

Introduction à la modélisation statistique bayésienne

Ladislas Nalborczyk
LPC, LNC, CNRS, 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

Cours n°09 : Modèles multi-niveaux généralisés

Cours n°10 : Data Hackathon



Le problème avec la distribution postérieure

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

Le problème avec la distribution postérieure

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

Petit problème : $p(\text{data})$ 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.

Le problème avec la distribution postérieure

Rappels cours n°02

Trois méthodes pour résoudre (contourner) ce problème :



Rappels cours n°02

Trois méthodes pour résoudre (contourner) ce problème :



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 (qu'on peut calculer de manière exacte) pour la distribution postérieure.



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 (qu'on peut calculer de manière exacte) pour la distribution postérieure.



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 (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.



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 (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.



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 (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

Objectifs du cours

- Présenter le principe de base de l'échantillonnage : Markov Chain Monte Carlo
- Présenter les algorithmes classiques



Objectifs du cours

- Présenter le principe de base de l'échantillonnage : Markov Chain Monte Carlo
- Présenter les algorithmes classiques
- Montrer les forces mais aussi les faiblesses de ces méthodes



Objectifs du cours

- Présenter le principe de base de l'échantillonnage : Markov Chain Monte Carlo
- Présenter les algorithmes classiques
- Montrer les forces mais aussi les faiblesses de ces méthodes
- Donner des outils de contrôle sur ces méthodes



Objectifs du cours

- Présenter le principe de base de l'échantillonnage : Markov Chain Monte Carlo
- Présenter les algorithmes classiques
- 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

- Markov chain **Monte Carlo**
 - Échantillonnage aléatoire
 - Le résultat est un ensemble de valeurs du paramètre

Markov Chain Monte Carlo

- Markov chain **Monte Carlo**
 - Échantillonnage aléatoire
 - Le résultat est un ensemble de valeurs du paramètre

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 : $\theta^1, \theta^2, \theta^3, \dots, \theta^t$



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 : $\theta^1, \theta^2, \theta^3, \dots, \theta^t$



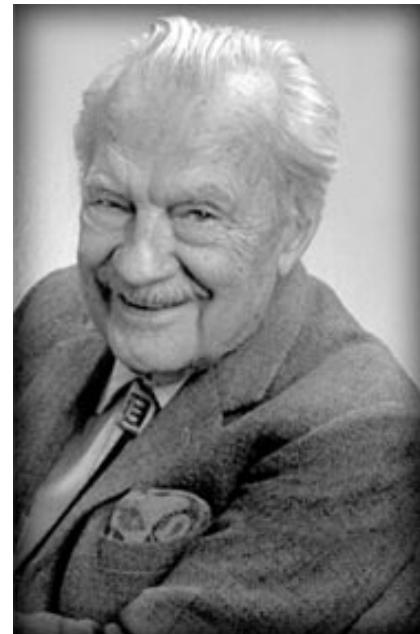
Markov Chain Monte Carlo

- 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 : $\theta^1, \theta^2, \theta^3, \dots, \theta^t$
- Markov chain Monte Carlo
 - La valeur de paramètre générée ne dépend que de la valeur du paramètre précédent
 - $P(\theta^{t+1} | \theta^t, \theta^{t-1}, \dots, \theta^1) = P(\theta^{t+1} | \theta^t)$



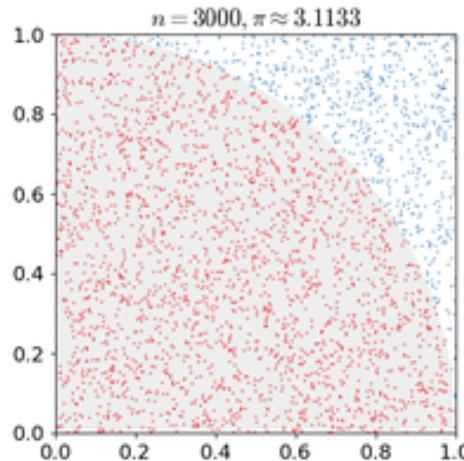
Méthodes Monte Carlo

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.



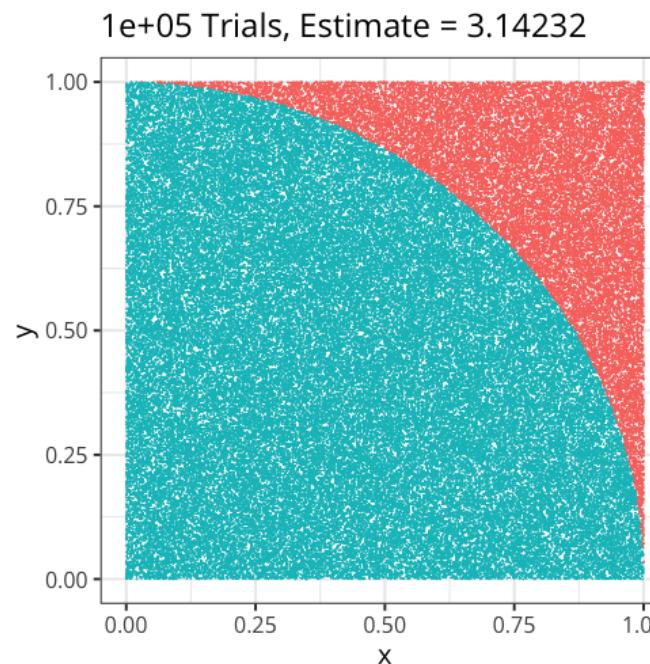
Méthodes Monte Carlo : Estimation de π

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 $x^2 + y^2 \leq 1$. La probabilité que le point M appartienne au disque est $\pi/4$ puisque le quart de disque est de surface $\sigma = \frac{\pi R^2}{4} = \frac{\pi}{4}$, et 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 de tomber dans le quart de disque vaut $\frac{\sigma}{S} = \frac{\pi}{4}$. En faisant le rapport du nombre de points dans le disque au nombre de tirages, on obtient une approximation du nombre $\pi/4$ si le nombre de tirages est grand.



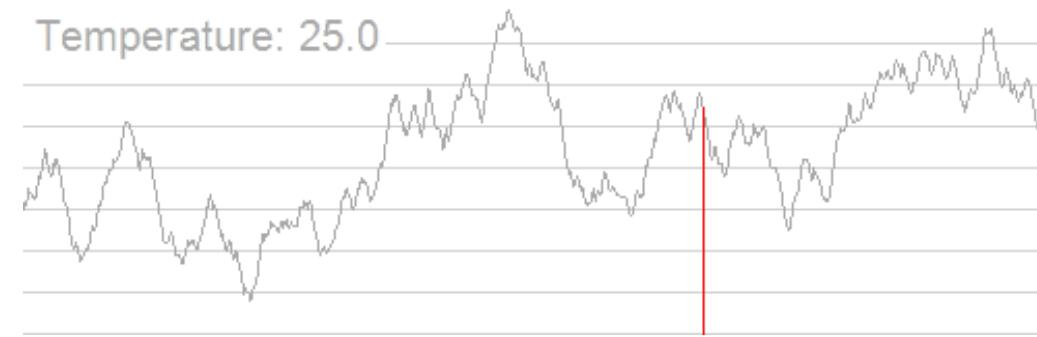
Méthodes Monte Carlo : Estimation de π

```
trials <- 1e5
radius <- 1
x <- runif(n = trials, min = 0, max = radius)
y <- runif(n = trials, min = 0, max = radius)
distance <- sqrt(x^2 + y^2)
inside <- distance < radius
pi_estimate <- 4 * sum(inside) / trials
```



Méthodes Monte Carlo

Autre exemple : déterminer la superficie d'un lac ou encore déterminer le maximum d'une fonction (optimisation) via *simulated annealing* ou *recuit simulé* (https://en.wikipedia.org/wiki/Simulated_annealing).



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 ?



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 $p(\text{data}|\theta_1, \theta_2)$



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 $p(\text{data}|\theta_1, \theta_2)$

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



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 $p(\text{data}|\theta_1, \theta_2)$

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

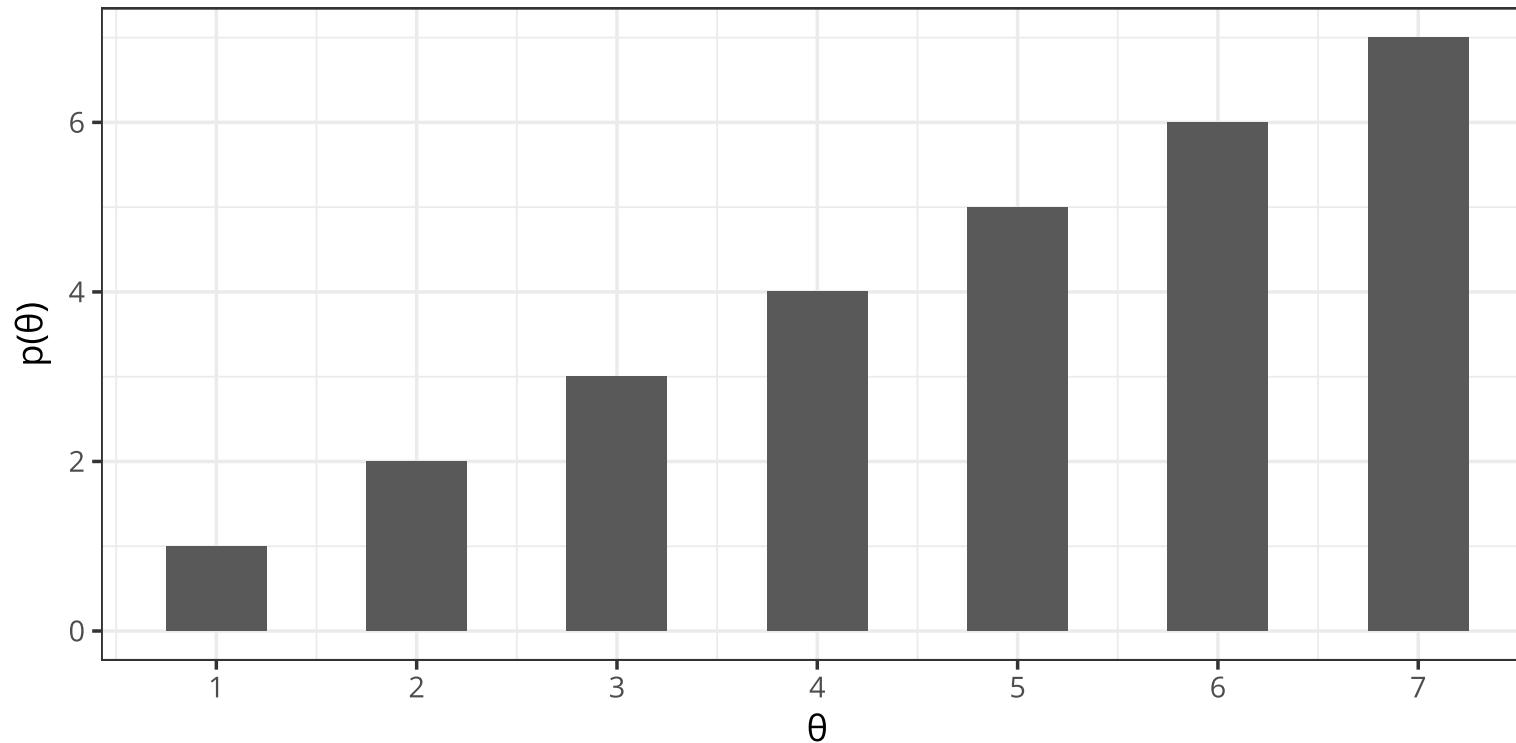
Où plutôt on ne sait pas calculer $p(\text{data})$... ! Mais on sait calculer la distribution postérieure à une constante près. Or, comme $p(\text{data})$ est une constante, elle ne change pas la forme de la distribution postérieure... ! On va donc explorer l'espace des paramètres et produire des échantillons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



Influence de la constante de normalisation

Méthodes Monte Carlo : Exemple

Considérons un exemple simple : Soit un paramètre θ avec 7 valeurs possibles et la fonction de répartition suivante.

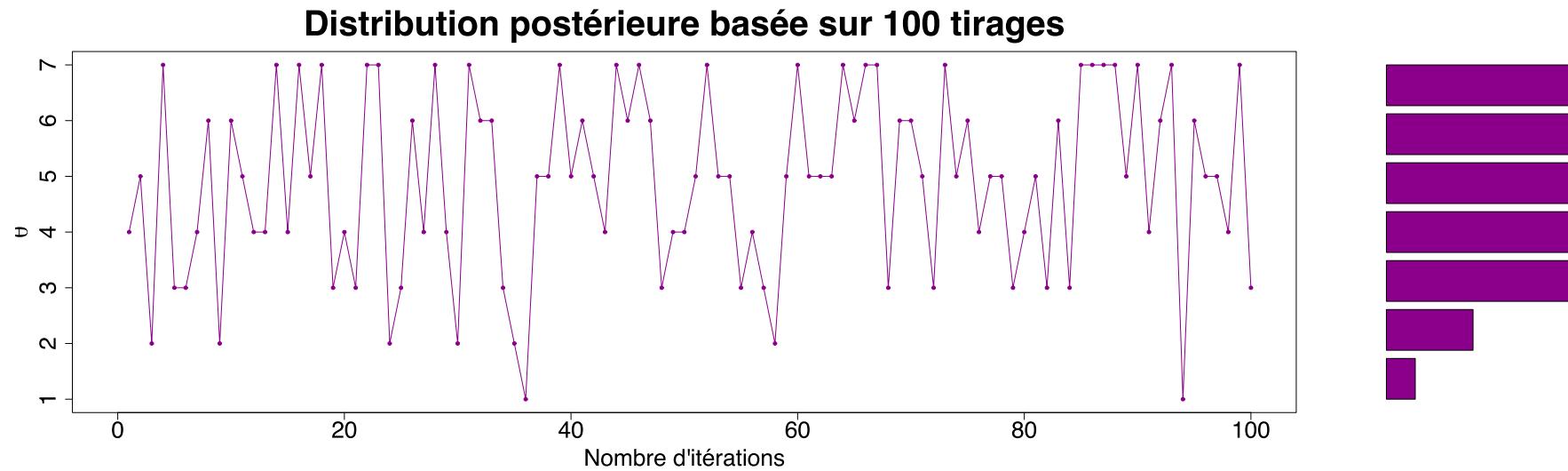


Méthodes Monte Carlo : Exemple

Estimation 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) !

