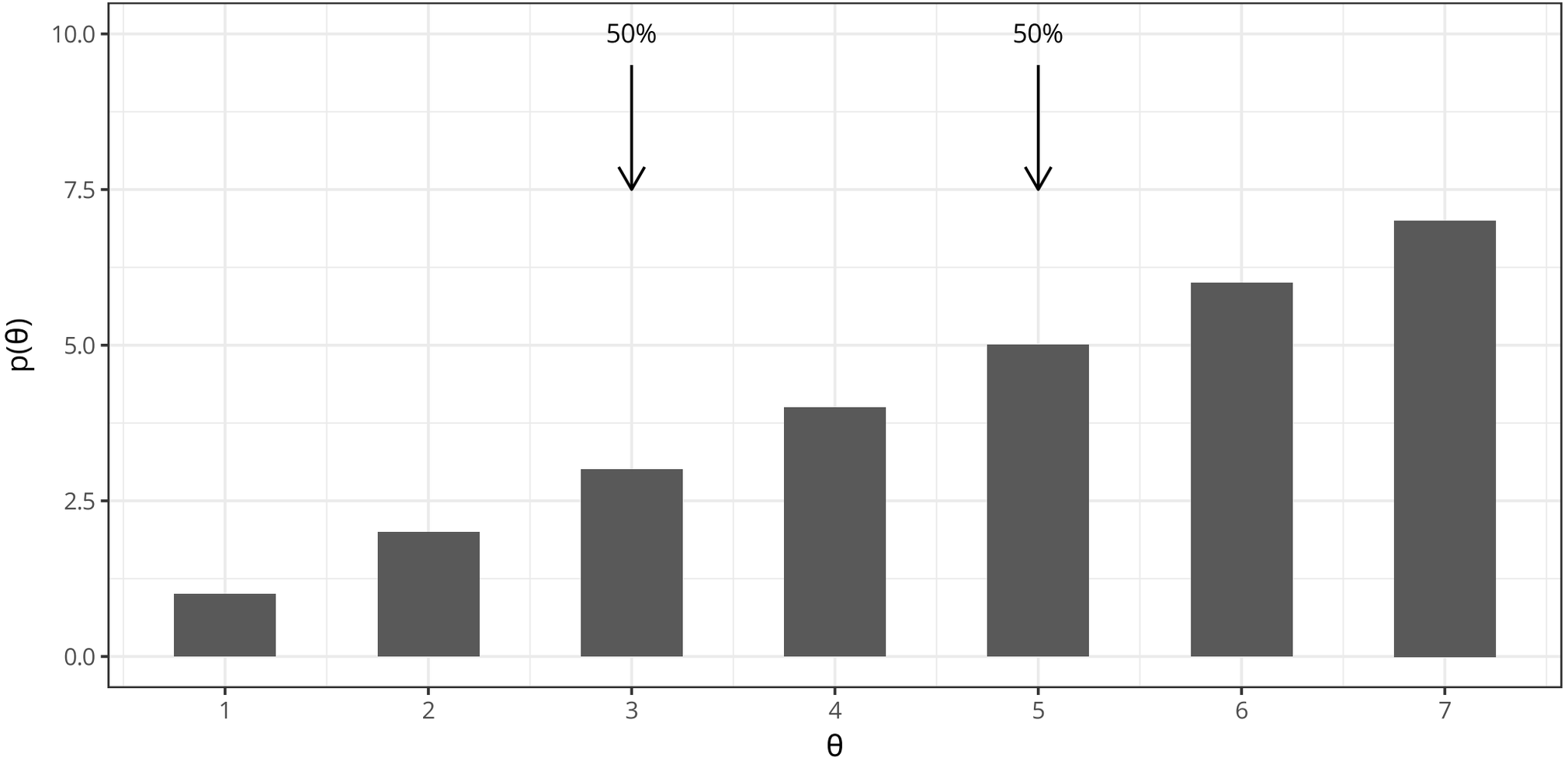
Algorithme Metropolis

Sélectionner un point de départ (on peut sélectionner n'importe quelle valeur).



Algorithme Metropolis

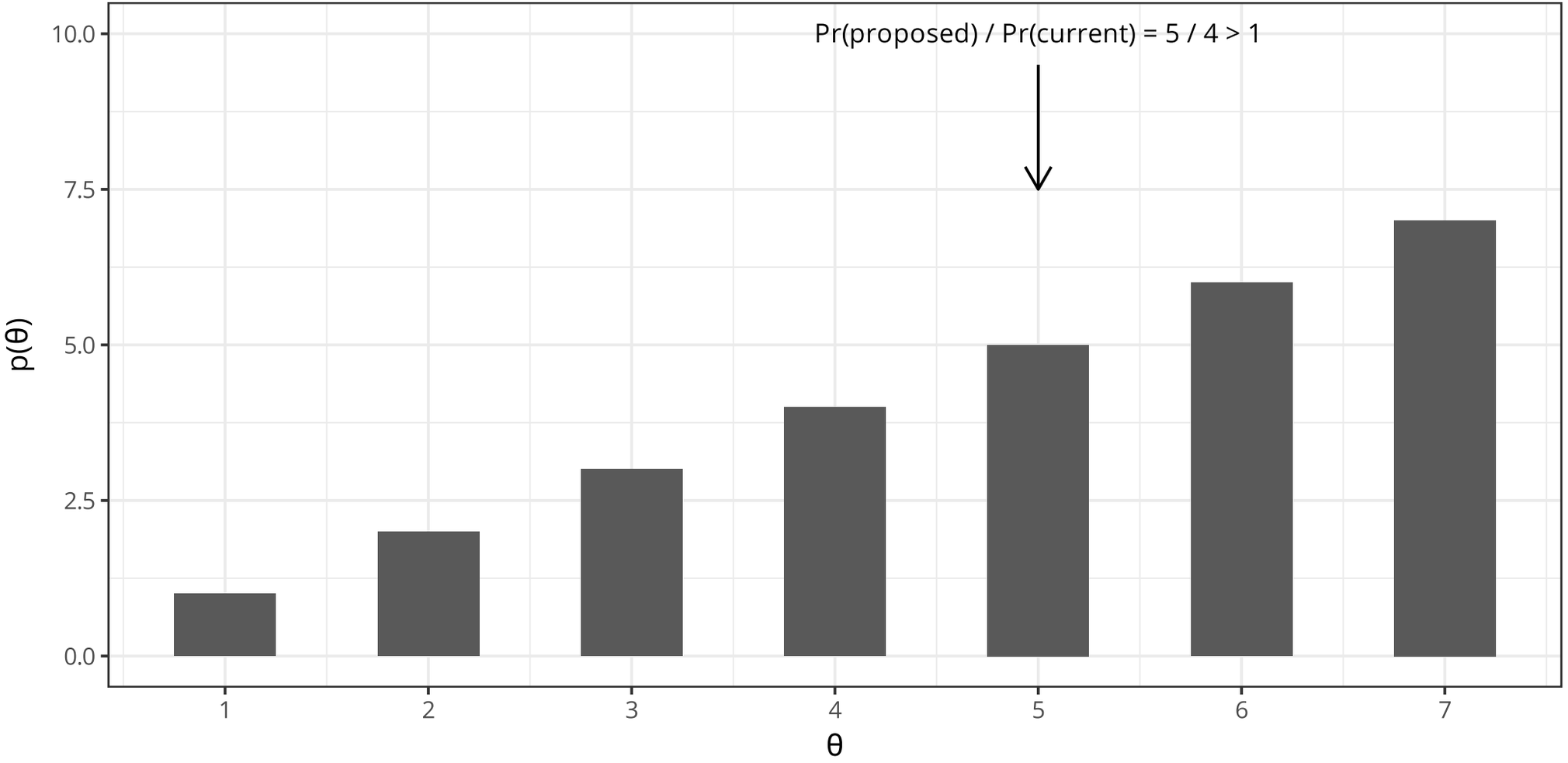
Faire une proposition de déplacement centrée sur la valeur courante de θ.



Algorithme Metropolis

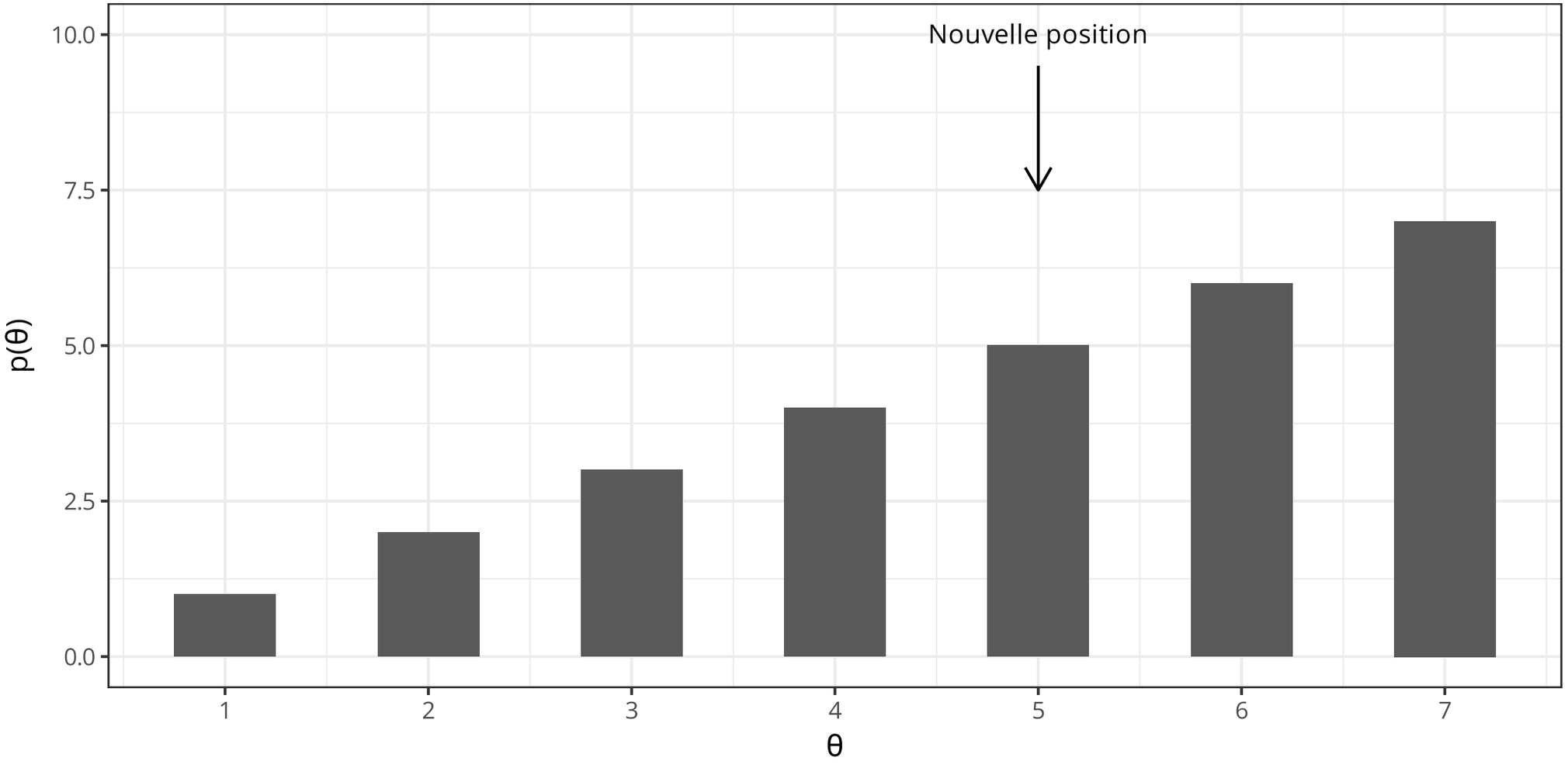
Calculer la **probabilité** d'accepter le déplacement selon la règle suivante :

$$\Pr_{\text{move}} = \min\left(\frac{\Pr(\theta_{\text{proposed}})}{\Pr(\theta_{\text{current}})}, 1\right)$$



Algorithme Metropolis

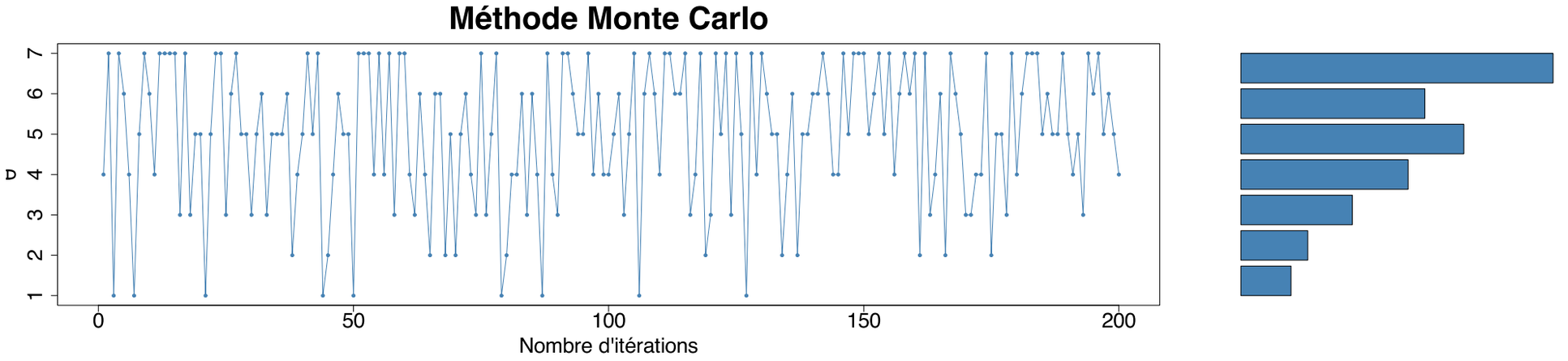
La position calculée devient la nouvelle position de départ et on répète l'algorithme.