4	4	4	4	3	3	3
5	5	5	5	5	2	2
6	6	6	6	6	6	1
7	7	7	7	7	7	7

Méthodes Monte Carlo : Exemple



- La distribution des échantillons obtenus converge vers la “vraie” distribution
- Mais, nécessite généralement beaucoup d'échantillons (ici pour 7 valeurs de paramètres et 100 tirages)
- 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.**



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



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



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

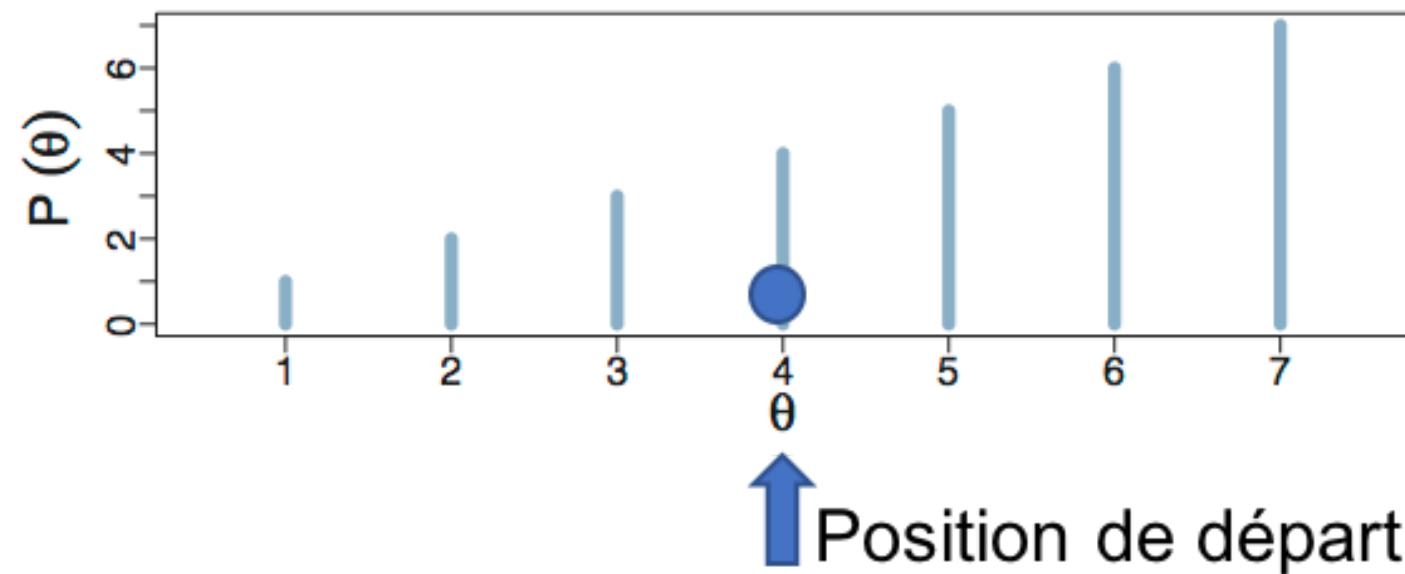
Deux idées centrales

- 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*)



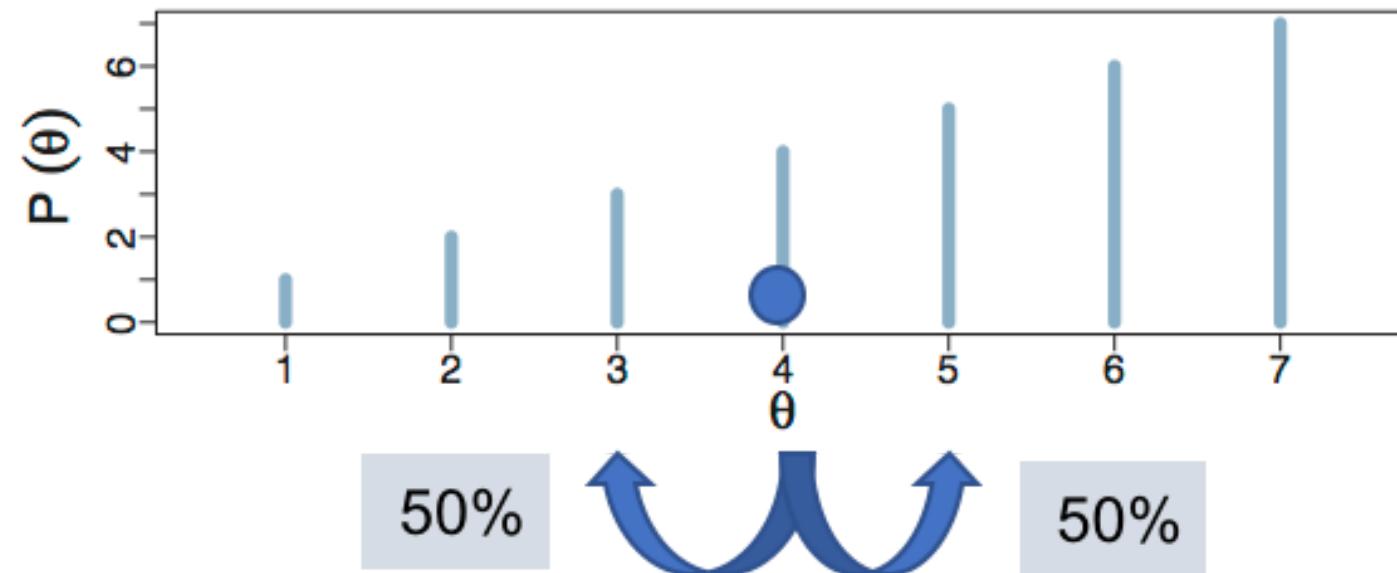
Algorithme Metropolis

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



Algorithme Metropolis

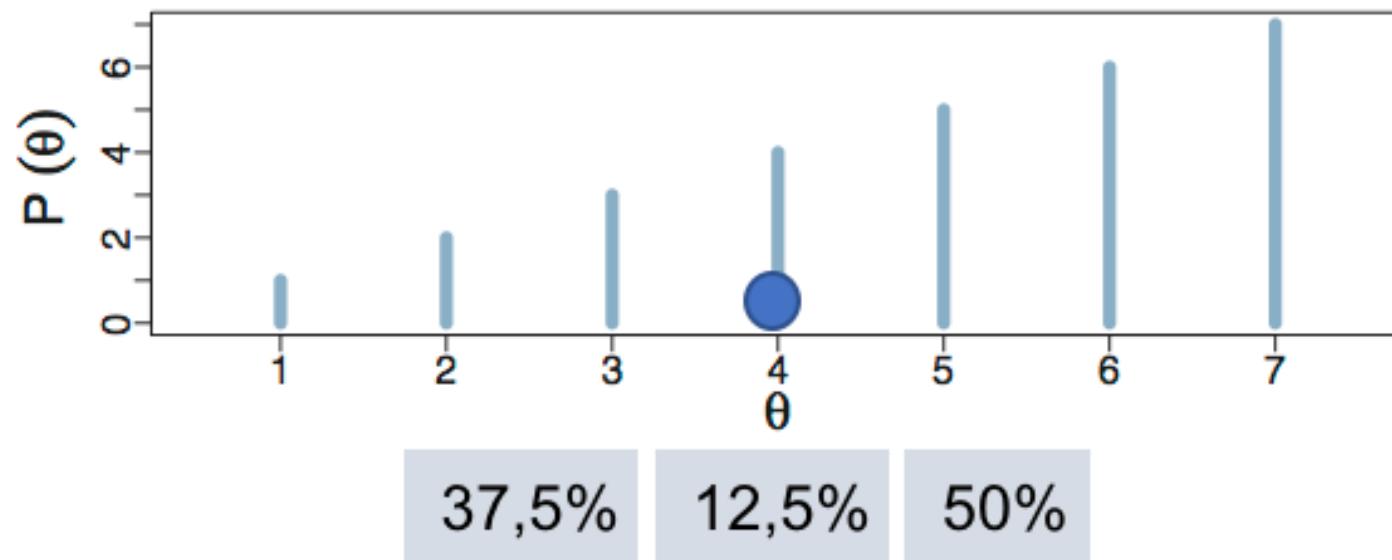
2) Faire une proposition de déplacement centrée sur la valeur courante de θ .



Algorithme Metropolis

3) Calculer la **probabilité** d'accepter le déplacement :

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

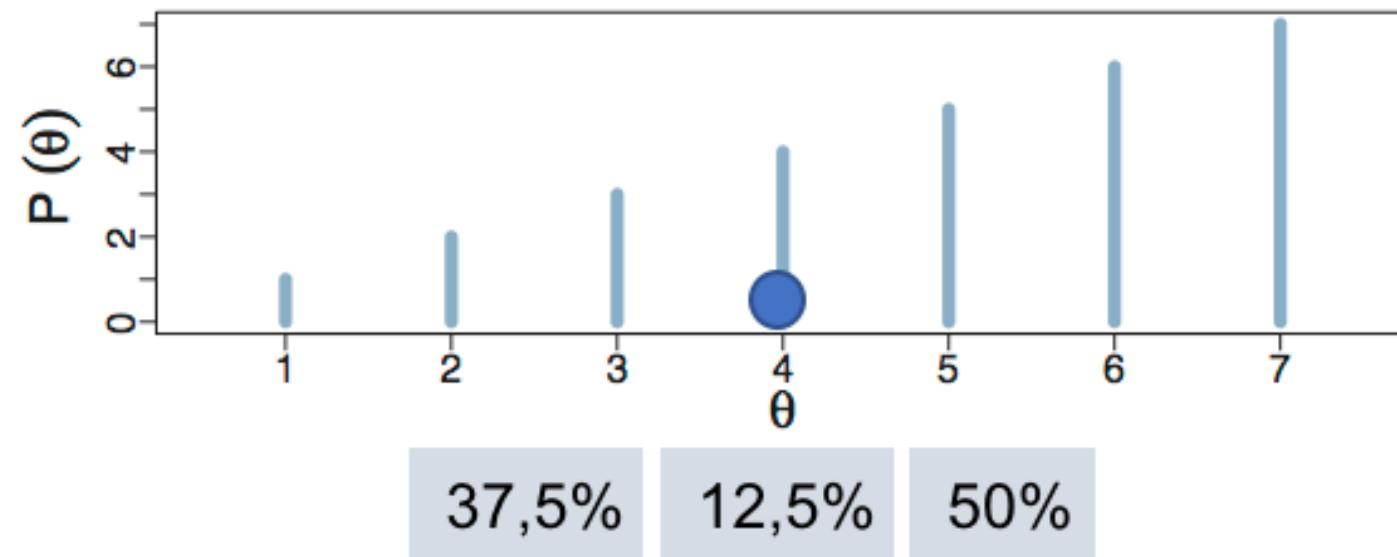


Algorithme Metropolis

4) Accepter ou rejeter la proposition de déplacement. On compare le ratio de probabilités \Pr_{move} à un tirage uniforme. On génère une valeur u suivant $u \sim \text{Uniform}(0, 1)$. Ensuite :

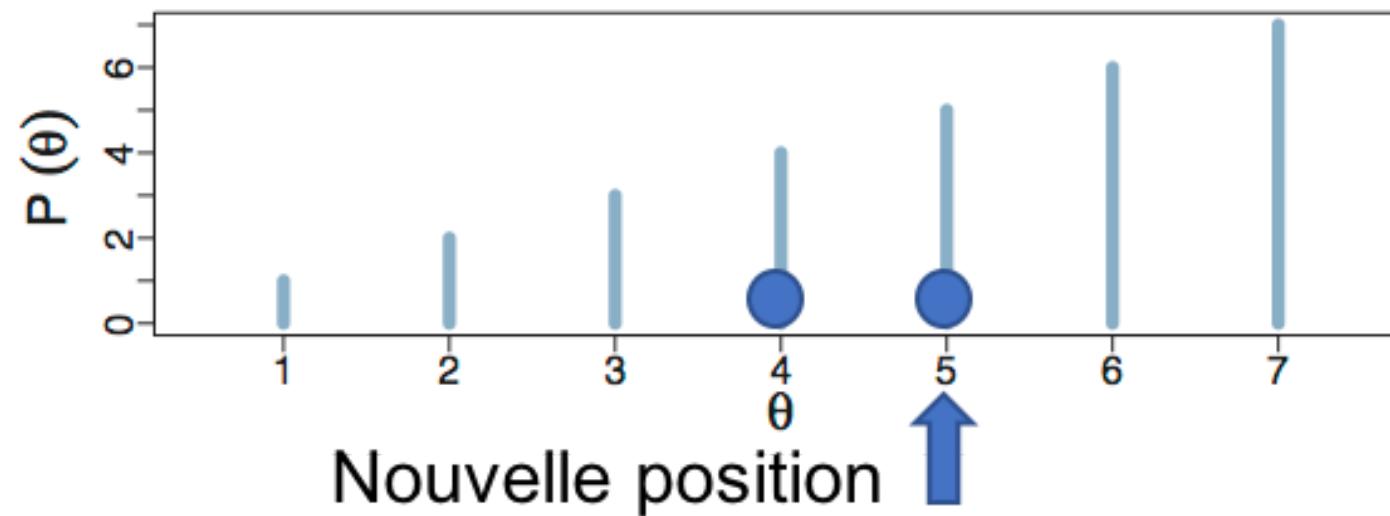
Si $u \leq \Pr_{\text{move}}$, Alors on accepte le déplacement

Si $u > \Pr_{\text{move}}$, Alors on rejette le déplacement

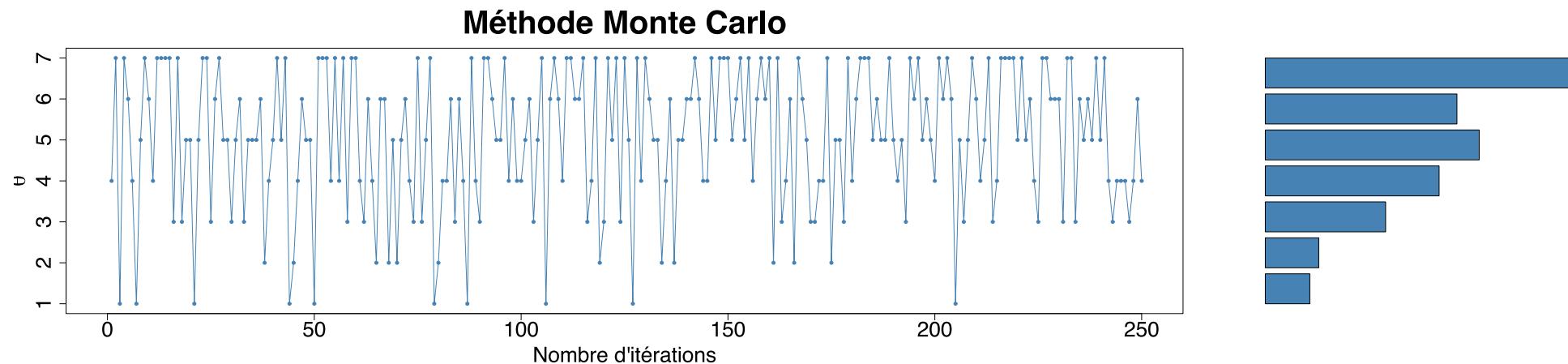


Algorithme Metropolis

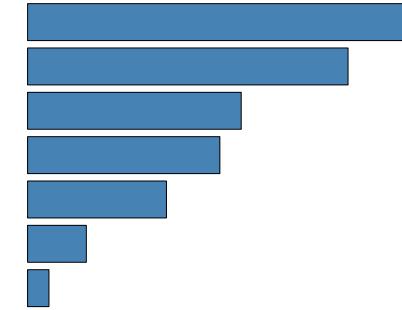
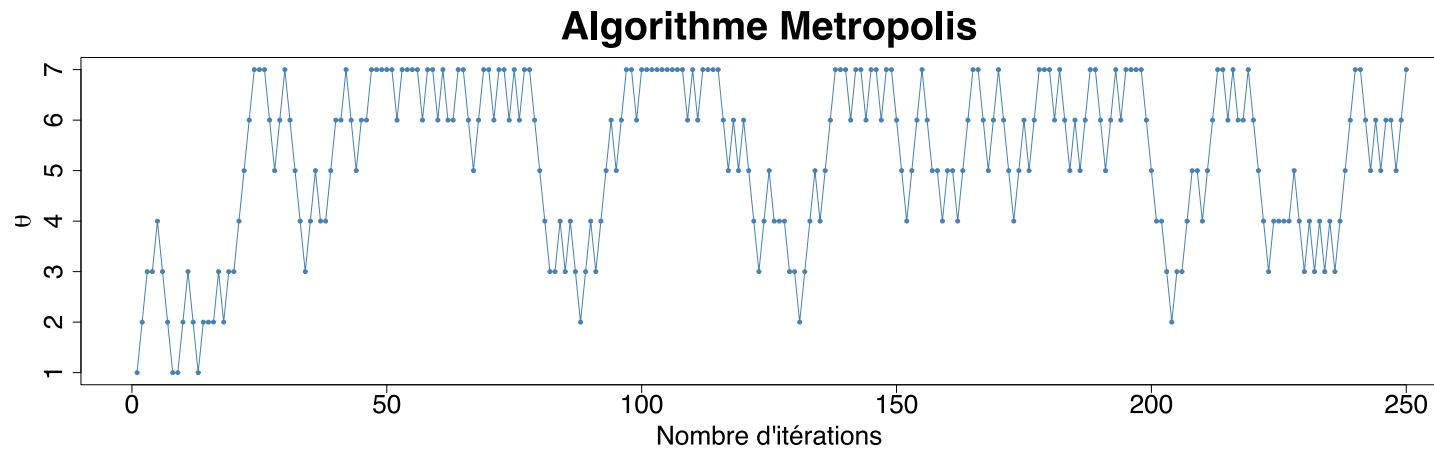
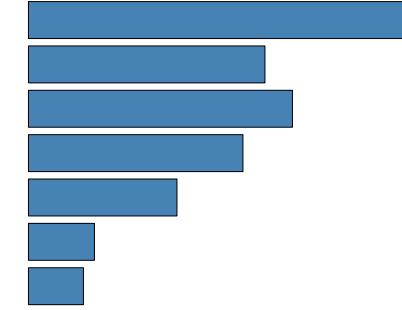
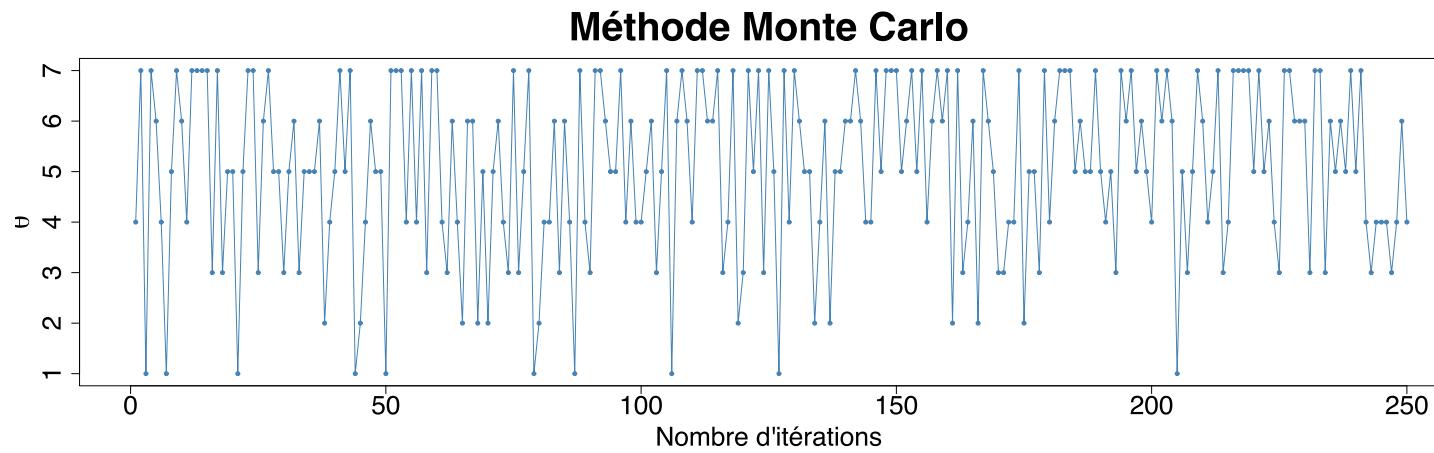
5) La position calculée devient la nouvelle position de départ et on répète l'algorithme.



Méthodes Monte Carlo vs. Algorithme Metropolis



Méthodes Monte Carlo vs. Algorithme Metropolis



Algorithme Metropolis

Application au lancer de pièce (cas continu)



Algorithme Metropolis

Application au lancer de pièce (cas continu)

- La fonction de vraisemblance est donnée par : $P(z \mid \theta, N) = \theta^z (1 - \theta)^{(N-z)}$
- 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 $[0, 1]$



Algorithme Metropolis

Application au lancer de pièce (cas continu)

- La fonction de vraisemblance est donnée par : $P(z | \theta, N) = \theta^z (1 - \theta)^{(N-z)}$
- Le prior est donné par : $P(\theta | a, b) \propto \theta^{(a-1)} (1 - \theta)^{(b-1)}$
- 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 ?



Algorithme Metropolis

Application au lancer de pièce (cas continu)

- La fonction de vraisemblance est donnée par : $P(z | \theta, N) = \theta^z (1 - \theta)^{N-z}$
- Le prior est donné par : $P(\theta | a, b) \propto \theta^{(a-1)} (1 - \theta)^{(b-1)}$
- 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 ?

Le déplacement est modélisé par une distribution normale : $\Delta\theta \sim \text{Normal}(0, \sigma)$

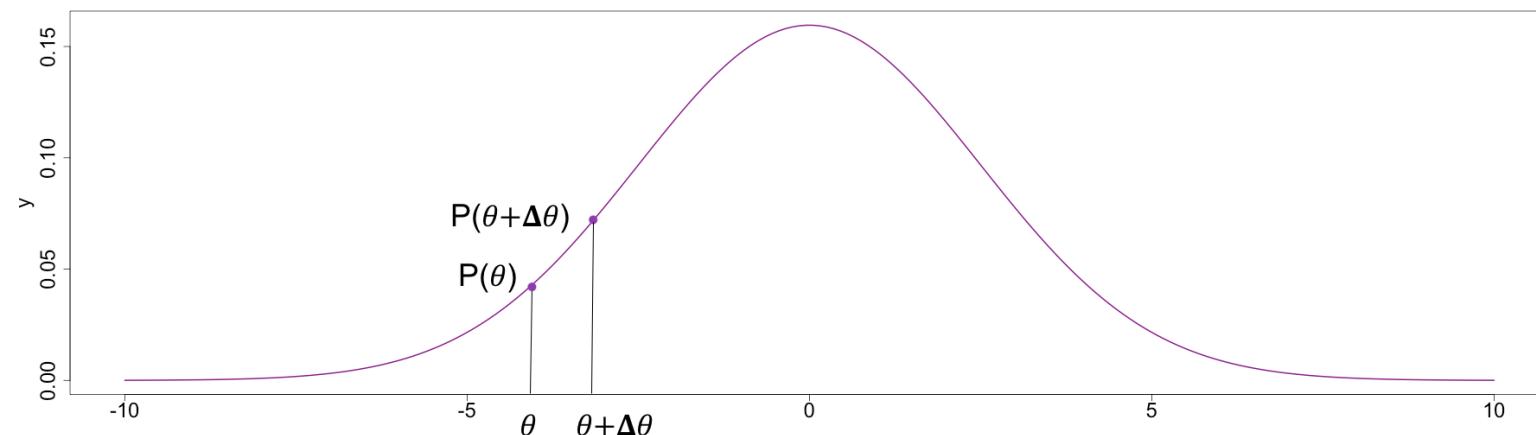
- La moyenne μ vaut 0 : le déplacement se fait autour de la valeur courante du paramètre
- La variance reste à déterminer, elle contrôle l'éloignement de la nouvelle valeur



Algorithme Metropolis

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

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



REMARQUE : Le rapport $\frac{\text{Pr}(\theta_{\text{current}} + \Delta\theta)}{\text{Pr}(\theta_{\text{current}})}$ 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 $\Pr(\theta_{initial}) \neq 0$

Algorithme Metropolis

→ Sélectionner un point de départ

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

→ Choisir une direction de déplacement

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



Algorithme Metropolis

- Sélectionner un point de départ
 - Il faut choisir $\theta \in [0, 1]$
 - Seule contrainte $\Pr(\theta_{initial}) \neq 0$
- Choisir une direction de déplacement
 - Faire un tirage suivant $\text{Normal}(0, \sigma)$
- Accepter ou rejeter la proposition de déplacement, suivant la probabilité :



Algorithme Metropolis

→ Sélectionner un point de départ

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

→ Choisir une direction de déplacement

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

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

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



Algorithme Metropolis

→ Sélectionner un point de départ

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

→ Choisir une direction de déplacement

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

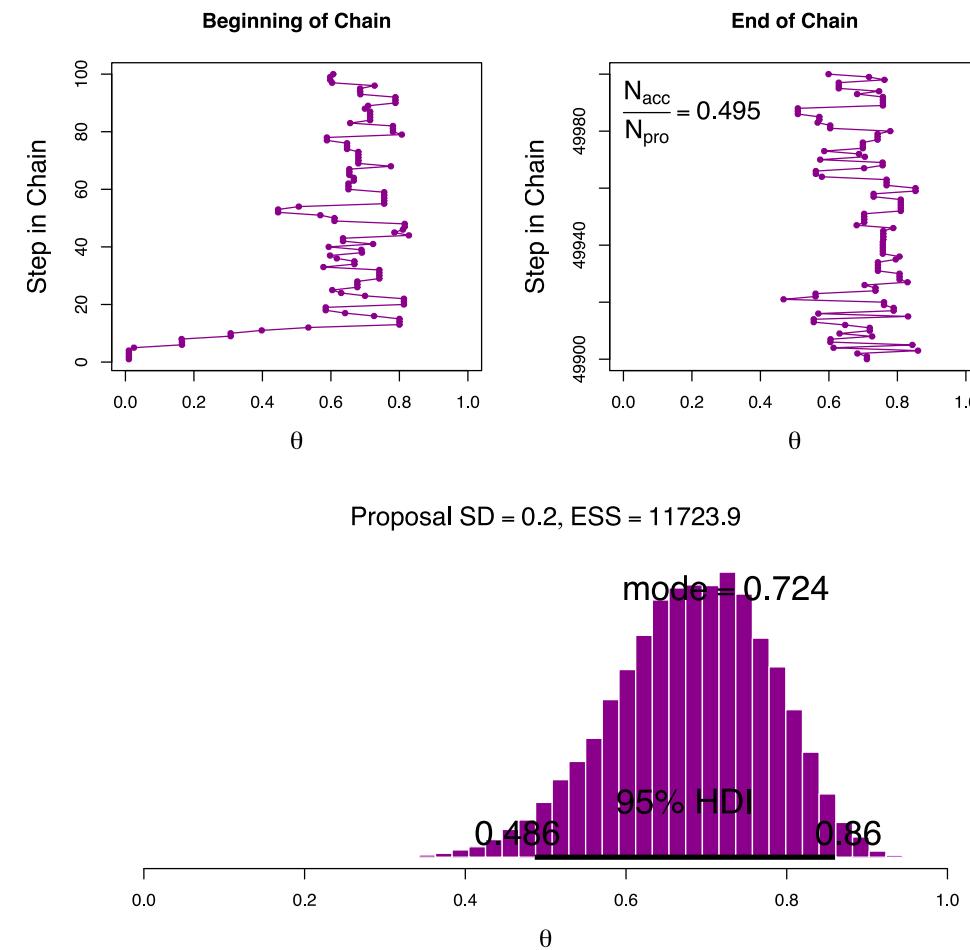
→ Accepter ou rejeter la proposition de déplacement, suivant la probabilité :

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

→ La position calculée devient la nouvelle position



Algorithme Metropolis



Algorithme Metropolis

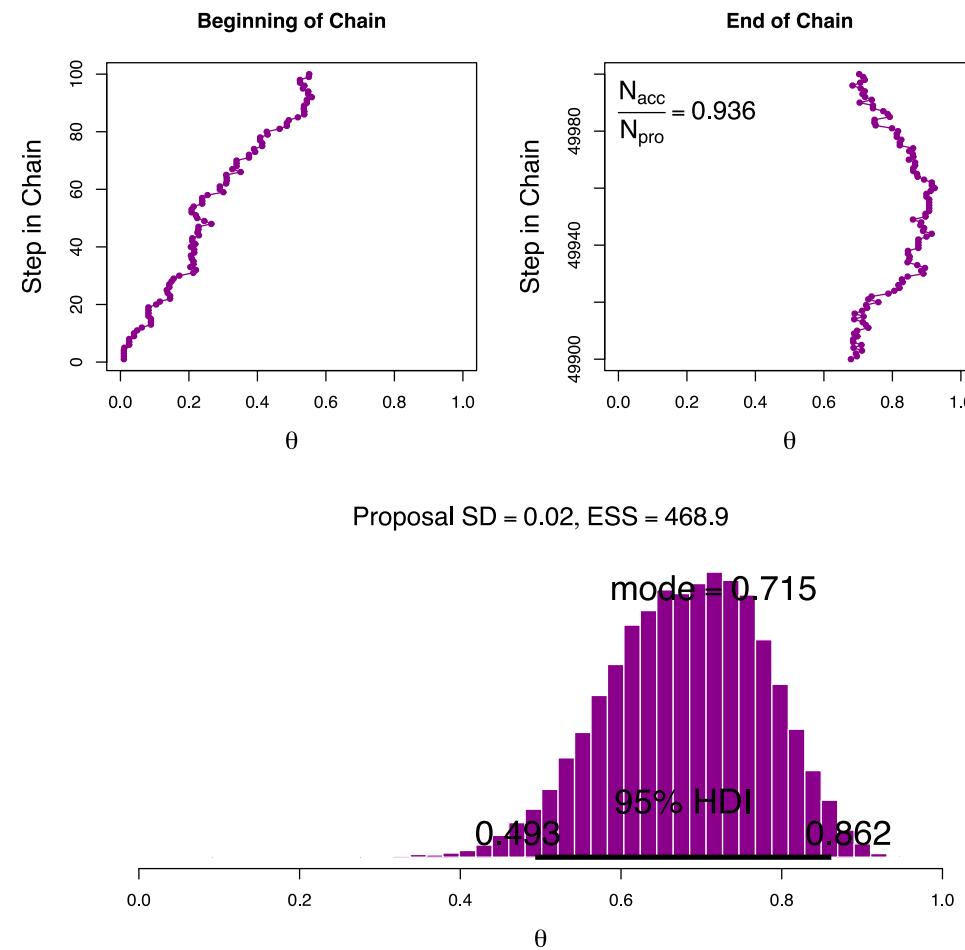
Le choix de sigma dans la proposition de déplacement