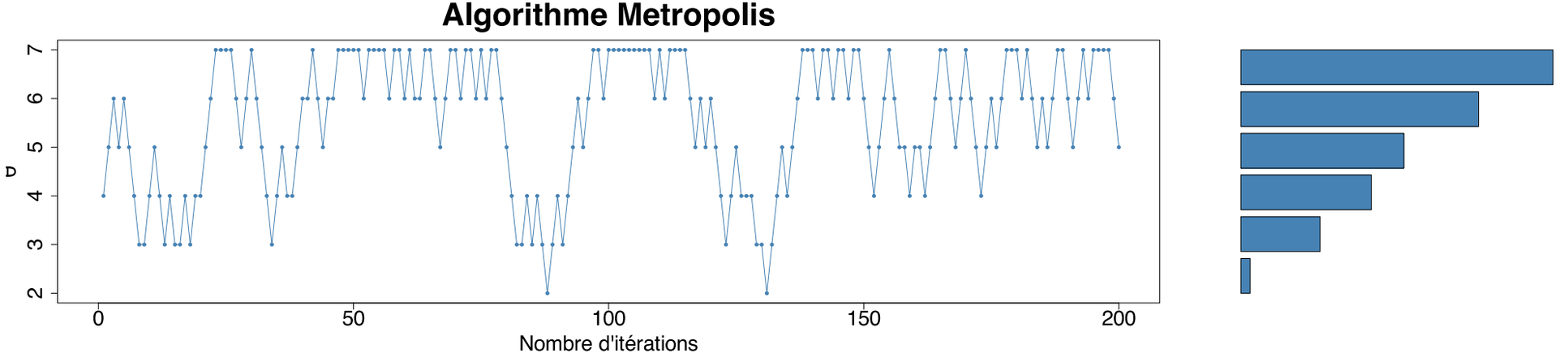


Algorithme Metropolis

```r
metropolis <- function (niter = 1e2, startval = 4) {

    x <- rep(0, niter) # initialise la chaîne (le vecteur) de longueur niter
    x[1] <- startval # définit la valeur de départ du paramètre

    for (i in 2:niter) {

        current <- x[i - 1] # valeur courante du paramètre
        proposal <- current + sample(c(-1, 1), size = 1)
        # on s'assure que la valeur proposée est bien dans l'intervalle [1, 7]
        if (proposal < 1) proposal <- 1
        if (proposal > 7) proposal <- 7
        # calcul de la probabilité de déplacement
        prob_move <- min(1, proposal / current)
        # on se déplace (ou pas) suivant cette probabilité
        # x[i] <- ifelse(prob_move > runif(n = 1, min = 0, max = 1), proposal, current)
        x[i] <- sample(c(proposal, current), size = 1, prob = c(prob_move, 1 - prob_move) )

    }

    return (x)
```

Méthodes Monte Carlo vs. Algorithme Metropolis





Algorithme Metropolis

Application au lancer de pièce (cas continu)

- La fonction de vraisemblance est donnée par : $p(y \mid \theta, n) \propto \theta^{y}(1-\theta)^{(n-y)}$
- Le prior est donné par : $p(\theta \mid a, b) \propto \theta^{(a-1)}(1-\theta)^{(b-1)}$
- Le paramètre que l'on cherche à estimer prend ses valeurs dans l'intervalle $[0, 1]$

Problème n°1 : Comment définir la proposition de déplacement ?

On peut modéliser le déplacement par une distribution normale : $\Delta\theta \sim \text{Normal}(0, \sigma)$

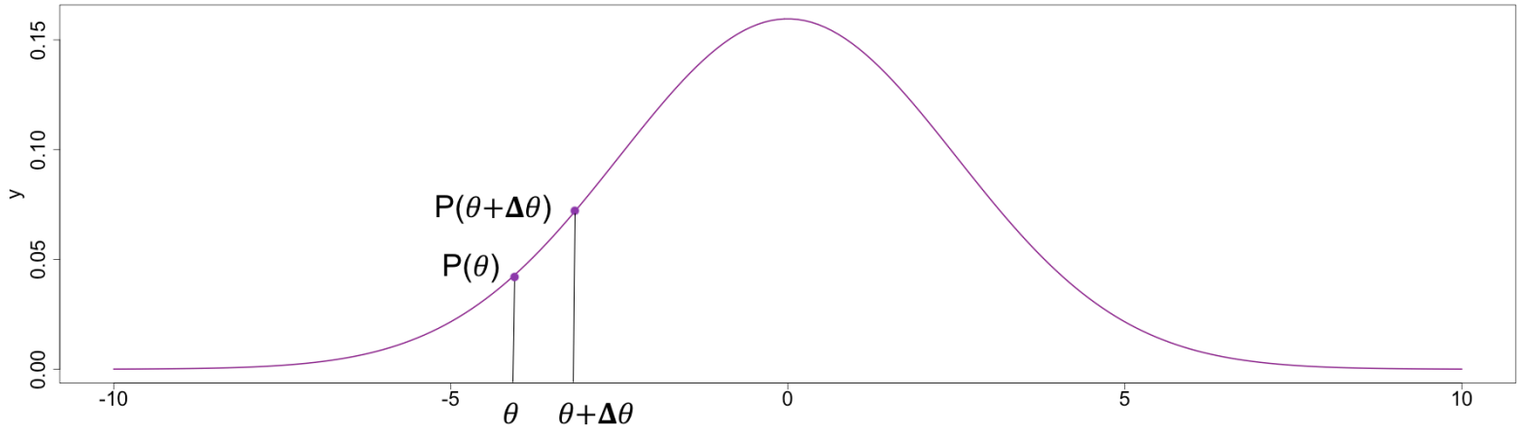
⟶ La moyenne μ vaut 0 : le déplacement se fait autour de la valeur courante du paramètre

⟶ La variance reste à déterminer, elle contrôle l'éloignement de la nouvelle valeur

Algorithme Metropolis

Problème n°2 : Quelle probabilité utiliser pour accepter ou refuser le déplacement ? Nous utilisons le produit de la vraisemblance et du prior : $\theta^{y}(1-\theta)^{(n-y)}\theta^{(a-1)}(1-\theta)^{(b-1)}$

La probabilité d'accepter le déplacement est donnée par : $\Pr_{\text{move}} = \min\left(\frac{\Pr(\theta_{\text{current}}+\Delta\theta)}{\Pr(\theta_{\text{current}})}, 1\right)$



REMARQUE : Le rapport $\frac{\Pr(\theta_{\text{current}}+\Delta\theta)}{\Pr(\theta_{\text{current}})}$ est le même que l'on utilise la distribution postérieure ou le produit prior par vraisemblance (car la constante de normalisation s'annule) !

Algorithme Metropolis

⟶ Sélectionner un point de départ

- Il faut choisir $\theta \in [0, 1]$
- Seule contrainte : $\Pr(\theta_{\text{initial}}) \neq 0$

⟶ Choisir une direction de déplacement

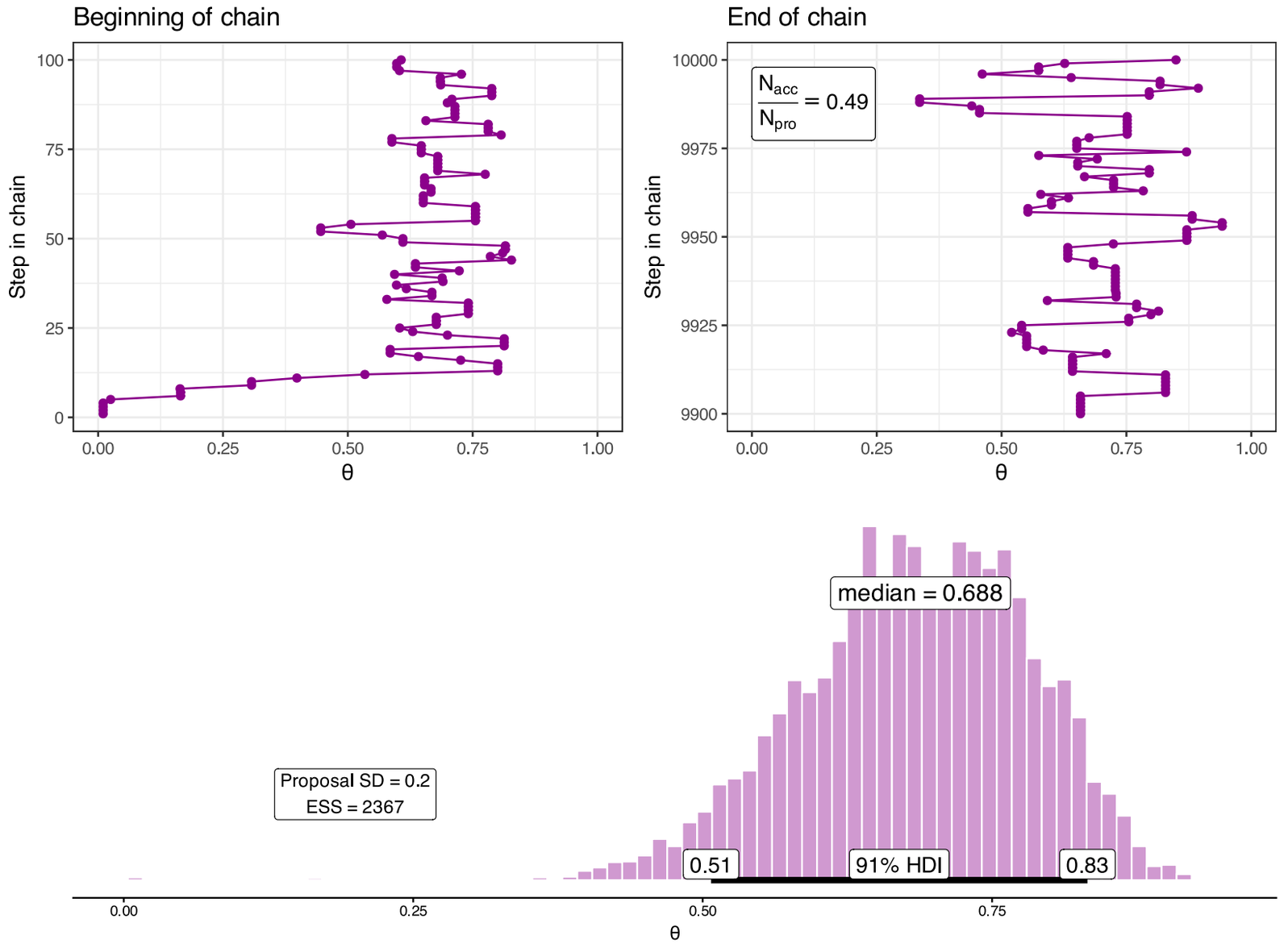
- Faire un tirage suivant $\text{Normal}(0, \sigma)$

⟶ Accepter ou rejeter la proposition de déplacement, suivant la probabilité :

$$\Pr_{\text{move}} = \min\left(\frac{\Pr(\theta_{\text{current}} + \Delta\theta)}{\Pr(\theta_{\text{current}})}, 1\right)$$

⟶ La position calculée devient la nouvelle position

Algorithme Metropolis



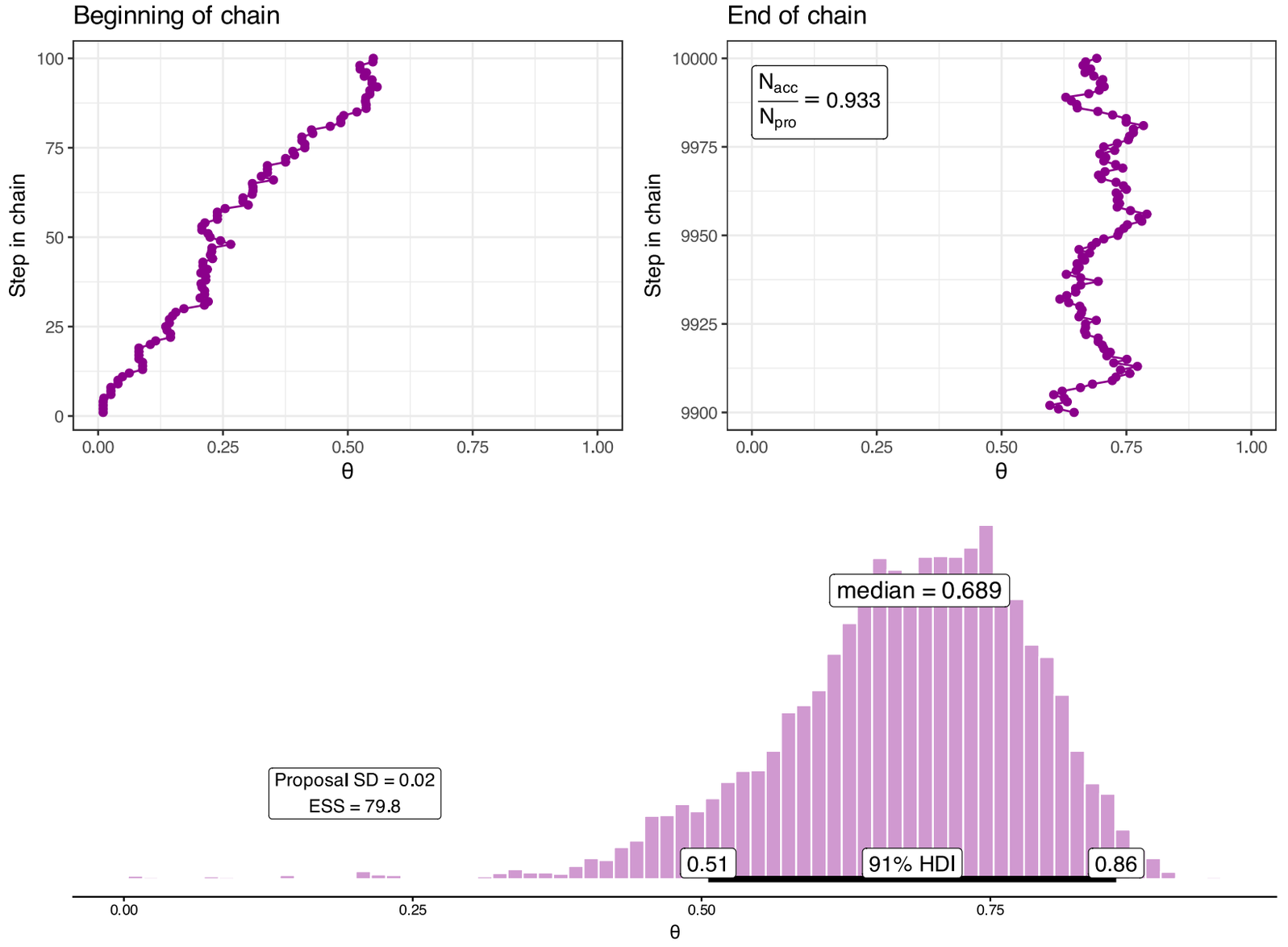
Algorithme Metropolis

Comment choisir σ pour la proposition de déplacement ? Deux indices permettent d'évaluer la qualité de l'échantillonnage :

⟶ Le rapport entre le nombre de déplacements proposés et le nombre de déplacements acceptés

⟶ L'effective sample size (i.e., le nombre de déplacements qui ne sont pas corrélés avec les précédents)

Algorithme Metropolis



Algorithme Metropolis

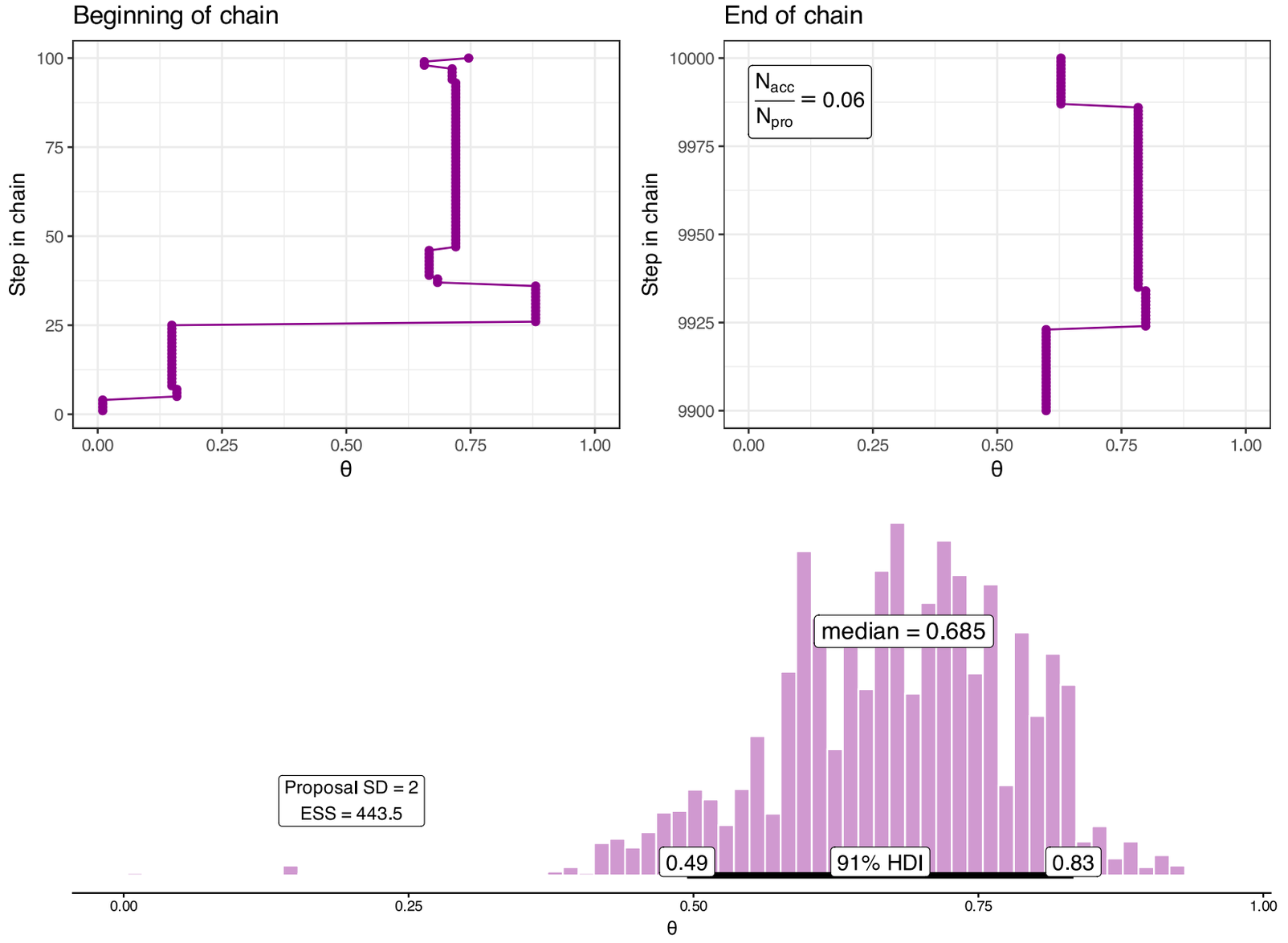
Le choix de sigma dans la proposition de déplacement

⟶ Toutes les propositions de déplacement (ou presque) sont acceptées

⟶ Peu de valeurs effectives

Il faut beaucoup d'itérations pour avoir un résultat satisfaisant...

Algorithme Metropolis



Algorithme Metropolis

Le choix de sigma dans la proposition de déplacement

⟶ Les propositions de déplacement sont rarement acceptées

⟶ Peu de valeurs effectives...

Il faut beaucoup d'itérations pour obtenir un résultat satisfaisant...

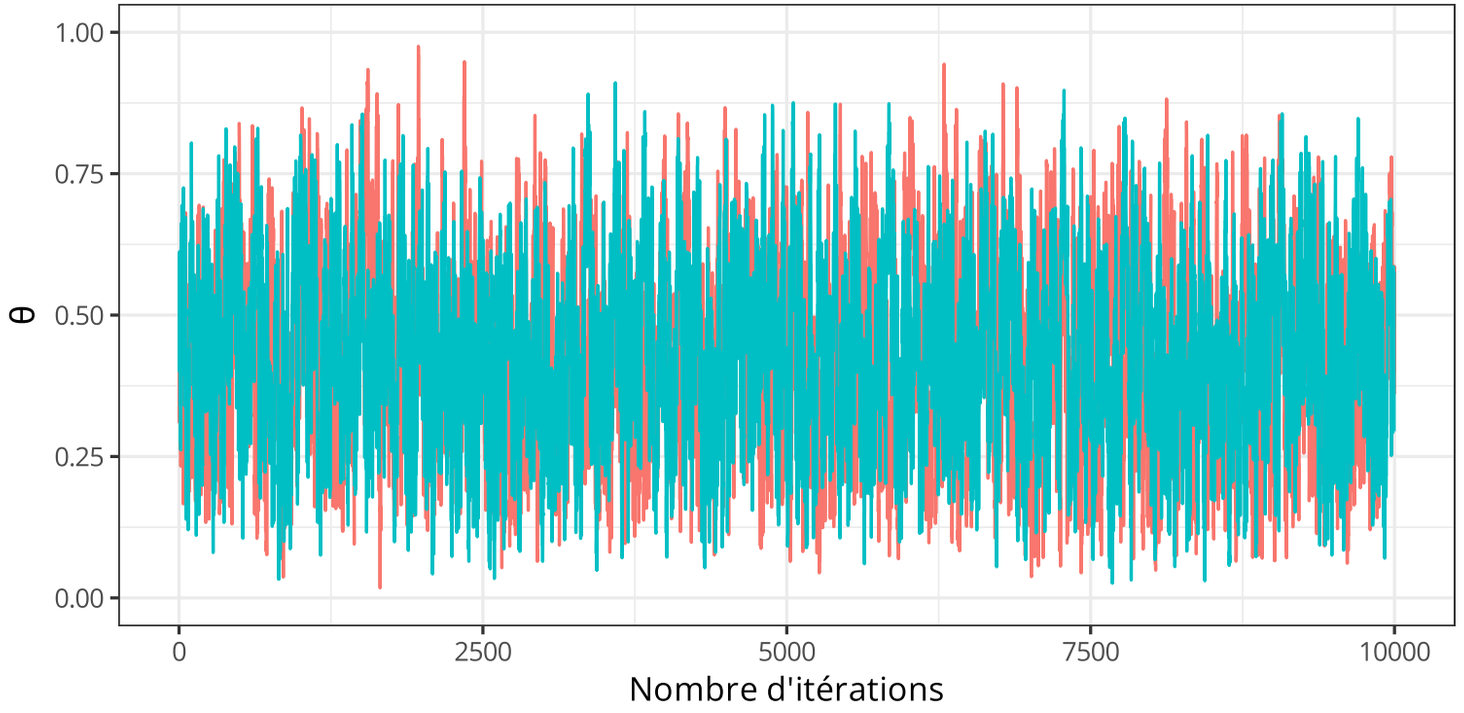
Algorithme Metropolis[1]

```r
metropolis_beta_binomial <- function (niter = 1e2, startval = 0.5) {

    x <- rep(0, niter) # initialise la chaîne (le vecteur) de longueur niter
    x[1] <- startval # définit la valeur de départ du paramètre

    for (i in 2:niter) {

        current <- x[i - 1] # valeur courante du paramètre
        current_plaus <- dbeta(current, 2, 3) * dbinom(1, 2, current)
        # proposal <- runif(n = 1, min = current - w, max = current + w) # valeur proposée
        proposal <- rnorm(n = 1, mean = current, sd = 0.1) # valeur proposée
        # on s'assure que la valeur proposée est bien dans l'intervalle [0, 1]
        if (proposal < 0) proposal <- 0
        if (proposal > 1) proposal <- 1
        proposal_plaus <- dbeta(proposal, 2, 3) * dbinom(1, 2, proposal)
        # calcul de la probabilité de déplacement
        alpha <- min(1, proposal_plaus / current_plaus)
        # on se déplace (ou pas) suivant cette probabilité
        x[i] <- sample(c(current, proposal), size = 1, prob = c(1 - alpha, alpha) )

    }
```

1. L'algorithme Metropolis-Hastings est une extension de l'algorithme Metropolis qui permet de faire des propositions de déplacement non symmétrique. Voir https://en.wikipedia.org/wiki/Metropolis–Hastings_algorithm.

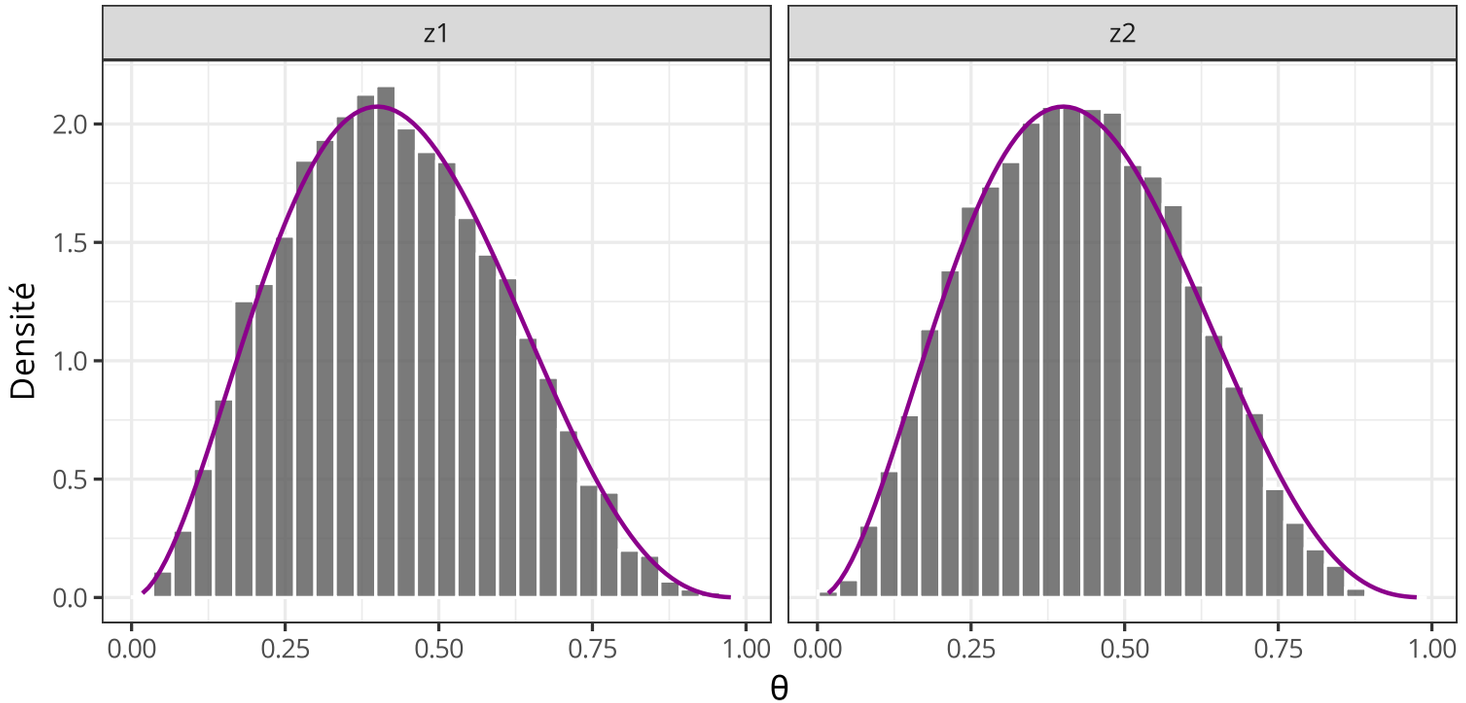
Algorithme Metropolis

```r
z1 <- metropolis_beta_binomial(niter = 1e4, startval = 0.5)
z2 <- metropolis_beta_binomial(niter = 1e4, startval = 0.5)

data.frame(z1 = z1, z2 = z2) %>%
  mutate(sample = 1:nrow(.) ) %>%
  pivot_longer(cols = z1:z2) %>%
  ggplot(aes(x = sample, y = value, colour = name) ) +
  geom_line(show.legend = FALSE) +
  labs(x = "Nombre d'itérations", y = expression(theta) ) + ylim(c(0, 1) )
```