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



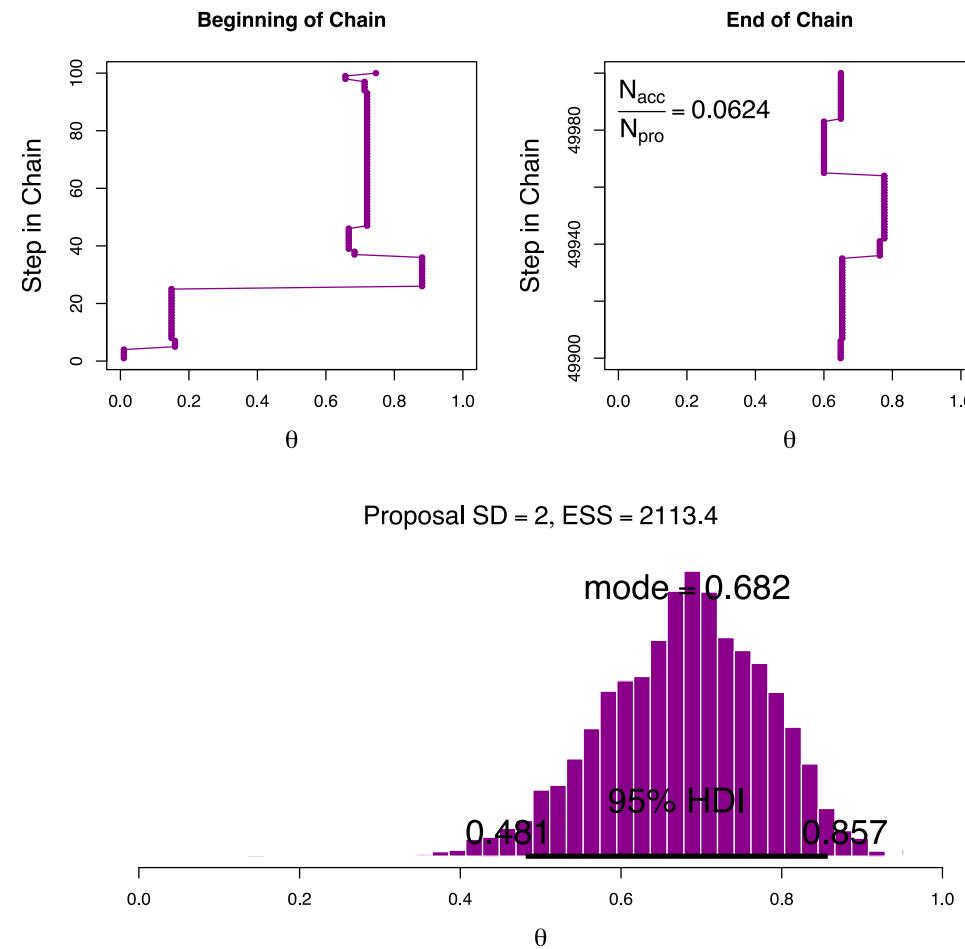
Algorithme Metropolis

Le choix de sigma dans la proposition de déplacement

- Toutes les propositions de déplacement (ou presque) sont acceptées
- Peu de valeurs effectives

Il va falloir beaucoup d'itérations pour avoir un résultat satisfaisant

Algorithme Metropolis



30



Algorithme Metropolis

Le choix de sigma dans la proposition de déplacement

- Les propositions de déplacement sont rarement acceptées
- Peu de valeurs effectives...

Il va falloir beaucoup d'itérations pour obtenir un résultat satisfaisant



Algorithme Metropolis-Hastings

```
target <- function(x) {
  # target distribution is Exponential(1)
  # implementation from https://stephens999.github.io/fiveMinuteStats/MH-examples1.html

  if (x < 0) {
    return(0)
  }
  else {
    return(exp(-x) )
  }
}

metropolis_hastings <- function(niter, startval, proposalsd) {

  x <- rep(0, niter) # initialises the chain vector
  x[1] <- startval # defines the starting value

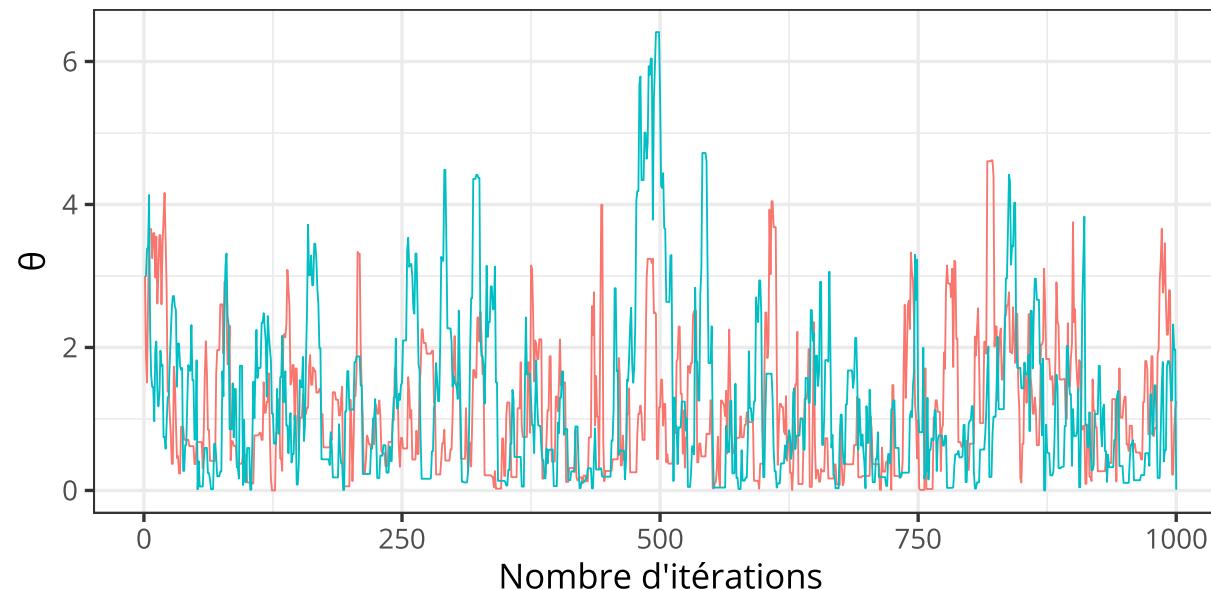
  for(i in 2:niter) {
    currentx <- x[i - 1] # retrieves current value of the parameter
    proposedx <- rnorm(1, mean = currentx, sd = proposalsd) # proposed move
    A <- target(proposedx) / target(currentx) # probability ratio
    if (runif(1) < A) {
      x[i] = proposedx # accept move with probability min(1, A)
    } else {
      x[i] = currentx # otherwise "reject" move, and stay where we are
    }
  }
  return(x)
}
```



Algorithme Metropolis-Hastings

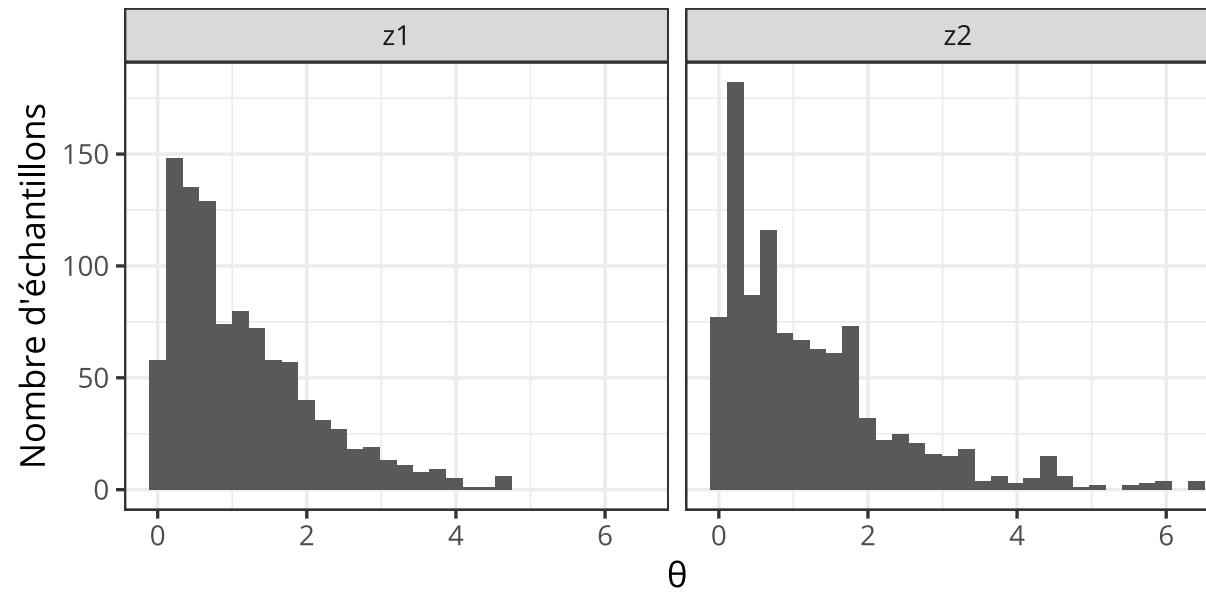
```
z1 <- metropolis_hastings(niter = 1000, startval = 3, proposalsd = 1)
z2 <- metropolis_hastings(niter = 1000, startval = 3, proposalsd = 1)

data.frame(z1 = z1, z2 = z2) %>%
  mutate(sample = 1:1000) %>%
  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))
```

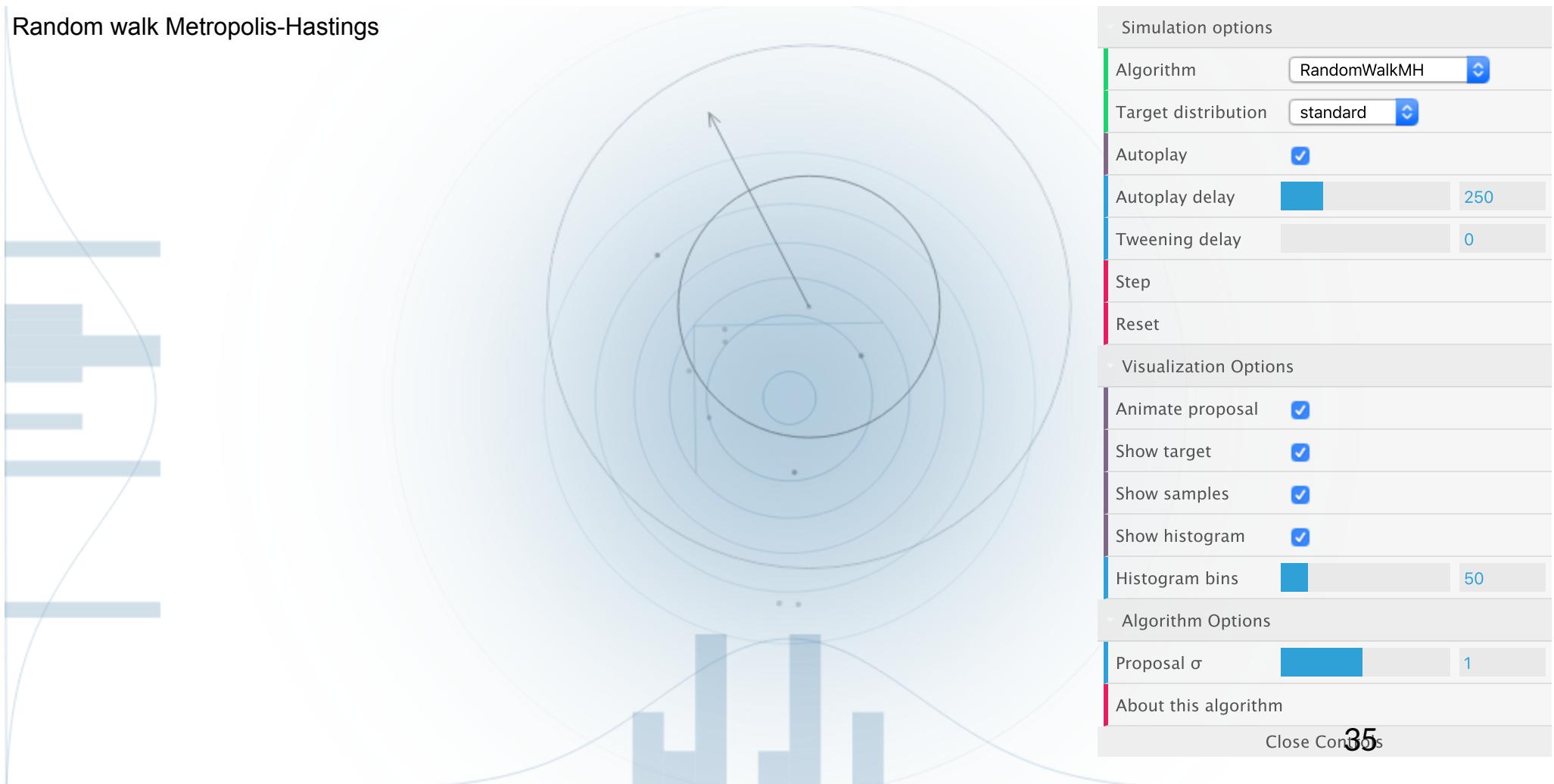


Algorithme Metropolis-Hastings

```
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() +  
  facet_wrap(~name) +  
  labs(x = expression(theta), y = "Nombre d'échantillons")
```



Algorithme Metropolis-Hastings



Algorithme Gibbs - Modèle bivarié

L'algorithme Gibbs est utilisé pour évaluer la distribution postérieure de modèles composés d'au moins deux variables et diffère de l'algorithme de Metropolis-Hastings en ce qu'il accepte toutes les propositions de déplacements...

Définition du modèle bivarié : On considère une situation dans laquelle on dispose de deux pièces de monnaie. À chaque pièce on associe un paramètre ou biais θ_1 et θ_2 (la probabilité d'obtenir Face). On cherche à estimer les biais de ces deux pièces : $p(\theta_1, \theta_2 | D)$. Nos variables θ_1 et θ_2 sont indépendantes et identiquement distribuées (iid).

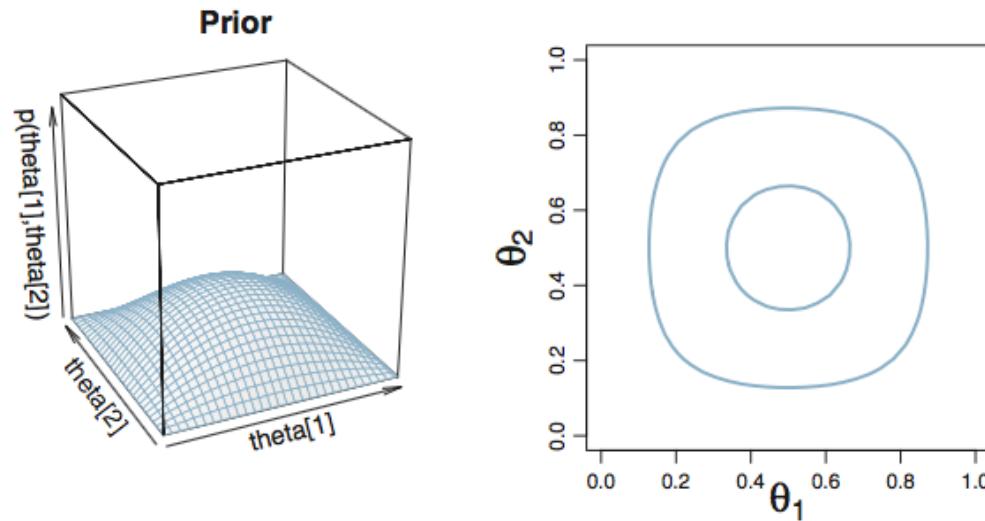
Définition du prior : On considère que chacun des biais est décrit par une distribution Beta : $\theta_1 \sim \text{Beta}(\theta_1 | a_1, b_1)$ et $\theta_2 \sim \text{Beta}(\theta_2 | a_2, b_2)$.



Algorithme Gibbs - Modèle bivarié

On peut donc calculer la distribution a priori de $P(\theta_1, \theta_2)$:

$$\begin{aligned} p(\theta_1, \theta_2) &= p(\theta_1)p(\theta_2) \\ &= \theta_1^{(a_1-1)} (1 - \theta_1)^{(b_1-1)} \theta_2^{(a_2-1)} (1 - \theta_2)^{(b_2-1)} \end{aligned}$$



Algorithme Gibbs - Modèle bivarié

Description des données : On dispose d'un ensemble de lancers pour les deux pièces. On appelle D_1 les données issues de la première pièce, composé de z_1 Face sur N_1 lancers. On appelle D_2 les données issues de la seconde pièce, composé de z_2 Face sur N_2 lancers.

On note les données : $D = \{z_1, N_1, z_2, N_2\}$

Définition de la fonction de vraisemblance

$$\begin{aligned} P(D \mid \theta_1, \theta_2) &= \prod_{y_{1i} \in D_1} P(y_{1i} \mid \theta_1, \theta_2) \prod_{y_{2i} \in D_2} P(y_{2i} \mid \theta_1, \theta_2) && \text{Règle d'indépendance} \\ &= \theta_1^{z_1} (1 - \theta_1)^{(N_1 - z_1)} \theta_2^{z_2} (1 - \theta_2)^{(N_2 - z_2)} \end{aligned}$$



Algorithme Gibbs - Modèle bivarié

Résolution analytique



Algorithme Gibbs - Modèle bivarié

Résolution analytique

$$\begin{aligned} p(\theta_1, \theta_2 \mid D) &= p(D \mid \theta_1, \theta_2) p(\theta_1, \theta_2) / p(D) \\ &= \theta_1^{z_1} (1 - \theta_1)^{(N_1 - z_1)} \theta_2^{z_2} (1 - \theta_2)^{(N_2 - z_2)} p(\theta_1, \theta_2) / p(D) \\ &= \frac{\theta_1^{(z_1 + a_1 - 1)} (1 - \theta_1)^{(N_1 - z_1 + b_1 - 1)} \theta_2^{(z_2 + a_2 - 1)} (1 - \theta_2)^{(N_2 - z_2 + b_2 - 1)}}{p(D) B(a_1, b_1) B(a_2, b_2)} \end{aligned}$$

Algorithme Gibbs - Modèle bivarié

Résolution analytique

$$\begin{aligned} p(\theta_1, \theta_2 \mid D) &= p(D \mid \theta_1, \theta_2) p(\theta_1, \theta_2) / p(D) \\ &= \theta_1^{z_1} (1 - \theta_1)^{(N_1 - z_1)} \theta_2^{z_2} (1 - \theta_2)^{(N_2 - z_2)} p(\theta_1, \theta_2) / p(D) \\ &= \frac{\theta_1^{(z_1 + a_1 - 1)} (1 - \theta_1)^{(N_1 - z_1 + b_1 - 1)} \theta_2^{(z_2 + a_2 - 1)} (1 - \theta_2)^{(N_2 - z_2 + b_2 - 1)}}{p(D) B(a_1, b_1) B(a_2, b_2)} \end{aligned}$$

Donc finalement :

Algorithme Gibbs - Modèle bivarié

Résolution analytique

$$\begin{aligned} p(\theta_1, \theta_2 | D) &= p(D | \theta_1, \theta_2) p(\theta_1, \theta_2) / p(D) \\ &= \theta_1^{z_1} (1 - \theta_1)^{(N_1 - z_1)} \theta_2^{z_2} (1 - \theta_2)^{(N_2 - z_2)} p(\theta_1, \theta_2) / p(D) \\ &= \frac{\theta_1^{(z_1 + a_1 - 1)} (1 - \theta_1)^{(N_1 - z_1 + b_1 - 1)} \theta_2^{(z_2 + a_2 - 1)} (1 - \theta_2)^{(N_2 - z_2 + b_2 - 1)}}{p(D) B(a_1, b_1) B(a_2, b_2)} \end{aligned}$$

Donc finalement :

$$p(\theta_1, \theta_2 | D) = \text{Beta}(\theta_1 | z_1 + a_1, N_1 - z_1 + b_1) \text{Beta}(\theta_2 | z_2 + a_2, N_2 - z_2 + b_2)$$

Algorithme Gibbs - Modèle bivarié

Résolution analytique

$$\begin{aligned} p(\theta_1, \theta_2 | D) &= p(D | \theta_1, \theta_2) p(\theta_1, \theta_2) / p(D) \\ &= \theta_1^{z_1} (1 - \theta_1)^{(N_1 - z_1)} \theta_2^{z_2} (1 - \theta_2)^{(N_2 - z_2)} p(\theta_1, \theta_2) / p(D) \\ &= \frac{\theta_1^{(z_1 + a_1 - 1)} (1 - \theta_1)^{(N_1 - z_1 + b_1 - 1)} \theta_2^{(z_2 + a_2 - 1)} (1 - \theta_2)^{(N_2 - z_2 + b_2 - 1)}}{p(D) B(a_1, b_1) B(a_2, b_2)} \end{aligned}$$

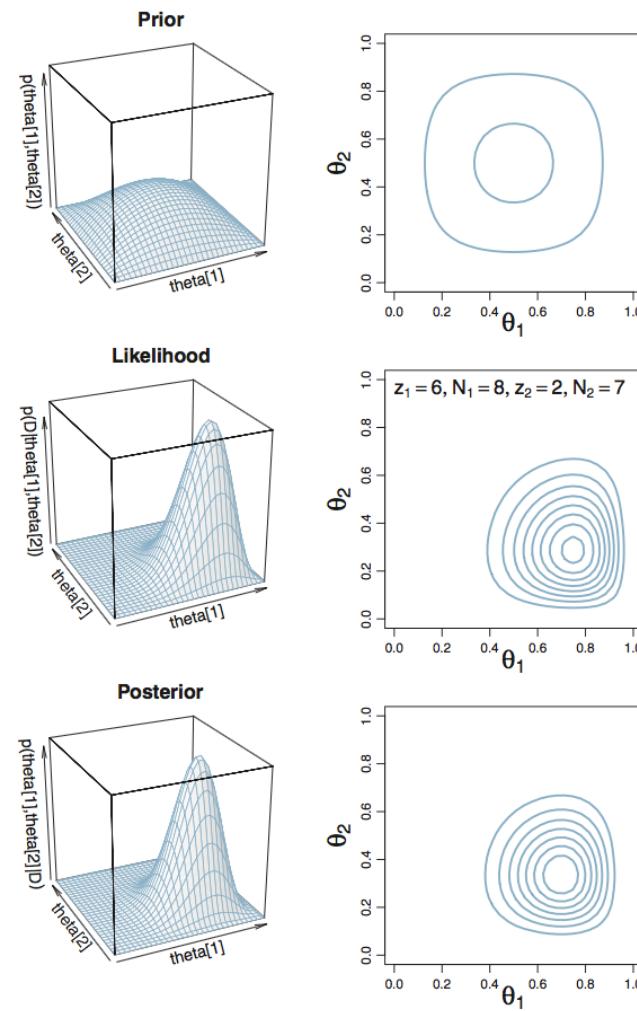
Donc finalement :

$$p(\theta_1, \theta_2 | D) = \text{Beta}(\theta_1 | z_1 + a_1, N_1 - z_1 + b_1) \text{Beta}(\theta_2 | z_2 + a_2, N_2 - z_2 + b_2)$$

On voit que lorsque le prior est le produit de deux distributions Beta indépendantes, alors la distribution postérieure est également le produit de deux distributions Beta indépendantes.



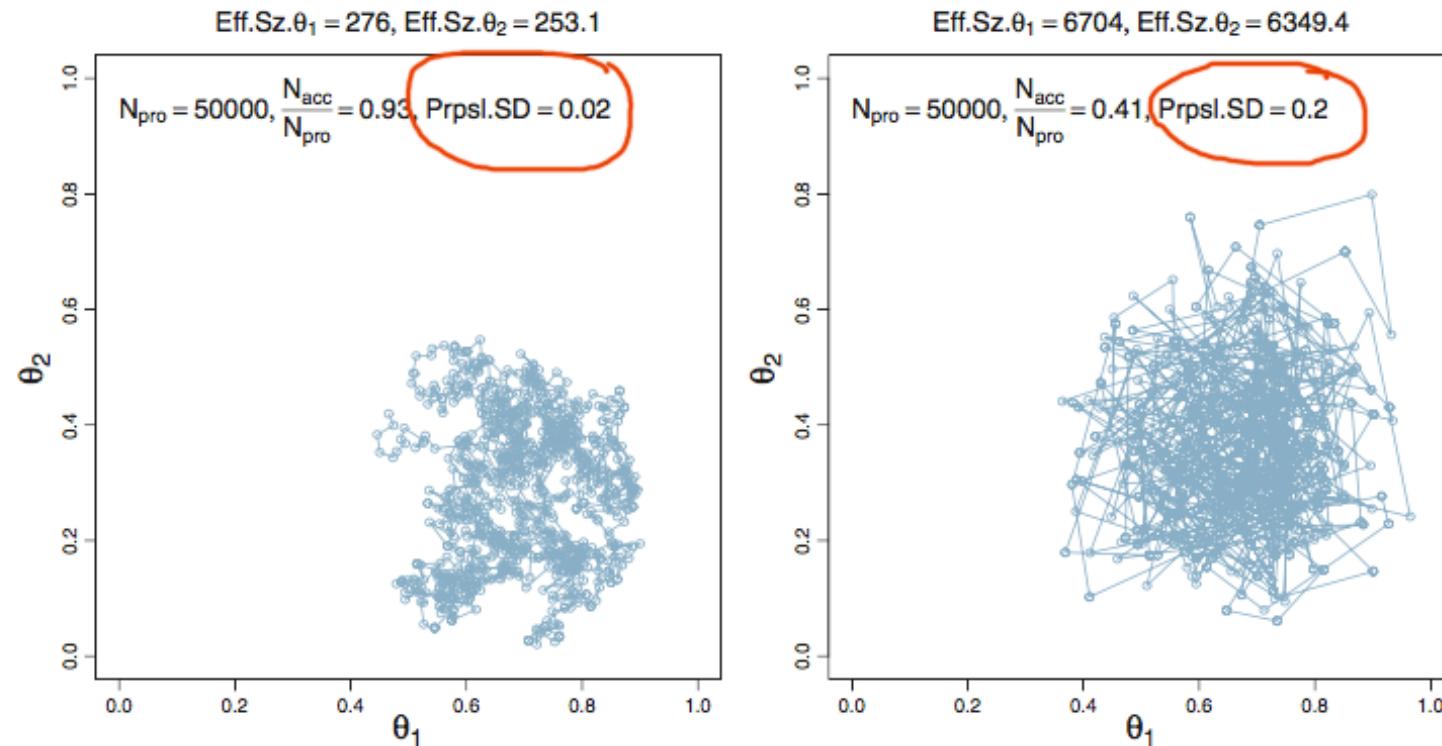
Algorithme Gibbs - Modèle bivarié



Modèle à deux variables - Algorithme Metropolis

- La proposition de saut repose sur une loi normale bivariée

→ La position suivante peut-être n'importe où sur la grille



Algorithme Gibbs - Description

On voit que l'efficacité de l'algorithme Metropolis repose sur la proposition de déplacement. Cette fonction peut-être difficile à calibrer :

- On voudrait que le SD soit petit dans les zones à forte densité
- Au contraire, on voudrait que le SD soit grand dans les zones à faible densité

L'algorithme Gibbs résout ce problème. L'algorithme de Gibbs parcourt l'espace des paramètres comme l'algorithme Metropolis, mais plutôt que de modifier tous les paramètres en même temps, il ne modifie qu'un paramètre après l'autre. La proposition de déplacement est donc basée sur la fonction de densité conditionnelle. Le déplacement est toujours accepté, mais l'algorithme requiert que l'on puisse calculer la densité postérieure conditionnelle...