Algorithme Metropolis

```
data.frame(z1 = z1, z2 = z2) %>%
  pivot_longer(cols = z1:z2) %>%
  rownames_to_column() %>%
  mutate(rowname = as.numeric(rowname) ) %>%
  ggplot(aes(x = value) ) +
  geom_histogram(aes(y = ..density..), color = "white", alpha = 0.8) +
  stat_function(fun = dbeta, args = list(3, 4), color = "magenta4", size = 1) +
  facet_wrap(~name) +
  labs(x = expression(theta), y = "Densité")
```



Algorithme Metropolis-Hastings

Algorithme Hamiltonian Monte Carlo

Les algorithmes Metropolis et Metropolis-Hastings (ou Gibbs) ont de mauvaises performances lorsque les paramètres du modèle sont fortement corrêlés. L'algorithme **Hamiltonian Monte Carlo** résout ces problème en utilisant la géométrie de l'espace postérieur. On va adapter la proposition de déplacement à la géométrie de la distribution postérieure aux alentours de la position courante.

On utilise l'opérateur hamiltonien (hamiltonians) qui représente l'énergie totale d'un système. Cette énergie se décompose en l'énergie potentielle (qui dépend de la position dans l'espace des paramètres θ) et son énergie cinétique, qui dépend de son **moment** (momentum, m) :

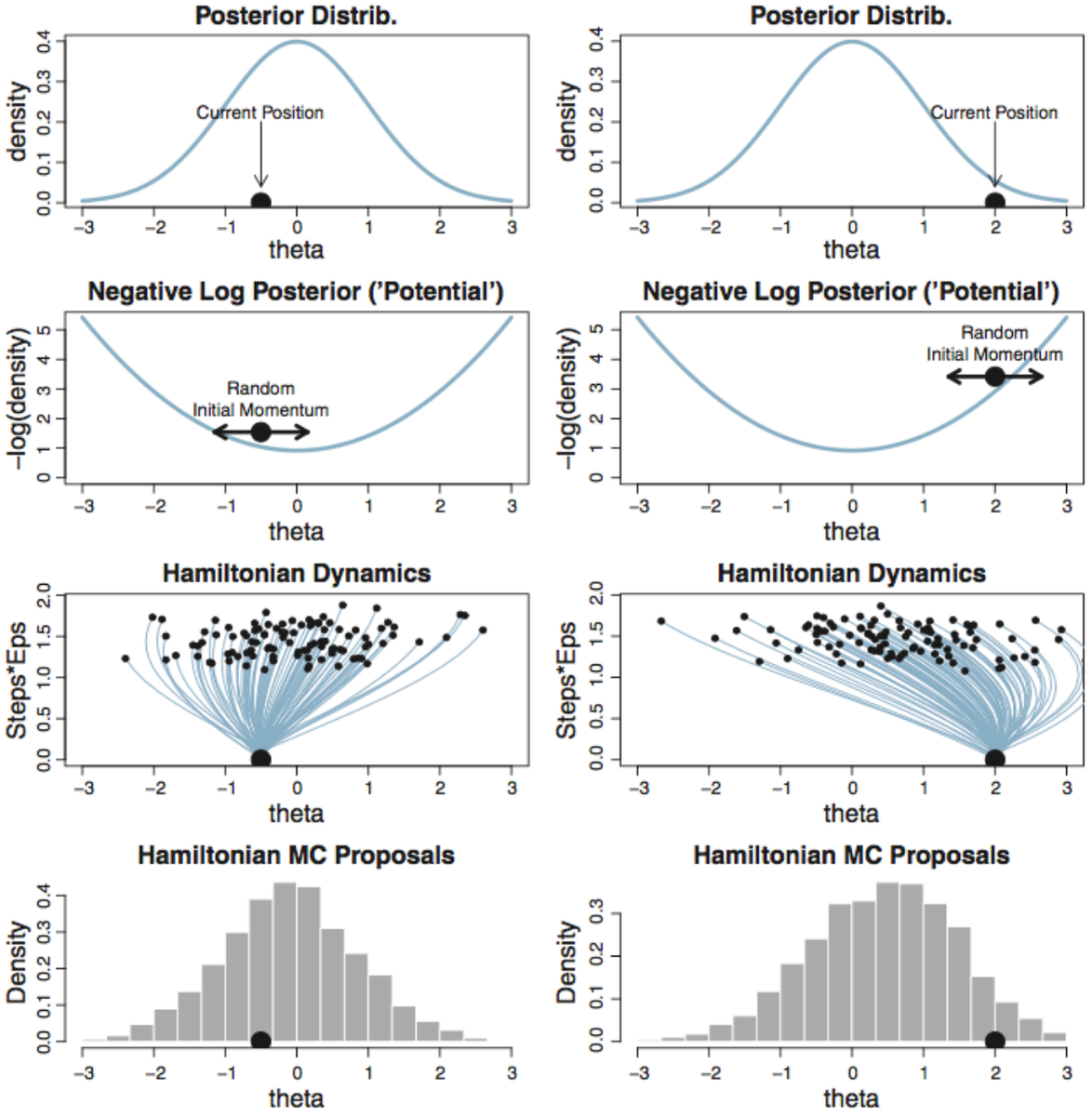
$$H(\theta, m) = \underbrace{U(\theta)}_{\text{énergie potentielle}} + \underbrace{KE(m)}_{\text{énergie cinétique}}$$

L'énergie potentielle est donnée par le négatif du log de la densité postérieure (non-normalisée) ;

$$U(\theta) = -\log[p(\text{data} \mid \theta) \times p(\theta)]$$

Quand la densité postérieure augmente, l'énergie potentielle diminue (i.e., devient plus négative).

Algorithme Hamiltonian Monte Carlo



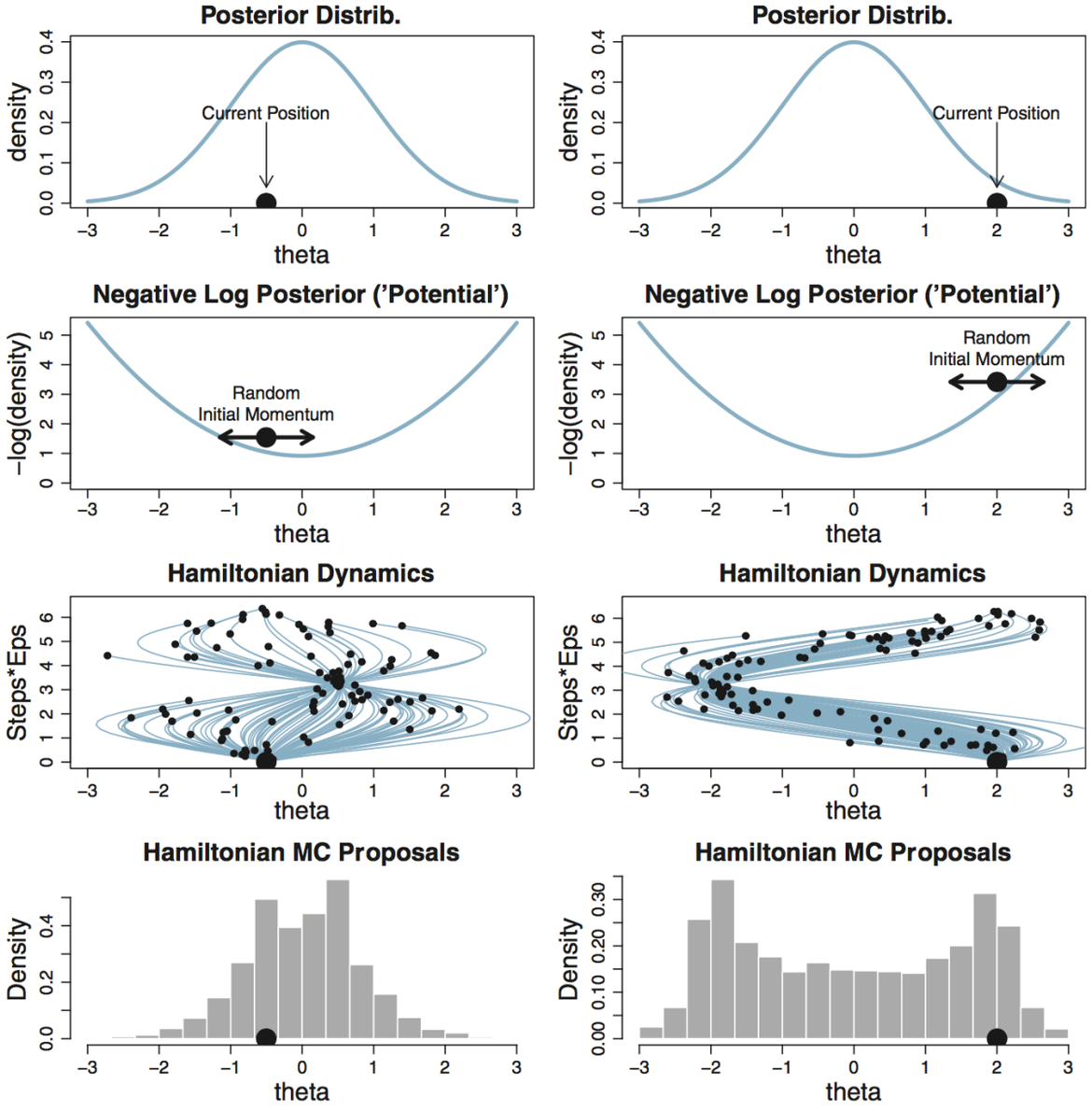
Algorithme Hamiltonian Monte Carlo

- Sélectionner un point de départ θ_0 : On peut sélectionner n'importe quelle valeur de θ dans l'espace postérieur.
- On génère aléatoirement la force avec laquelle on lance la bille (moment), par exemple à partir d'une loi normale multivariée : $m \sim \text{MVNormal}(\mu, \Sigma)$.
- On utilise un algorithme d'approximation de la trajectoire (e.g., leapfrog) pour estimer la trajectoire et la position finale de la bille dans l'espace postérieur pour une certaine durée.
- Après un certain temps, on enregistre la position finale de la bille et son moment.
- On accepte ou rejette la proposition de déplacement suivant la probabilité suivante (où ϕ (phi) est le moment associé à la bille) :

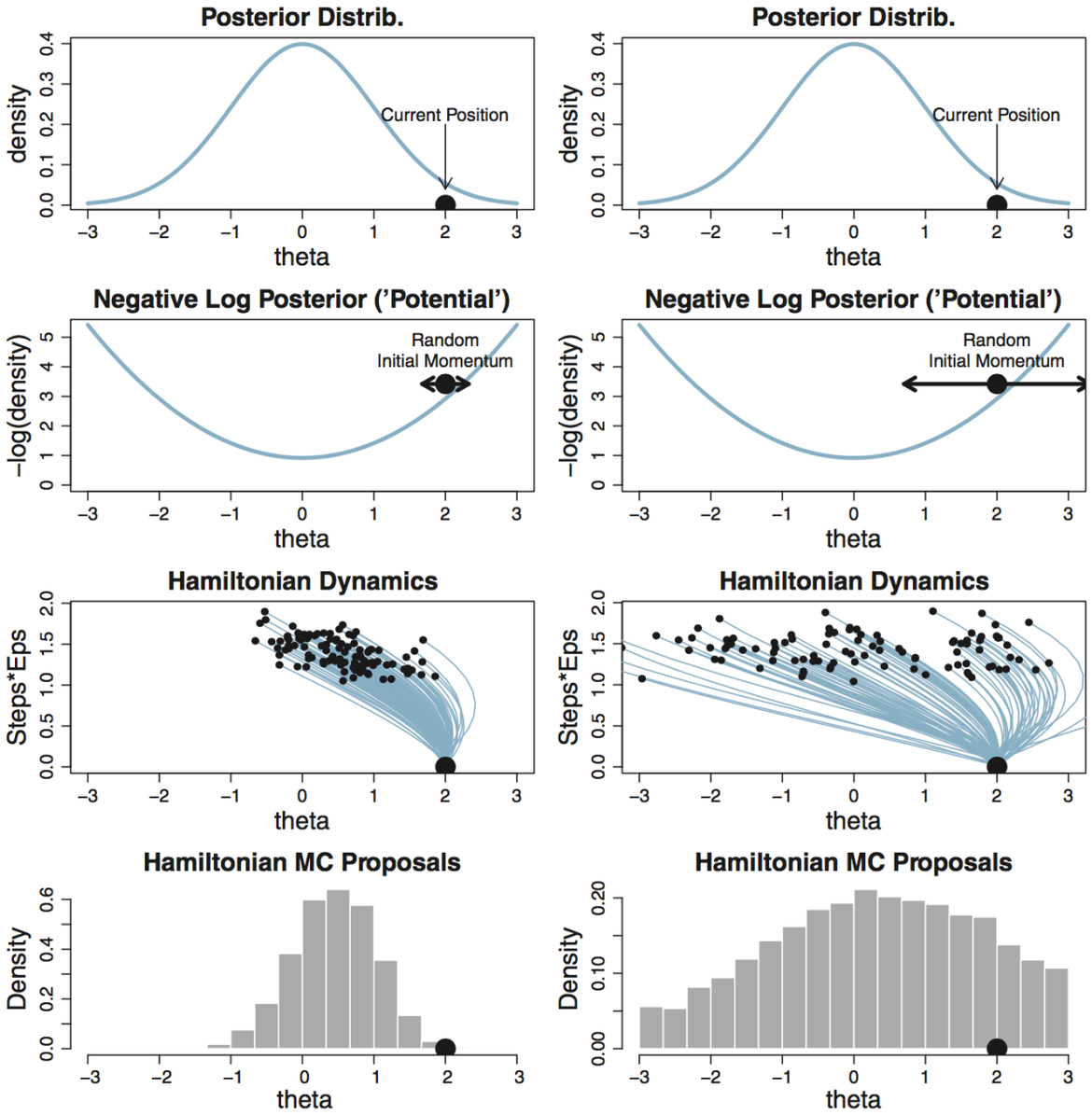
$$\Pr_{\text{move}} = \min\left(\frac{p(\theta_{\text{proposed}} \mid \text{data})\ p(\phi_{\text{proposed}})}{p(\theta_{\text{current}} \mid \text{data})\ p(\phi_{\text{current}})}, 1\right)$$

- On enregistre la nouvelle position et on recommence...

Influence de la durée de déplacement...



Influence de la variabilité du moment initial...



Algorithme Hamiltonian Monte Carlo

Évaluation des MCMC

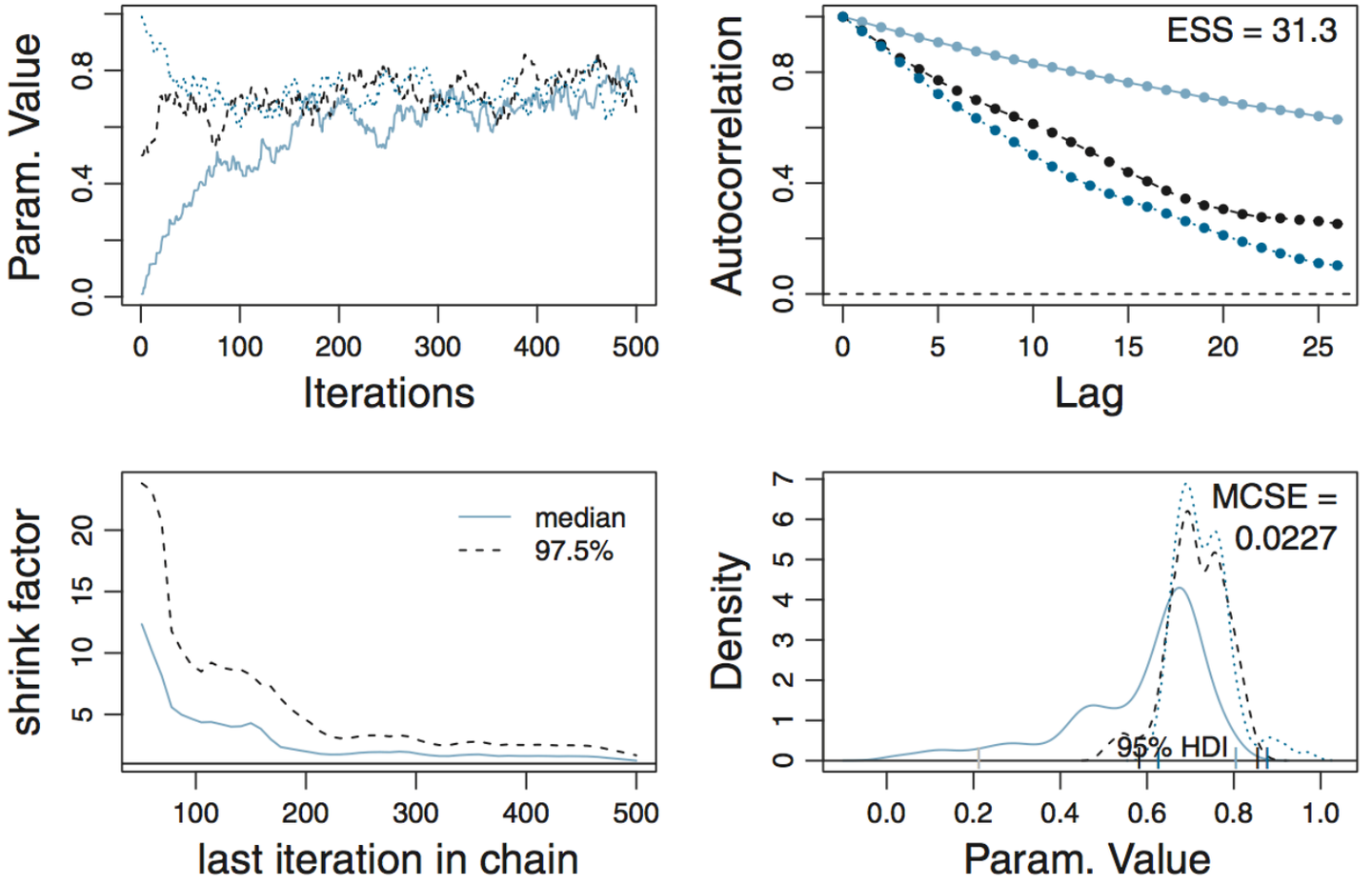
Ces méthodes peuvent ne pas converger vers la “vraie” distribution postérieure, en raison du temps de calcul limité, du paramétrage de certains hyper-paramètres (e.g., variance de la distribution normale de la proposition, ou variance du moment initial pour HMC).

Ces méthodes produisent des chaînes de valeurs de paramètres (échantillons). L’utilisation de tel ou tel algorithme MCMC pour échantillonner la distribution postérieure repose sur trois objectifs :

- Les valeurs de la chaîne doivent être représentatives de la distribution postérieure. Ces valeurs ne doivent pas dépendre du point de départ. Ces valeurs ne doivent pas être cantonnées à une région particulière de l’espace des paramètres.
- La chaîne doit être suffisamment longue pour assurer la précision et la stabilité du résultat. La tendance centrale et le HDI calculés à partir de la chaîne ne doivent pas changer si on relance la procédure.
- La chaîne doit être générée de manière efficace (i.e., avec le moins d’itérations possible).

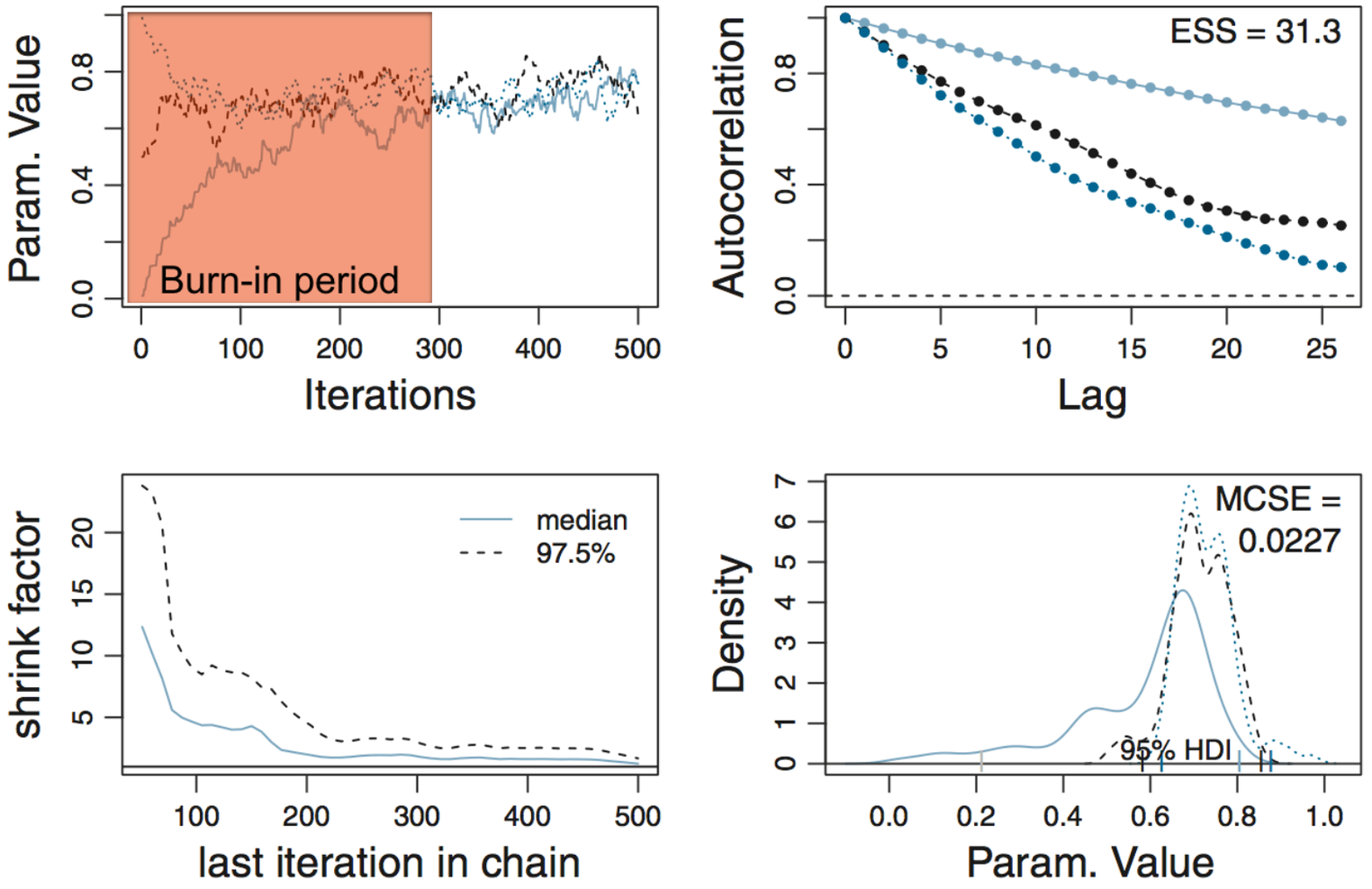
Évaluation des MCMC - Représentativité

- Vérification visuelle des trajectoires : Les chaînes doivent occuper le même espace, la convergence ne dépend pas du point de départ, aucune chaîne ne doit avoir de trajectoire particulière (e.g., cyclique).
- Vérification visuelle des densités : Les densités doivent se superposer.



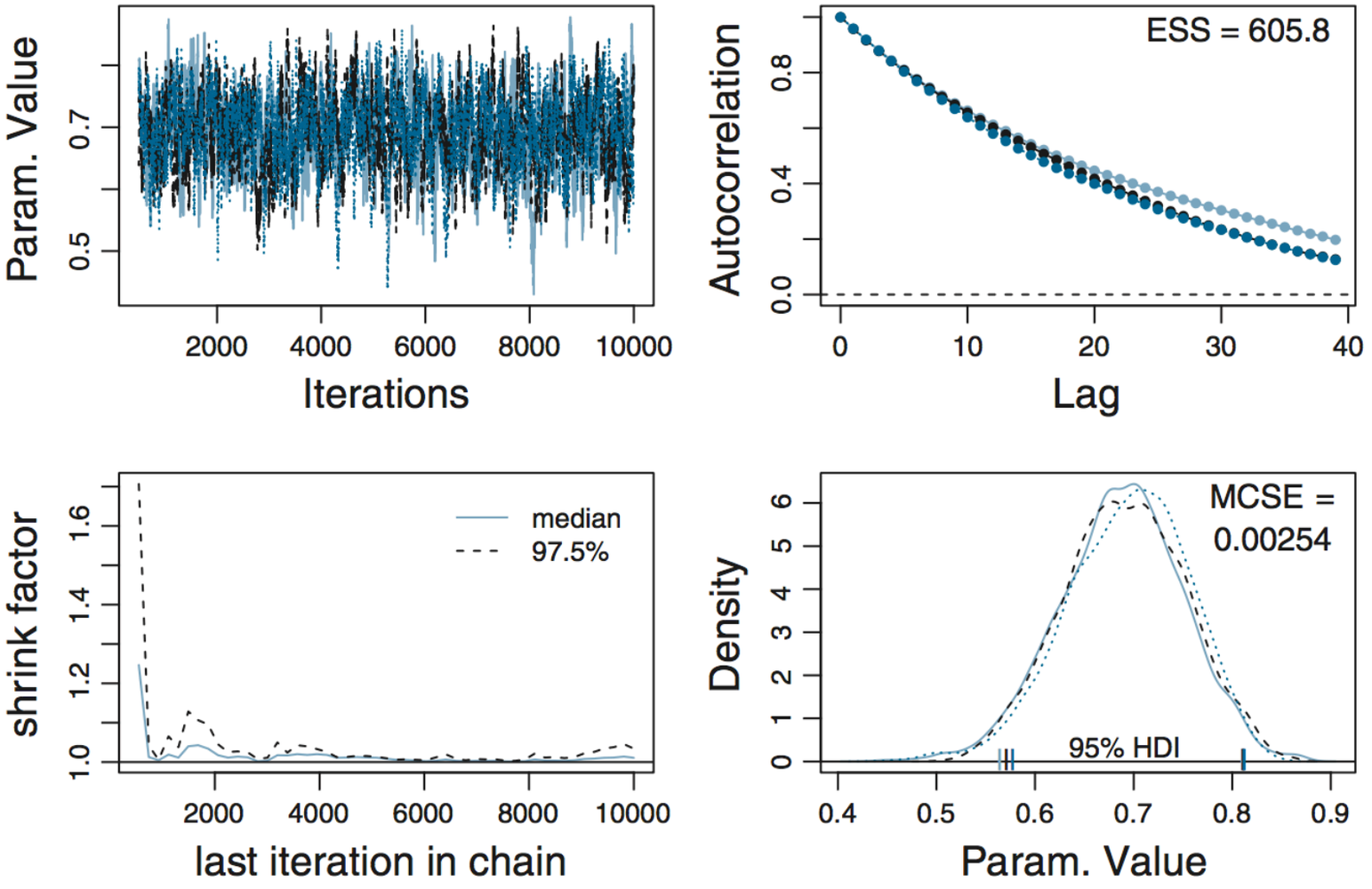
Évaluation des MCMC - Représentativité

Cette affichage ne montre que les 500 premières itérations. Les trajectoires ne se superposent pas au début (zone orange). La densité est également affectée. En pratique on supprime ces premières itérations (période de “burn-in” ou de “warm-up”).