Algorithme Gibbs - Description

→ Sélectionner un point de départ $(\theta_{1init}, \theta_{2init})$

On peut sélectionner n'importe quelle valeur

Algorithme Gibbs - Description

→ Sélectionner un point de départ $(\theta_{1init}, \theta_{2init})$

On peut sélectionner n'importe quelle valeur

→ Sélectionner le premier paramètre par exemple θ_1



Algorithme Gibbs - Description

→ Sélectionner un point de départ $(\theta_{1init}, \theta_{2init})$

On peut sélectionner n'importe quelle valeur

→ Sélectionner le premier paramètre par exemple θ_1

→ Générer une valeur aléatoire de θ_1

Faire un tirage directement suivant la probabilité $p(\theta_1 | \theta_2, D)$



Algorithme Gibbs - Description

→ Sélectionner un point de départ $(\theta_{1init}, \theta_{2init})$

On peut sélectionner n'importe quelle valeur

→ Sélectionner le premier paramètre par exemple θ_1

→ Générer une valeur aléatoire de θ_1

Faire un tirage directement suivant la probabilité $p(\theta_1 | \theta_2, D)$

→ Générer une valeur aléatoire de θ_2

Faire un tirage suivant la probabilité $p(\theta_2 | \theta_1, D)$

Avec pour valeur de θ_1 la valeur calculée précédemment



Algorithme Gibbs - Description

→ Sélectionner un point de départ $(\theta_{1init}, \theta_{2init})$

On peut sélectionner n'importe quelle valeur

→ Sélectionner le premier paramètre par exemple θ_1

→ Générer une valeur aléatoire de θ_1

Faire un tirage directement suivant la probabilité $p(\theta_1 | \theta_2, D)$

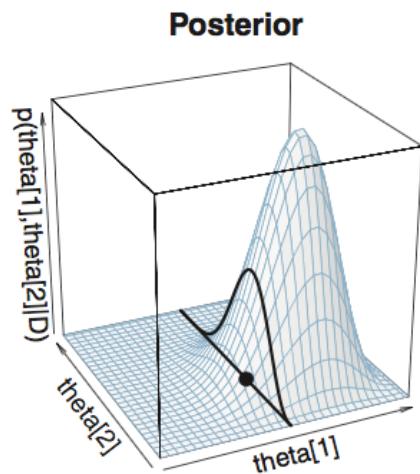
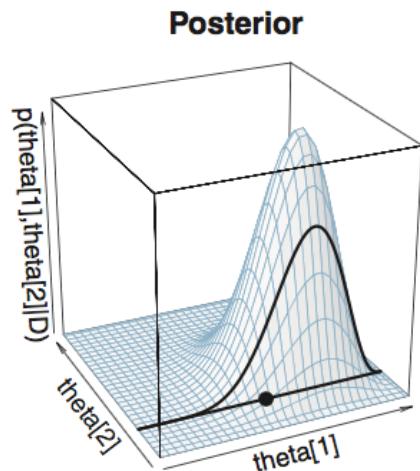
→ Générer une valeur aléatoire de θ_2

Faire un tirage suivant la probabilité $p(\theta_2 | \theta_1, D)$

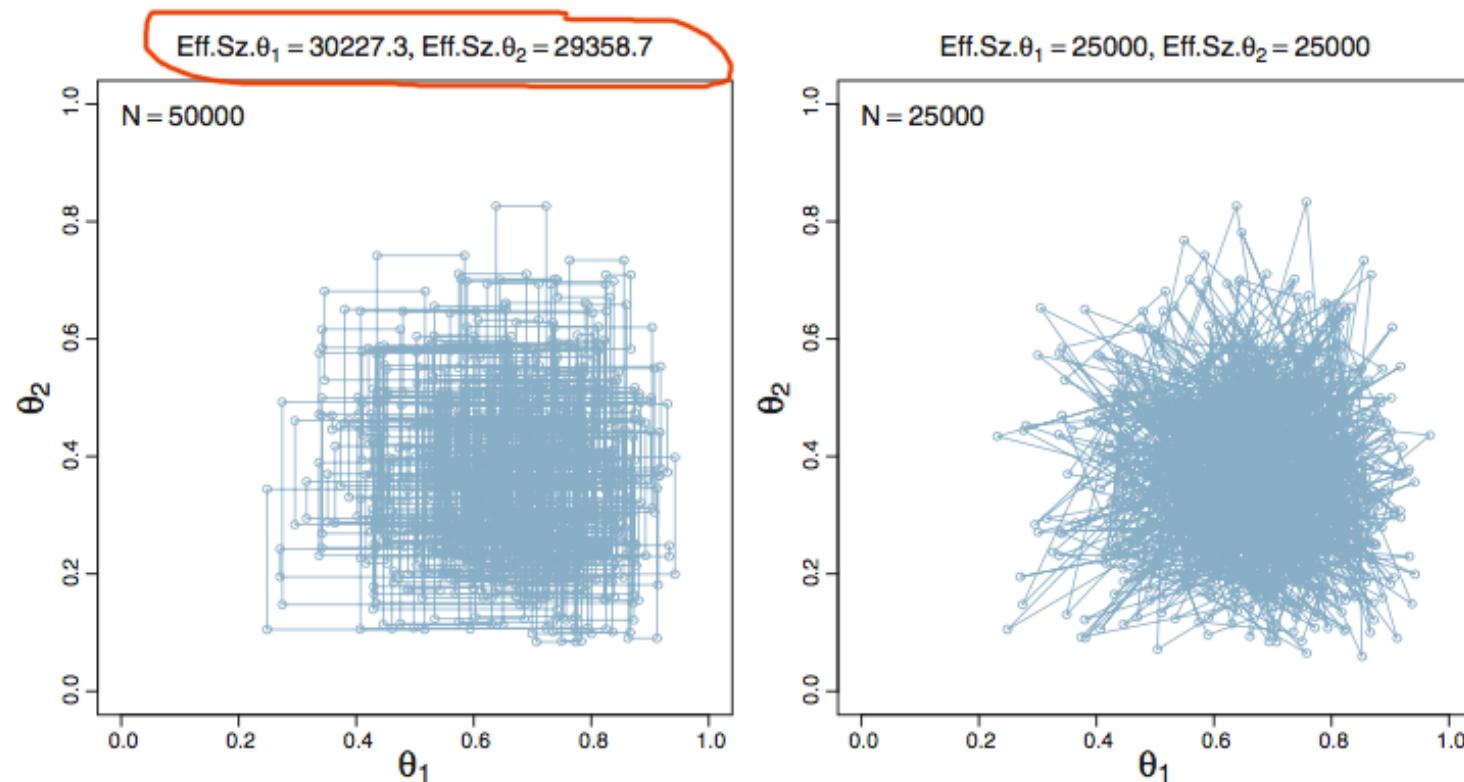
Avec pour valeur de θ_1 la valeur calculée précédemment

→ On boucle sur les deux dernières étapes pour parcourir

Algorithme Gibbs - Description



Algorithme Gibbs - Description



On voit que l'effective sample size est beaucoup plus grand que celui obtenu avec l'algorithme Metropolis...

Algorithme Gibbs - Implémentation

On suppose que $Y \sim \text{Normal}(\mu, \frac{1}{\tau})$. On souhaite approximer la distribution postérieure de μ et τ en utilisant l'algorithme Gibbs. On note que μ représente la moyenne dans la population, τ représente la précision (réciproque de la variance) dans la population, n représente la taille d'échantillon, \bar{y} représente la moyenne dans l'échantillon et s^2 la variance de l'échantillon. L'algorithme Gibbs peut être résumé de la manière suivante (from Casella & Georges, 1992) :



Algorithme Gibbs - Implémentation

On suppose que $Y \sim \text{Normal}(\mu, \frac{1}{\tau})$. On souhaite approximer la distribution postérieure de μ et τ en utilisant l'algorithme Gibbs. On note que μ représente la moyenne dans la population, τ représente la précision (réciproque de la variance) dans la population, n représente la taille d'échantillon, \bar{y} représente la moyenne dans l'échantillon et s^2 la variance de l'échantillon. L'algorithme Gibbs peut être résumé de la manière suivante (from Casella & Georges, 1992) :

- Échantillonner $\mu^{(i)}$ à partir de $p(\mu \mid \tau^{(i-1)}, \text{données})$



Algorithme Gibbs - Implémentation

On suppose que $Y \sim \text{Normal}(\mu, \frac{1}{\tau})$. On souhaite approximer la distribution postérieure de μ et τ en utilisant l'algorithme Gibbs. On note que μ représente la moyenne dans la population, τ représente la précision (réciproque de la variance) dans la population, n représente la taille d'échantillon, \bar{y} représente la moyenne dans l'échantillon et s^2 la variance de l'échantillon. L'algorithme Gibbs peut être résumé de la manière suivante (from Casella & Georges, 1992) :

- Échantillonner $\mu^{(i)}$ à partir de $p(\mu | \tau^{(i-1)}, \text{données})$
- Échantillonner $\tau^{(i)}$ à partir de $p(\tau | \mu^{(i)}, \text{données})$



Algorithme Gibbs - Implémentation

On suppose que $Y \sim \text{Normal}(\mu, \frac{1}{\tau})$. On souhaite approximer la distribution postérieure de μ et τ en utilisant l'algorithme Gibbs. On note que μ représente la moyenne dans la population, τ représente la précision (réciproque de la variance) dans la population, n représente la taille d'échantillon, \bar{y} représente la moyenne dans l'échantillon et s^2 la variance de l'échantillon. L'algorithme Gibbs peut être résumé de la manière suivante (from Casella & Georges, 1992) :

- Échantillonner $\mu^{(i)}$ à partir de $p(\mu | \tau^{(i-1)}, \text{données})$
- Échantillonner $\tau^{(i)}$ à partir de $p(\tau | \mu^{(i)}, \text{données})$

On définit les priors suivants : $p(\mu, \tau) = p(\mu) \times p(\tau)$ où $p(\mu) \propto 1$ et $p(\tau) \propto \tau^{-1}$.



Algorithme Gibbs - Implémentation

On suppose que $Y \sim \text{Normal}(\mu, \frac{1}{\tau})$. On souhaite approximer la distribution postérieure de μ et τ en utilisant l'algorithme Gibbs. On note que μ représente la moyenne dans la population, τ représente la précision (réciproque de la variance) dans la population, n représente la taille d'échantillon, \bar{y} représente la moyenne dans l'échantillon et s^2 la variance de l'échantillon. L'algorithme Gibbs peut être résumé de la manière suivante (from Casella & Georges, 1992) :

- Échantillonner $\mu^{(i)}$ à partir de $p(\mu | \tau^{(i-1)}, \text{données})$
- Échantillonner $\tau^{(i)}$ à partir de $p(\tau | \mu^{(i)}, \text{données})$

On définit les priors suivants : $p(\mu, \tau) = p(\mu) \times p(\tau)$ où $p(\mu) \propto 1$ et $p(\tau) \propto \tau^{-1}$.

Dans cette configuration la distribution postérieure conditionnelle pour la moyenne, sachant la précision, est donnée par $(\mu | \tau, \text{data}) \sim \text{Normal} \left(\bar{y}, \frac{1}{n\tau} \right)$. La distribution postérieure conditionnelle pour la précision, sachant la moyenne, est donnée par $(\tau | \mu, \text{data}) \sim \text{Gamma} \left(\frac{n}{2}, \frac{2}{(n-1)s^2 + n(\mu - \bar{y})^2} \right)$.



Algorithme Gibbs - Implémentation

```
# code from https://stats.stackexchange.com/questions/266665/gibbs-sampler-examples-in-r
n <- 30 # sample size
ybar <- 15 # sample mean
s2 <- 3 # sample variance

mu <- rep(NA, 11000) # initialises mu vector
tau <- rep(NA, 11000) # initialises tau vector

burn <- 1000 # burnin period
tau[1] <- 1 # initialisation value for tau

# samples from the joint posterior (mu, tau | data)
for(i in 2:11000) {

  mu[i] <- rnorm(n = 1, mean = ybar, sd = sqrt(1 / (n * tau[i - 1])))
  tau[i] <- rgamma(n = 1, shape = n / 2, scale = 2 / ((n - 1) * s2 + n * (mu[i] - ybar)^2))

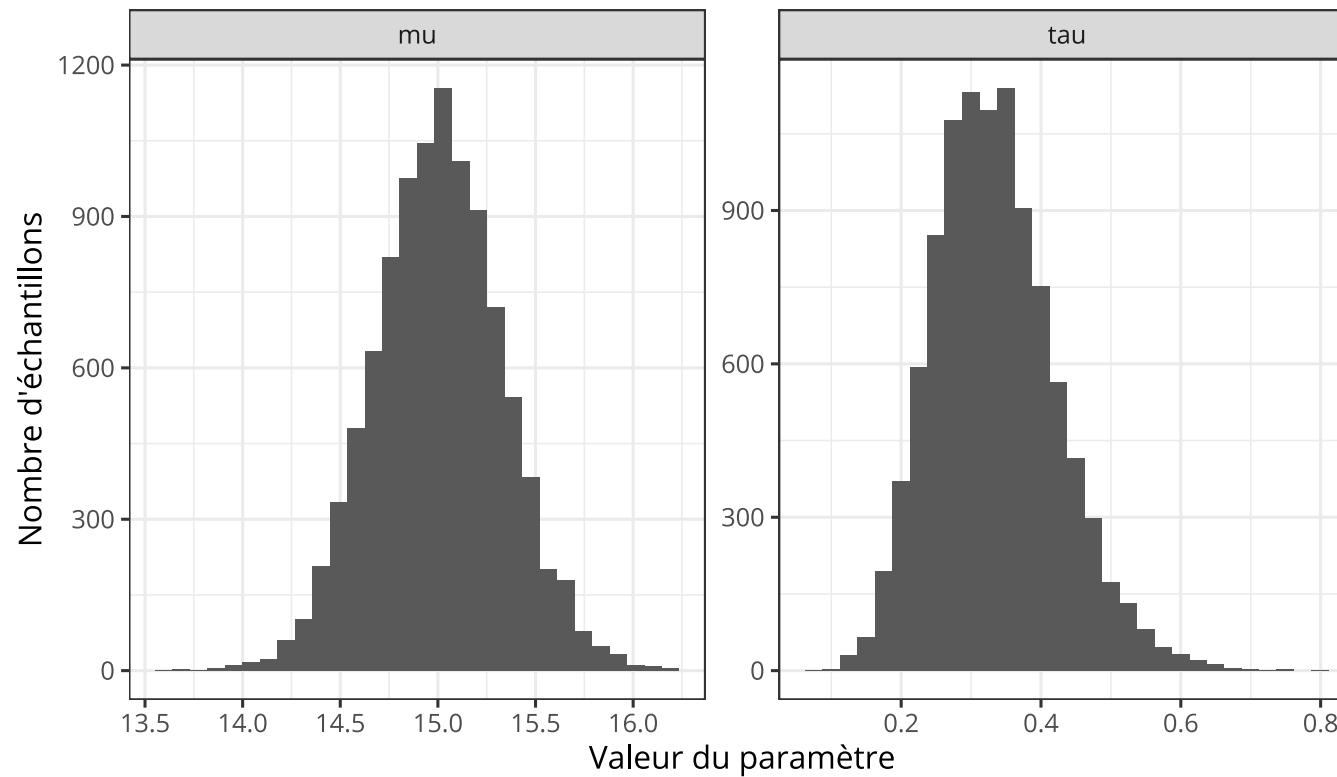
}

mu <- mu[-(1:burn)] # removes burnin
tau <- tau[-(1:burn)] # removes burnin
```