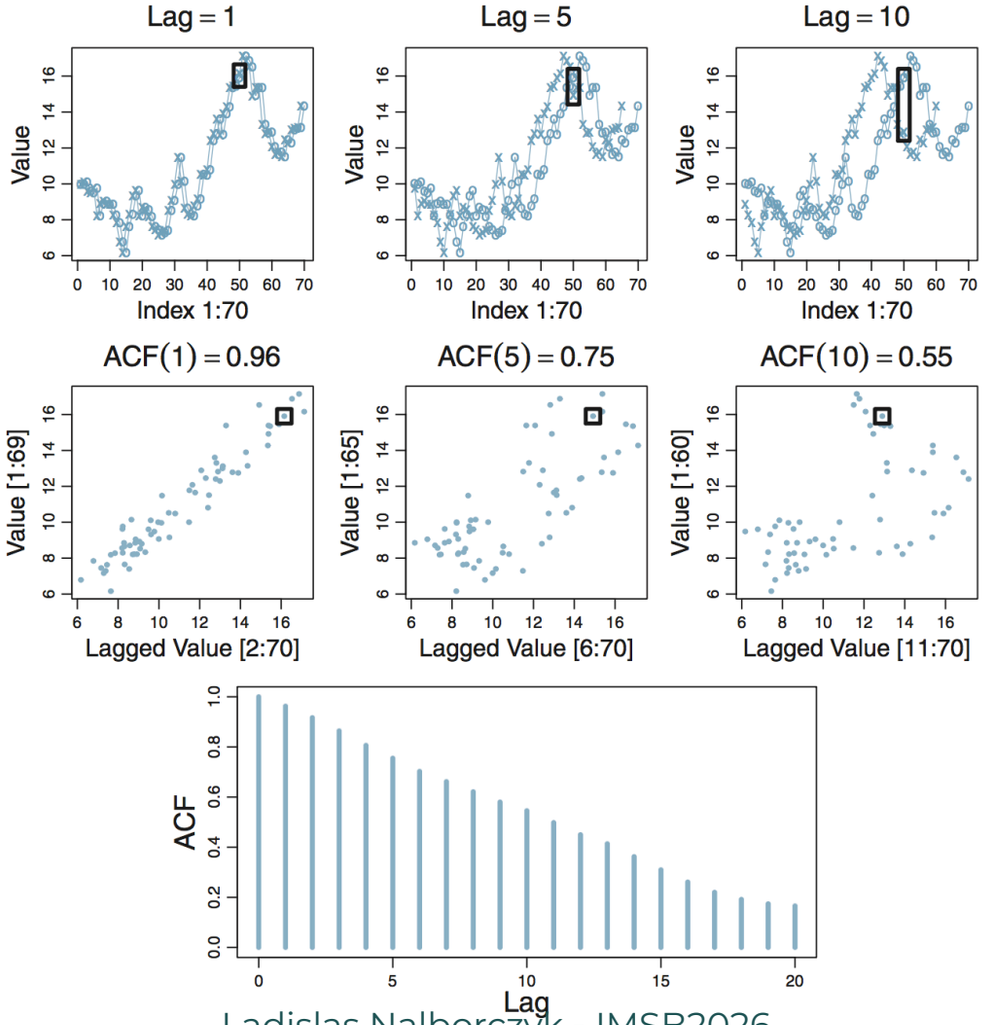
Évaluation des MCMC - Représentativité

Vérification numérique des chaînes : Le **shrink factor** (aussi connu comme $\widehat{R}$ ou `Rhat`) est le rapport entre la variance inter-chaînes et intra-chaîne. Cette valeur devrait idéalement tendre vers 1 (on la considère comme acceptable jusqu'à 1.1).



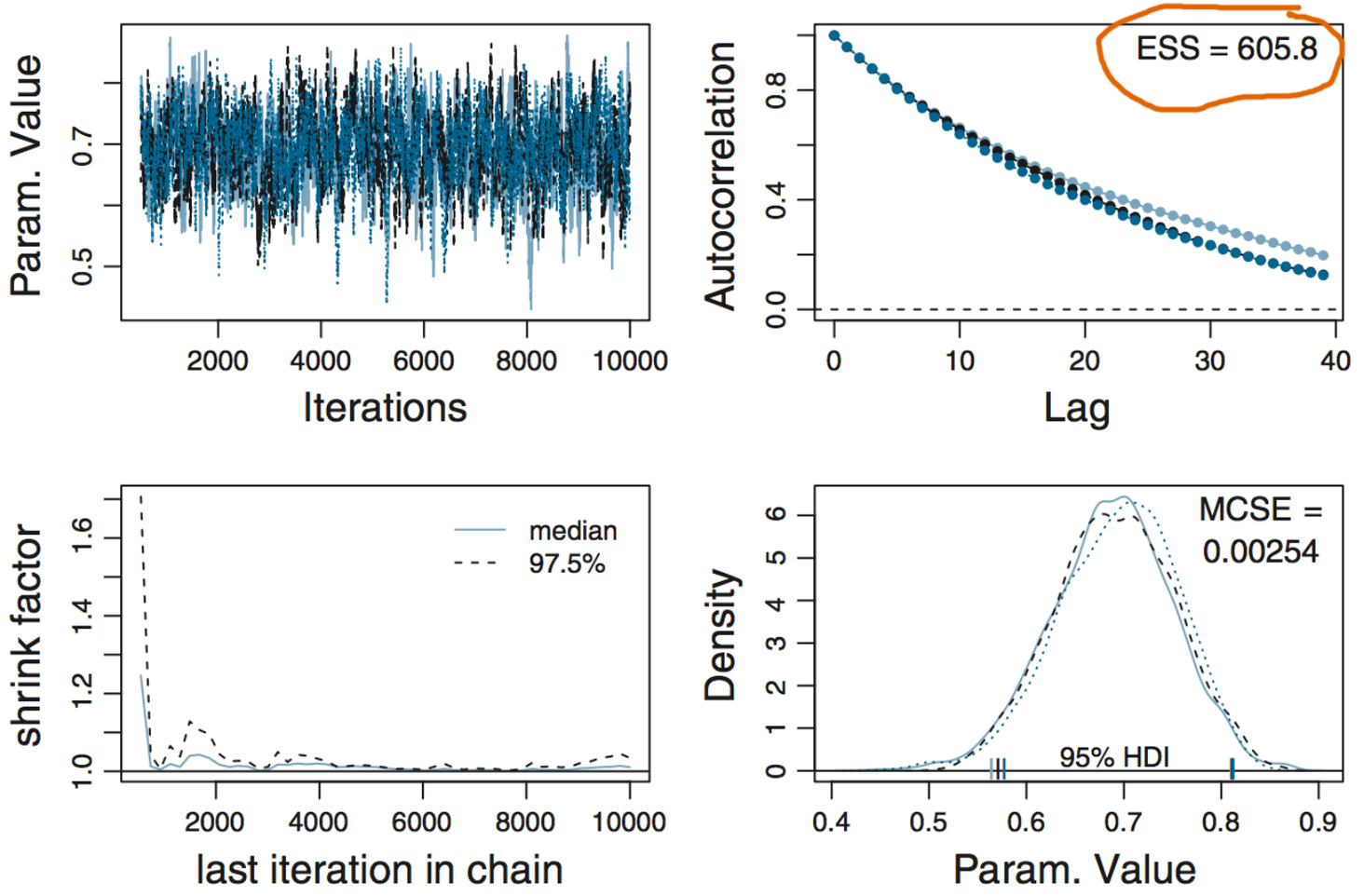
Évaluation des MCMC - Stabilité et précicion

Plus la chaîne est longue et plus le résultat sera précis et stable. Si la chaîne “s’attarde” sur chaque position, et que le nombre d’itérations reste le même, alors on perd en précision. Il lui faudra plus d’itérations pour arriver au même niveau de précision. L’autocorrélation est la corrélation de la chaîne avec elle-même mais décalé de k itérations (lag).



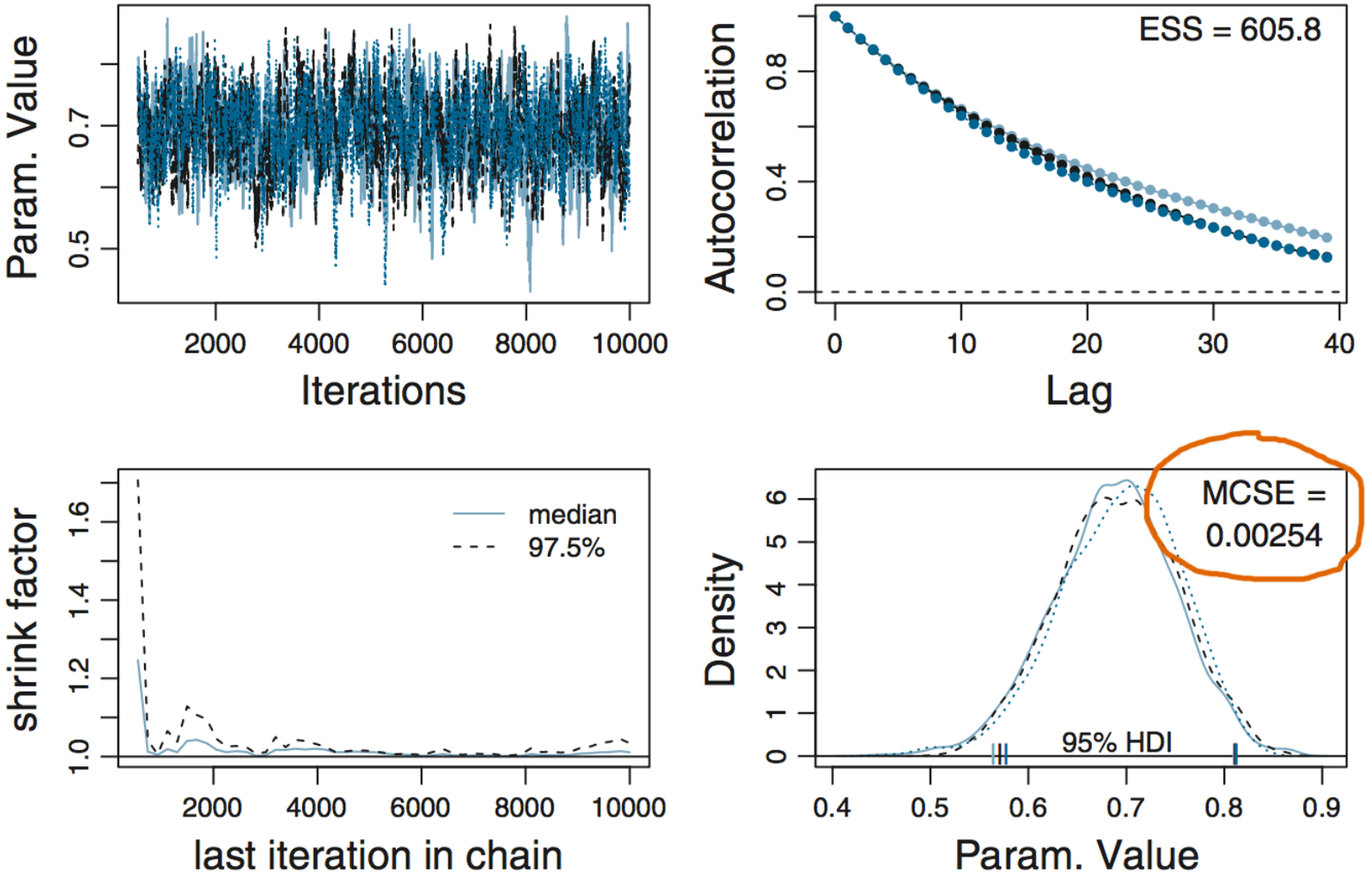
Évaluation des MCMC - Stabilité et précicion

La fonction d'autocorrélation est représentée pour chaque chaîne (en haut à droite). Un autre résultat rend compte de la précision de l'échantillon : l'effective sample size, $ESS = \frac{N}{1+2\sum_k ACF(k)}$. Il représente la taille d'un échantillon non-autocorrélé extrait de la somme de toutes les chaînes. Pour une précision raisonnable du HDI, il est recommandé d'avoir un ESS supérieur à 1000.



Évaluation des MCMC - Stabilité et précicion

L'erreur standard d'un ensemble d'échantillons est donné par : $SE = SD/\sqrt{N}$. Plus N augmente, plus l'erreur standard diminue. On peut généraliser cette idée aux chaînes de Markov : $MCSE = SD/\sqrt{ESS}$. Pour une précision raisonnable de la tendance centrale, il faut que cette valeur soit faible.

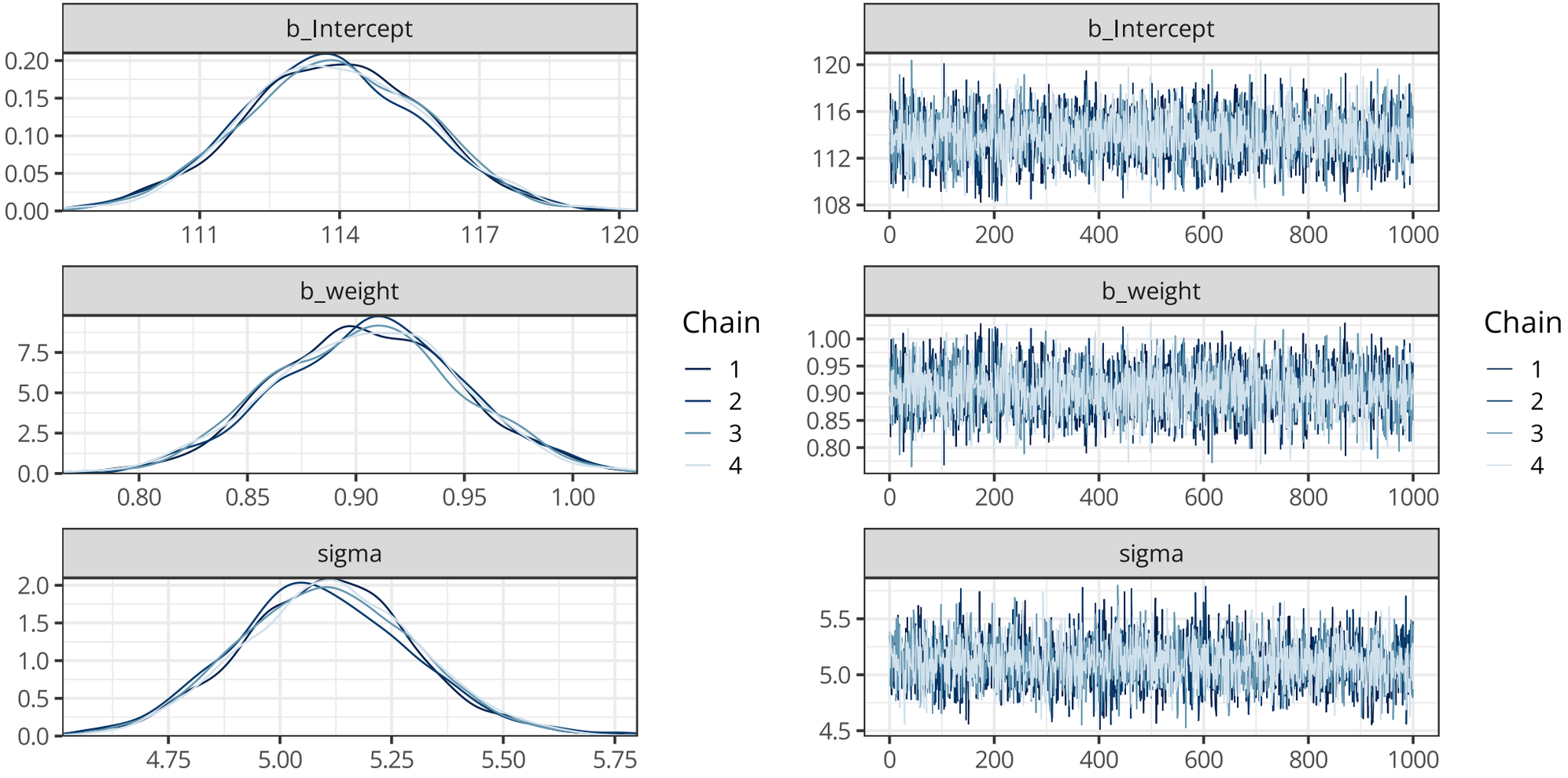


Évaluation des MCMC - Implémentation via brms

```r
library(tidyverse)
library(imsb)
library(brms)

d <- open_data(howell)
d2 <- d %>% filter(age >= 18)

priors <- c(
  prior(normal(150, 20), class = Intercept),
  prior(normal(0, 10), class = b),
  prior(exponential(0.01), class = sigma)
  )

mod1 <- brm(
  formula = height ~ 1 + weight,
  prior = priors,
  family = gaussian(),
  data = d2,
  chains = 4, # nombre de MCMCs
  iter = 2000, # nombre total d'itérations (par chaîne)
  warmup = 1000, # nombre d'itérations pour le warm-up
```

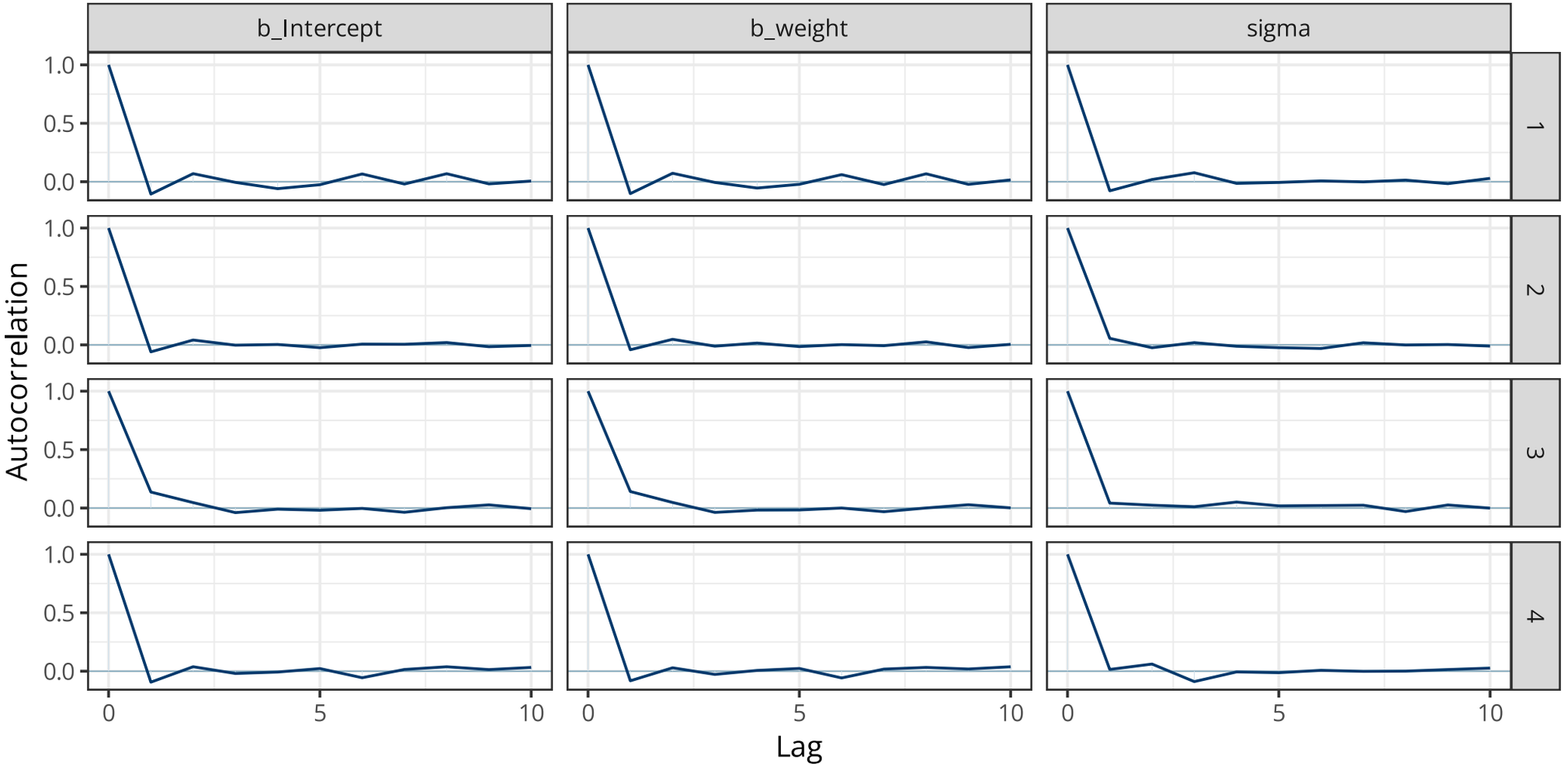
Évaluation des MCMC - Implémentation via brms

```
# combo can be hist, dens, dens_overlay, trace, trace_highlight...
# cf. https://mc-stan.org/bayesplot/reference/MCMC-overview.html
plot(x = mod1, combo = c("dens_overlay", "trace") )
```



Évaluation des MCMC - Implémentation via brms

```
1 library(bayesplot)
2 post <- posterior_samples(mod1, add_chain = TRUE)
3 post %>% mcmc_acf(pars = vars(b_Intercept:sigma), lags = 10)
```



Évaluation des MCMC - Implémentation via brms

```
1 summary(mod1)
```

```
 Family: gaussian
  Links: mu = identity
Formula: height ~ 1 + weight
   Data: d2 (Number of observations: 352)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept   113.87      1.93   110.10   117.63 1.00     3986     2758
weight        0.91      0.04     0.82     0.99 1.00     3939     2847

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     5.11      0.20     4.73     5.51 1.00     3717     2996

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

Évaluation des MCMC - Implémentation via brms

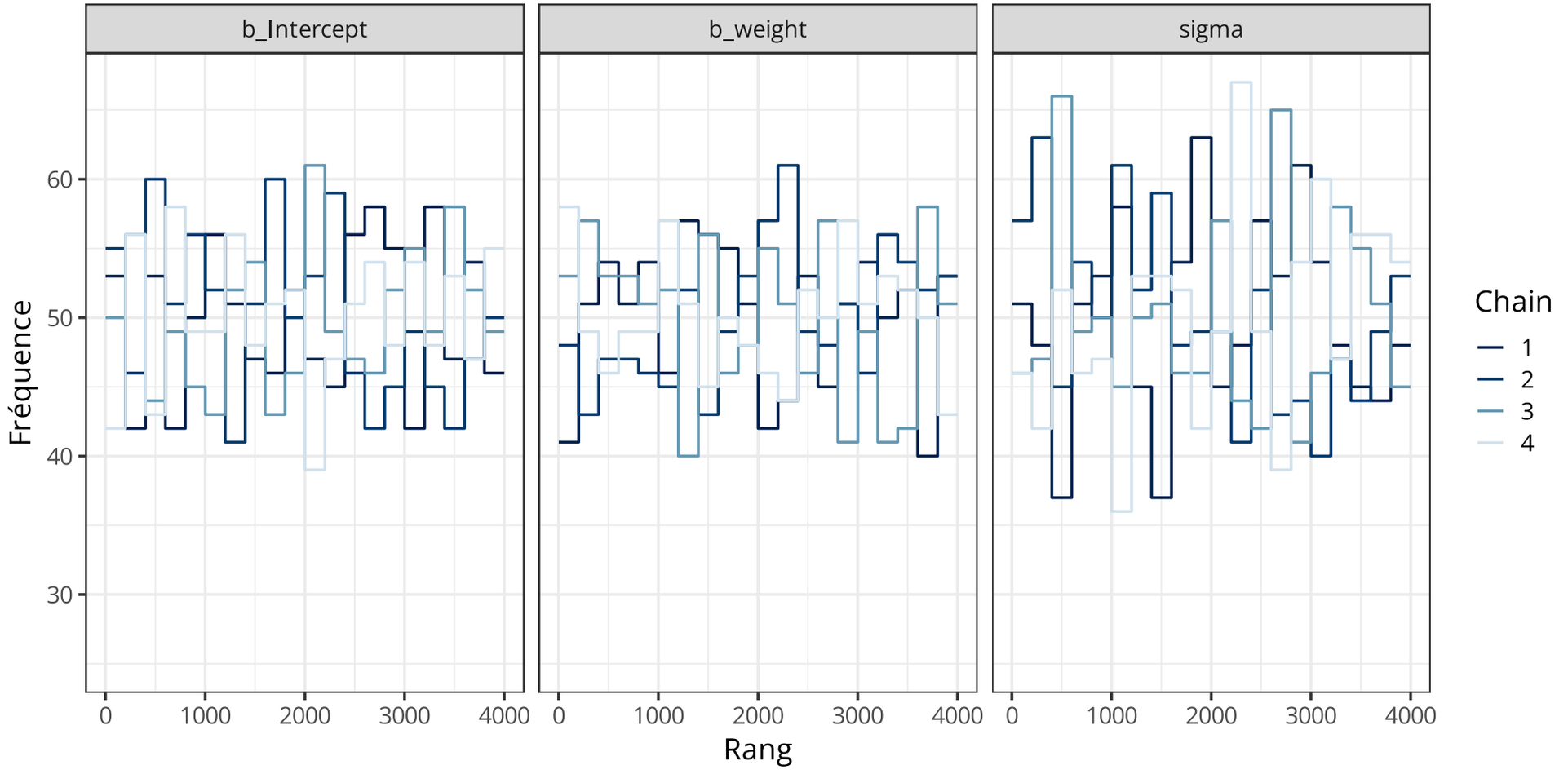
Bulk-ESS fait référence à l'ESS calculé sur la distribution des échantillons normalisée par leur rang, et plus particulièrement autour de la position centrale de cette distribution (e.g., moyenne ou médiane). On recommande que le Bulk-ESS soit au moins 100 fois plus élevé que le nombre de chaînes (i.e., pour 4 chaînes, le Bulk-ESS devrait être d'au moins 400).

Tail-ESS donne le minimum de l'ESS calculé pour les quantiles à 5% et 95% (i.e., pour les queues de la distribution des échantillons normalisés par leur rang). Cette valeur doit être élevée si nous accordons de l'importance à l'estimation des valeurs extrêmees (par exemple pour calculer un intervalle de crédibilité).

Quand tout va mal, voir ces recommendations de l'équipe de Stan concernant les choix de prior, ou ce guide concernant les messages d'erreur fréquents. Voir aussi l'article récent ou cet article de blog introduisant ces nouveaux indices.

Évaluation des MCMC - Implémentation via brms