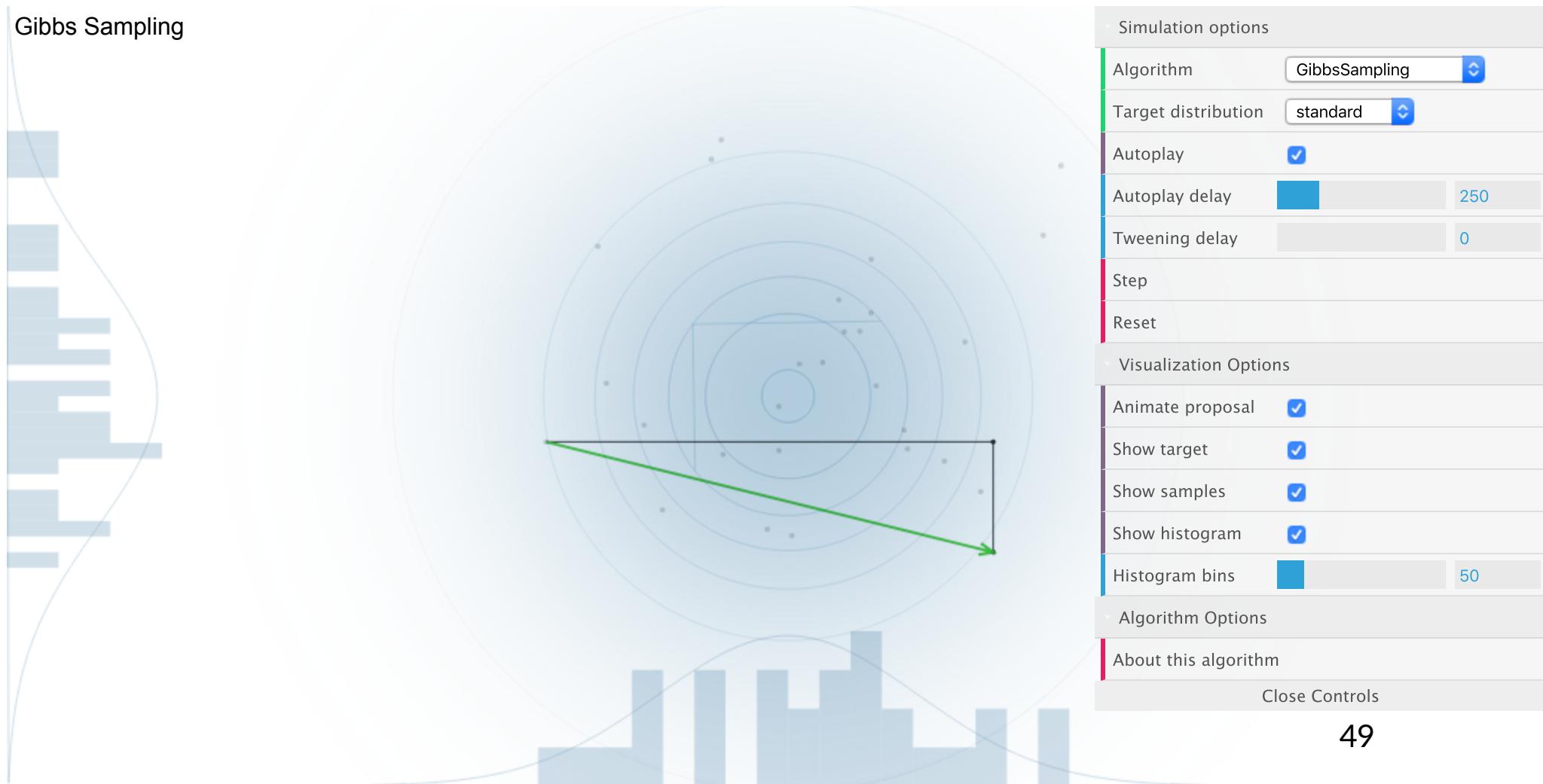


Algorithme Gibbs - Implémentation

```
data.frame(mu = mu, tau = tau) %>%
  pivot_longer(cols = mu:tau) %>%
  ggplot(aes(x = value)) +
  geom_histogram() +
  facet_wrap(~name, scales = "free") +
  labs(x = "Valeur du paramètre", y = "Nombre d'échantillons")
```



Algorithm Gibbs



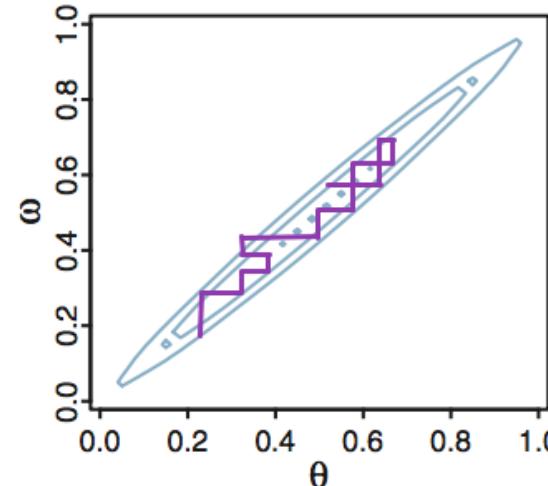
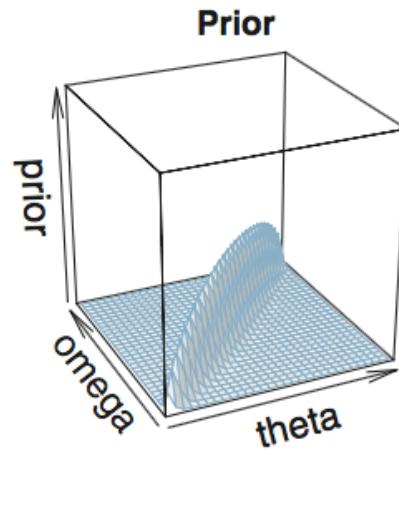
Algorithme Gibbs - Conclusions

1. L'algorithme Gibbs est particulièrement utile lorsque la distribution jointe $p(\{\theta_i\} | \text{data})$ n'est pas calculable ou n'est pas facile à échantillonner, mais que les densités conditionnelles $p(\theta_i | \{\theta_{j \neq i}\}, \text{data})$ le sont.
2. L'algorithme Gibbs peut être vu comme un cas particulier de l'algorithme Metropolis pour lequel la proposition de saut dépend de la position dans l'espace des paramètres et du paramètre sélectionné.
3. En général, l'algorithme Gibbs améliore les performances de l'algorithme Metropolis.



Algorithme Hamiltonian Monte Carlo

On a vu que l'algorithme Gibbs améliore les performances de convergence de l'algorithme Metropolis. Mais il nécessite de pouvoir calculer les densités conditionnelles $p(\theta_i | \theta_{j \neq i}, \text{data})$. L'algorithme est performant lorsqu'il y a indépendance des paramètres $p(\theta_i | \theta_{j \neq i}, \text{data}) = p(\theta_i | \text{data})$... mais ce n'est pas le cas en général !



Algorithme Hamiltonian Monte Carlo

Les algorithmes Metropolis-Hastings et Gibbs ont de mauvaises performances lorsque les paramètres du modèle sont fortement corrélés. L'algorithme **Hamiltonian Monte Carlo** repose sur l'algorithme Metropolis mais résout ces problèmes en utilisant la géométrie de l'espace postérieur. 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 *momentum* (m):



Algorithme Hamiltonian Monte Carlo

Les algorithmes Metropolis-Hastings et Gibbs ont de mauvaises performances lorsque les paramètres du modèle sont fortement corrélés. L'algorithme **Hamiltonian Monte Carlo** repose sur l'algorithme Metropolis mais résout ces problèmes en utilisant la géométrie de l'espace postérieur. 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 *momentum* (m):

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



Algorithme Hamiltonian Monte Carlo

Les algorithmes Metropolis-Hastings et Gibbs ont de mauvaises performances lorsque les paramètres du modèle sont fortement corrélés. L'algorithme **Hamiltonian Monte Carlo** repose sur l'algorithme Metropolis mais résout ces problèmes en utilisant la géométrie de l'espace postérieur. 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 *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) ;



Algorithme Hamiltonian Monte Carlo

Les algorithmes Metropolis-Hastings et Gibbs ont de mauvaises performances lorsque les paramètres du modèle sont fortement corrélés. L'algorithme **Hamiltonian Monte Carlo** repose sur l'algorithme Metropolis mais résout ces problèmes en utilisant la géométrie de l'espace postérieur. 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 *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) ;

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



Algorithme Hamiltonian Monte Carlo

Les algorithmes Metropolis-Hastings et Gibbs ont de mauvaises performances lorsque les paramètres du modèle sont fortement corrélés. L'algorithme **Hamiltonian Monte Carlo** repose sur l'algorithme Metropolis mais résout ces problèmes en utilisant la géométrie de l'espace postérieur. 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 *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) ;

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

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



Algorithme Hamiltonian Monte Carlo

Si on "suit" simplement la gravité de notre position dans l'espace postérieur (imaginons que nous suivions la trajectoire d'une luge ou d'une bille dans cet espace), on arrivera à un minimum de cet espace, mais on veut explorer tout l'espace proportionnellement à la densité postérieure, pas seulement les endroits de forte densité...

Algorithme Hamiltonian Monte Carlo

Si on "suit" simplement la gravité de notre position dans l'espace postérieur (imaginons que nous suivions la trajectoire d'une luge ou d'une bille dans cet espace), on arrivera à un minimum de cet espace, mais on veut explorer tout l'espace proportionnellement à la densité postérieure, pas seulement les endroits de forte densité...

L'algorithme Metropolis repose sur les principes suivants :



Algorithme Hamiltonian Monte Carlo

Si on “suit” simplement la gravité de notre position dans l'espace postérieur (imaginons que nous suivions la trajectoire d'une luge ou d'une bille dans cet espace), on arrivera à un minimum de cet espace, mais on veut explorer tout l'espace proportionnellement à la densité postérieure, pas seulement les endroits de forte densité...

L'algorithme Metropolis repose sur les principes suivants :

- La proposition de déplacement est toujours une gaussienne multivariée centrée sur la position courante.



Algorithme Hamiltonian Monte Carlo

Si on “suit” simplement la gravité de notre position dans l'espace postérieur (imaginons que nous suivions la trajectoire d'une luge ou d'une bille dans cet espace), on arrivera à un minimum de cet espace, mais on veut explorer tout l'espace proportionnellement à la densité postérieure, pas seulement les endroits de forte densité...

L'algorithme Metropolis repose sur les principes suivants :

- La proposition de déplacement est toujours une gaussienne multivariée centrée sur la position courante.
- La proposition de déplacement est une gaussienne qui garde toujours la même forme (variance constante).



Algorithme Hamiltonian Monte Carlo

Si on “suit” simplement la gravité de notre position dans l'espace postérieur (imaginons que nous suivions la trajectoire d'une luge ou d'une bille dans cet espace), on arrivera à un minimum de cet espace, mais on veut explorer tout l'espace proportionnellement à la densité postérieure, pas seulement les endroits de forte densité...

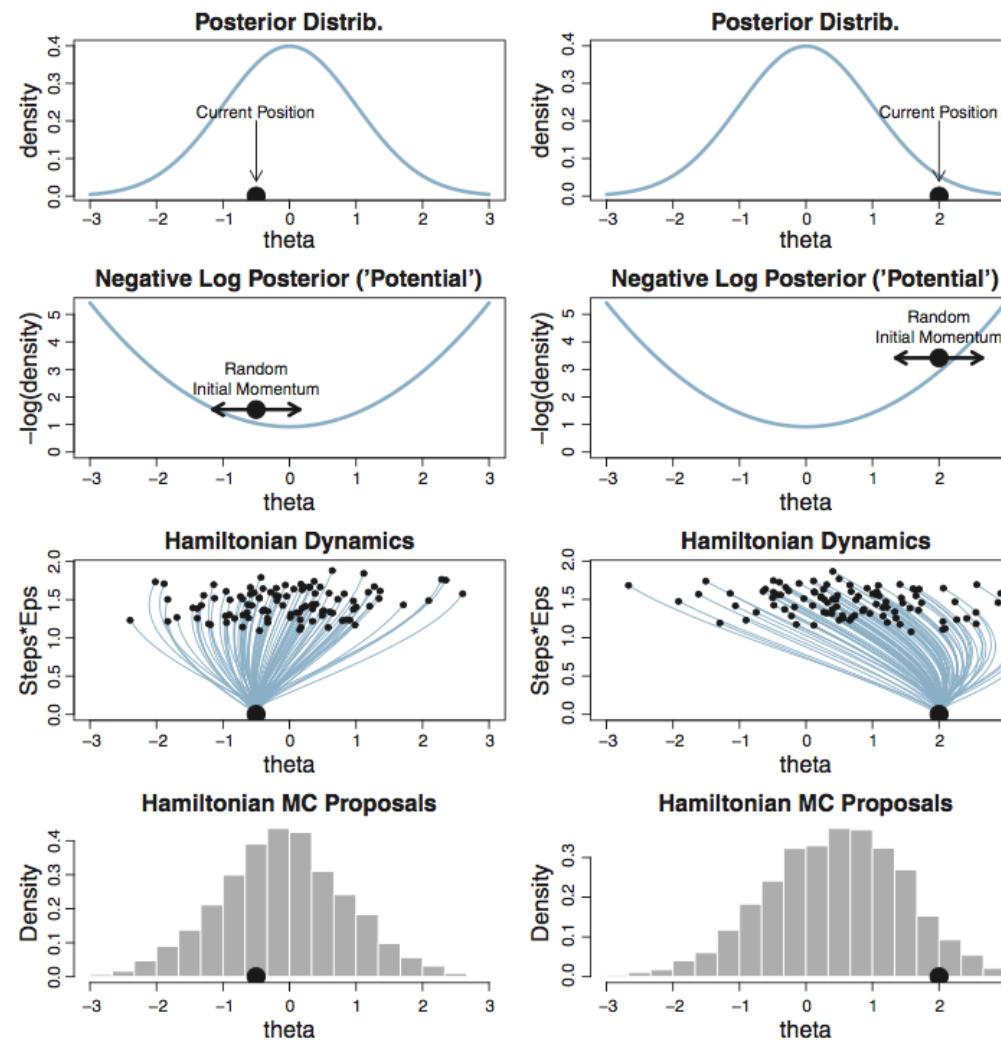
L'algorithme Metropolis repose sur les principes suivants :

- La proposition de déplacement est toujours une gaussienne multivariée centrée sur la position courante.
- La proposition de déplacement est une gaussienne qui garde toujours la même forme (variance constante).

L'algorithme HMC repose sur une autre idée : Adapter la proposition de déplacement en fonction de la position courante du paramètre et de la géométrie du postérieur aux alentours de cette position.