```
post %>% # rank plots
  mcmc_rank_overlay(pars = vars(b_Intercept:sigma) ) +
  labs(x = "Rang", y = "Fréquence") +
  coord_cartesian(ylim = c(25, NA) )
```



Résumé du cours

Nous avons introduit et discuté l'utilisation d'échantillonneurs pour obtenir des échantillons issus de la distribution postérieure (non-normalisée). Ces échantillons peuvent ensuite être utilisés pour calculer différentes statistiques de la distribution postérieure (e.g., moyenne, médiane, HDI).

L'algorithme Metropolis-Hastings peut être utilisé pour n'importe quel problème pour lequel une vraisemblance peut être calculée. Cependant, bien que cet algorithme soit simple à coder, sa convergence peut être très lente... De plus, cet algorithme ne fonctionne pas bien lorsqu'il existe de fortes corrélations entre les différents paramètres...

L'algorithme HMC évite ces problèmes en prenant en considération la géométrie de l'espace postérieur lors de son exploration (i.e., lorsque l'algorithme décide où il doit aller ensuite). Cet algorithme converge beaucoup plus rapidement, et donc moins d'échantillons seront nécessaires pour approcher la distribution postérieure.

Le résultat d'une inférence bayésienne est donc, en pratique, un ensemble d'échantillons obtenus en utilisant des MCMCs. La fiabilité des ces estimations doit être évaluée en vérifiant (visuellement et numériquement) que les MCMCs ont bien convergé vers une solution optimale.

Travaux pratiques

On s'intéresse à la performance économique des capitales `rgdppc_2000` en fonction de deux paramètres : la rudesse du paysage (plus ou moins vallonné) `rugged` et son appartenance au continent africain `cont_africa`.

```r
library(tidyverse)
library(imsb)

d <- open_data(rugged) %>% mutate(log_gdp = log(rgdppc_2000) )
df1 <- d[complete.cases(d$rgdppc_2000), ]
str(df1)
```

```
'data.frame':   170 obs. of  6 variables:
 $ isocode    : chr  "AGO" "ALB" "ARE" "ARG" ...
 $ country    : chr  "Angola" "Albania" "United Arab Emirates" "Argentina" ...
 $ rugged     : num  0.858 3.427 0.769 0.775 2.688 ...
 $ cont_africa: int  1 0 0 0 0 0 0 0 0 1 ...
 $ rgdppc_2000: num  1795 3703 20604 12174 2422 ...
 $ log_gdp    : num  7.49 8.22 9.93 9.41 7.79 ...
```

Travaux pratiques

Écrire le modèle qui prédit `log_gdp` en fonction de la rudesse du terrain, du continent, et de l'interaction de ces deux variables avec `brms::brm()`, en spécifiant vos propres priors.

$$
\begin{aligned}
\log(\text{gdp}_i) &\sim \text{Normal}(\mu_i, \sigma_i) \\
\mu_i &= \alpha + \beta_1 \times \text{rugged}_i + \beta_2 \times \text{continent}_i + \beta_3 \times (\text{rugged}_i \times \text{continent}_i) \\
\alpha &\sim \dots \\
\beta_1, \beta_2, \beta_3 &\sim \dots
\end{aligned}
$$

Examinez ensuite les estimations de ce modèle (interprétation des paramètres, diagnostiques des MCMCs). Puis, essayez de complexifier le modèle, voire de le rendre incohérent, afin d'identifier des problèmes liés aux MCMCs.

Proposition de réponse

```r
priors2 <- c(
  prior(normal(0, 100), class = Intercept),
  prior(normal(0, 10), class = b),
  prior(exponential(0.01), class = sigma)
  )

mod2 <- brm(
  formula = log_gdp ~ 1 + rugged * cont_africa,
  prior = priors2,
  family = gaussian(),
  data = df1,
  chains = 4, # nombre de chaînes
  cores = 4, # nombre de coeurs parallèles
  warmup = 1000, # nombre d'itérations pour le warm-up
  iter = 2000 # nombre total d'itérations (par chaîne)
  )
```

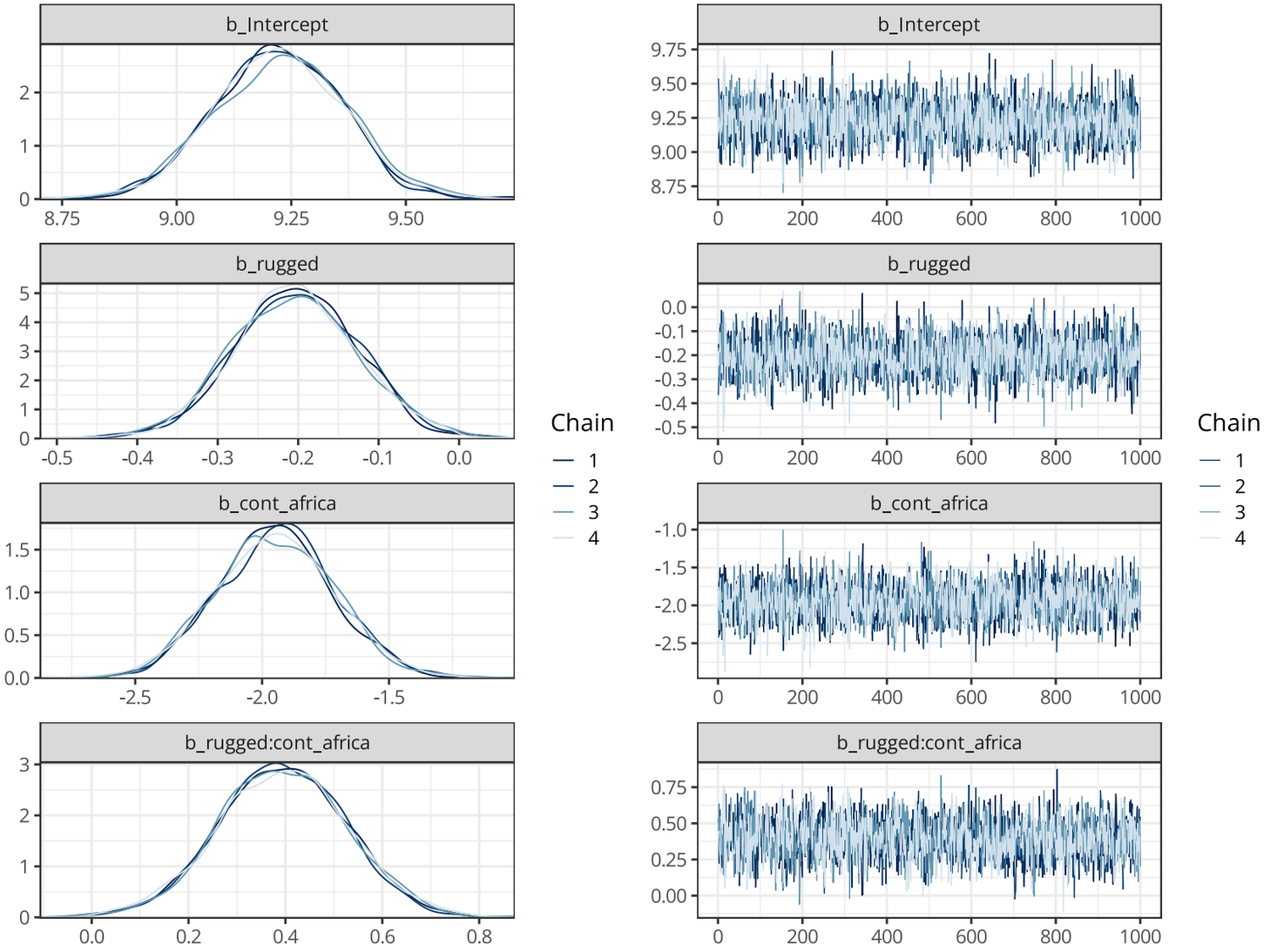
Proposition de réponse

```
1 summary(mod2)
```

```
 Family: gaussian
  Links: mu = identity
Formula: log_gdp ~ 1 + rugged * cont_africa
   Data: df1 (Number of observations: 170)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
                  Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept             9.22      0.14     8.95     9.50 1.00     2475     2837
rugged               -0.20      0.08    -0.36    -0.05 1.00     2385     2804
cont_africa          -1.95      0.23    -2.38    -1.50 1.00     2321     2790
rugged:cont_africa    0.39      0.13     0.13     0.65 1.00     2178     2858

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     0.95      0.06     0.85     1.07 1.00     3768     2714

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

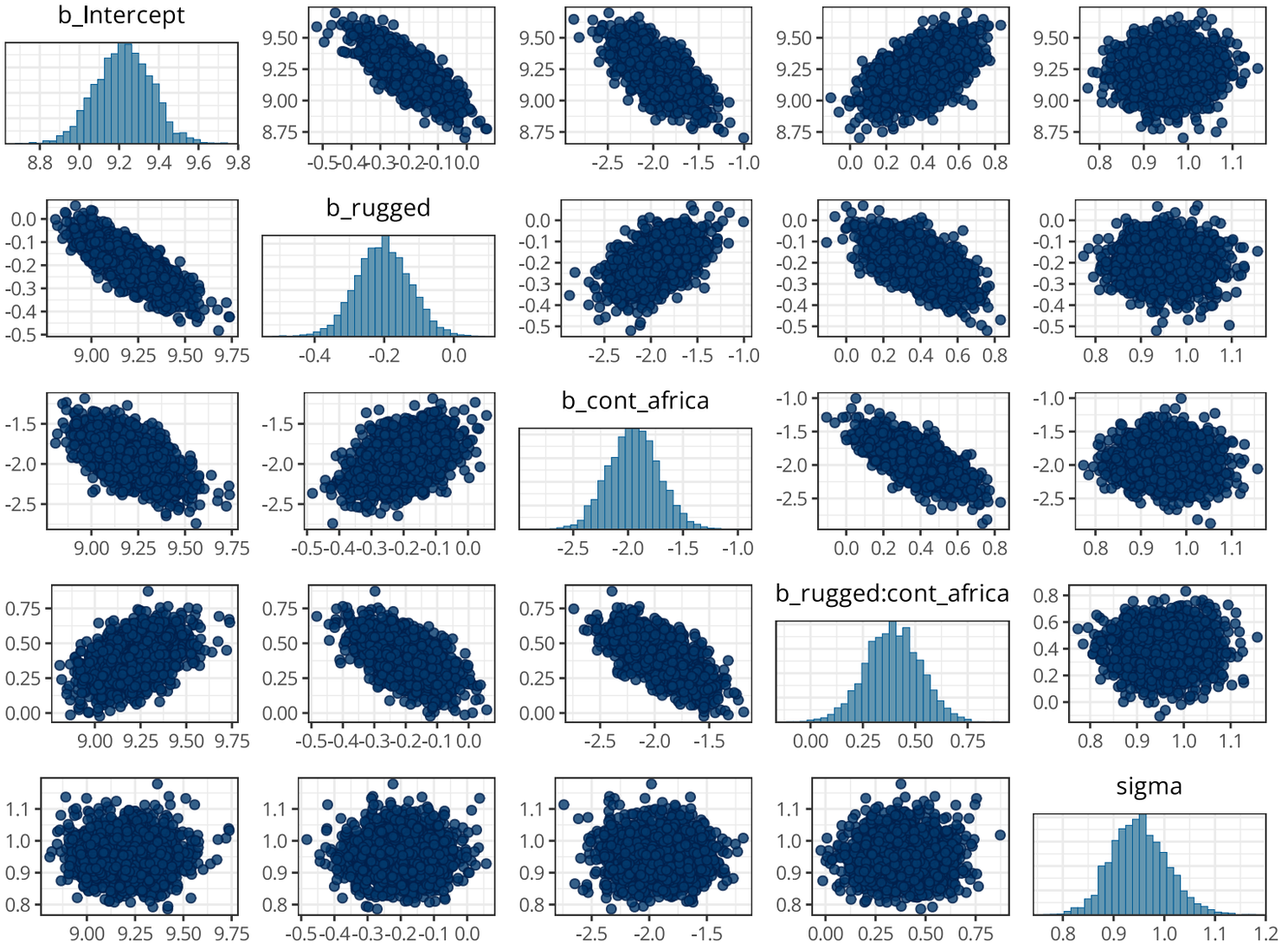
Proposition de réponse

```
1  plot(x = mod2, combo = c("dens_overlay", "trace"), pars = "^b_")
```



Proposition de réponse

```
1  pairs(x = mod2, np = nuts_params(mod2) ) # voir ?nuts_params
```



Proposition de réponse

```
mod3 <- brm(
  # modèle incohérent, interaction entre deux variables redondantes
  formula = log_gdp ~ 1 + log_gdp * rgdppc_2000,
  family = gaussian(),
  data = df1,
  chains = 2, # nombre de chaînes
  cores = 2, # nombre de coeurs parallèles
  warmup = 1000, # nombre d'itérations pour le warm-up
  iter = 2000 # nombre total d'itérations (par chaîne)
  )
```

Proposition de réponse

```
1  plot(x = mod3, combo = c("dens_overlay", "trace"), pars = "^b_")
```