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 des paramètres

Algorithme Hamiltonian Monte Carlo

- Sélectionner un point de départ θ_0 : On peut sélectionner n'importe quelle valeur de θ dans l'espace des paramètres
- Générer une nouvelle position à partir de la fonction de potentiel
 - *L'algorithme génère une proposition par analogie au lancer d'une bille sur la fonction de potentiel*
 - *Le pas de déplacement, ainsi que le nombre de déplacement, sont fixés à l'avance*
 - *La force avec laquelle on lance la bille est déterminée par une loi normale multivariée : $m \sim MVNormal(\mu, \Sigma)$*
 - *Estimation de la trajectoire discrétisée via la méthode leapfrog*



Algorithme Hamiltonian Monte Carlo

- Sélectionner un point de départ θ_0 : On peut sélectionner n'importe quelle valeur de θ dans l'espace des paramètres
- Générer une nouvelle position à partir de la fonction de potentiel
 - *L'algorithme génère une proposition par analogie au lancer d'une bille sur la fonction de potentiel*
 - *Le pas de déplacement, ainsi que le nombre de déplacement, sont fixés à l'avance*
 - *La force avec laquelle on lance la bille est déterminée par une loi normale multivariée : $m \sim MVNormal(\mu, \Sigma)$*
 - *Estimation de la trajectoire discrétisée via la méthode leapfrog*
- Accepter ou rejeter la proposition de déplacement suivant la probabilité (où ϕ est le moment associé à la bille) :



Algorithme Hamiltonian Monte Carlo

- Sélectionner un point de départ θ_0 : On peut sélectionner n'importe quelle valeur de θ dans l'espace des paramètres
- Générer une nouvelle position à partir de la fonction de potentiel
 - *L'algorithme génère une proposition par analogie au lancer d'une bille sur la fonction de potentiel*
 - *Le pas de déplacement, ainsi que le nombre de déplacement, sont fixés à l'avance*
 - *La force avec laquelle on lance la bille est déterminée par une loi normale multivariée : $m \sim MVNormal(\mu, \Sigma)$*
 - *Estimation de la trajectoire discrétisée via la méthode leapfrog*
- Accepter ou rejeter la proposition de déplacement suivant la probabilité (où ϕ est le moment associé à la bille) :

$$\Pr_{\text{déplacement}} = \min \left(\frac{\Pr(\theta_{\text{proposé}} | D) \Pr(\phi_{final})}{\Pr(\theta_{\text{courant}} | D) \Pr(\phi_{initial})}, 1 \right)$$



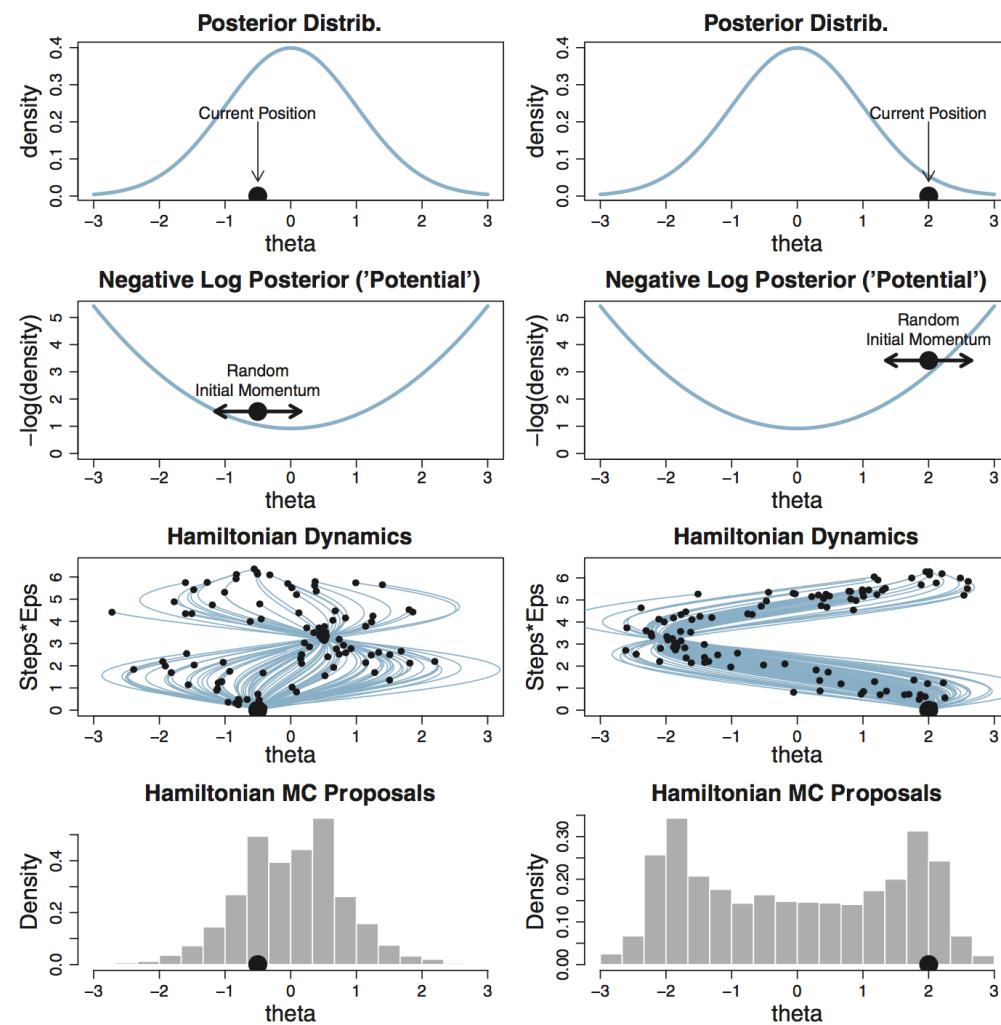
Algorithme Hamiltonian Monte Carlo

- Sélectionner un point de départ θ_0 : On peut sélectionner n'importe quelle valeur de θ dans l'espace des paramètres
- Générer une nouvelle position à partir de la fonction de potentiel
 - *L'algorithme génère une proposition par analogie au lancer d'une bille sur la fonction de potentiel*
 - *Le pas de déplacement, ainsi que le nombre de déplacement, sont fixés à l'avance*
 - *La force avec laquelle on lance la bille est déterminée par une loi normale multivariée : $m \sim MVNormal(\mu, \Sigma)$*
 - *Estimation de la trajectoire discrétisée via la méthode leapfrog*
- Accepter ou rejeter la proposition de déplacement suivant la probabilité (où ϕ est le moment associé à la bille) :

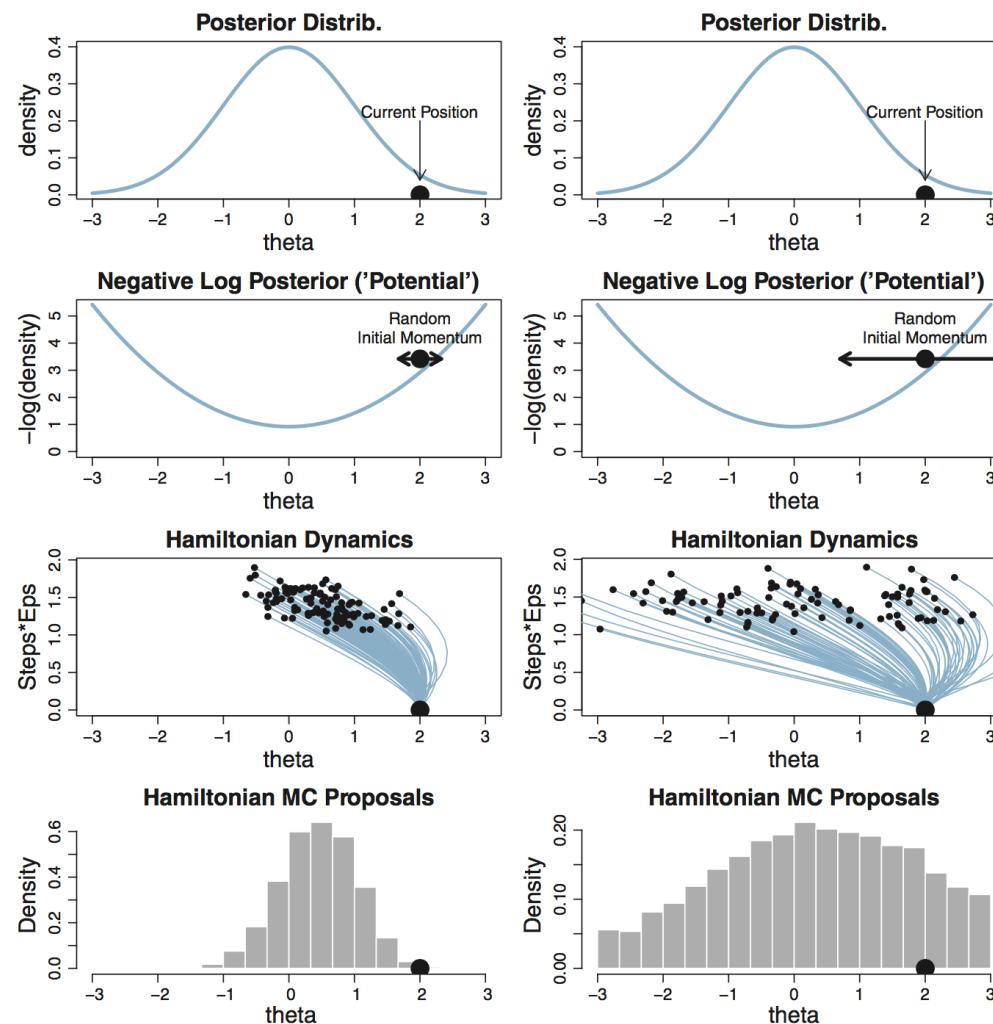
$$\Pr_{\text{déplacement}} = \min \left(\frac{\Pr(\theta_{\text{proposé}} | D) \Pr(\phi_{final})}{\Pr(\theta_{\text{courant}} | D) \Pr(\phi_{initial})}, 1 \right)$$

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

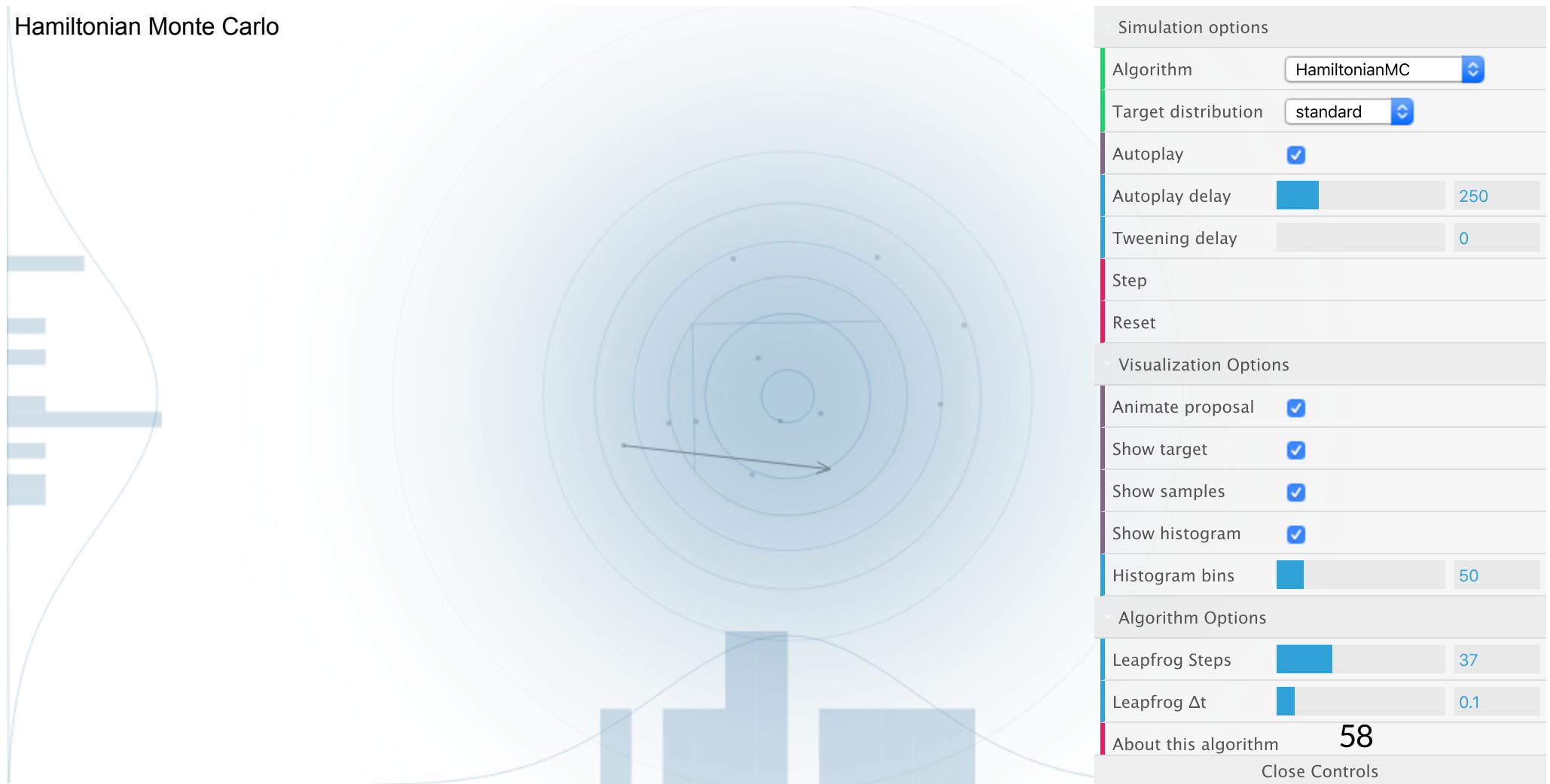
Influence de la durée de déplacement...



Influence de la variabilité du moment initial...



Algorithme Hamiltonian Monte Carlo



Algorithme Hamiltonian Monte Carlo

1. L'algorithme HMC est particulièrement utile lorsque le modèle contient un grand nombre de paramètres et quand les paramètres sont corrélés entre eux.
2. Il nécessite un paramétrage assez fin. Des logiciels proposent des méthodes clés en main très efficaces.
3. Le package brms utilise une variation de cet algorithme (l'algorithme NUTS) qui permet d'éviter les trajectoires cycliques.



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 pour Gibbs, ou variance du moment initial pour HMC).

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 pour Gibbs, 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 :



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 pour Gibbs, 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 :



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 pour Gibbs, 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 :

1. 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



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 pour Gibbs, 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 :

1. 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



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 pour Gibbs, 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 :

1. 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

2. 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



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 pour Gibbs, 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 :

1. 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

2. 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



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 pour Gibbs, 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 :

1. 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

2. 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

3. La chaîne doit être générée de manière efficace (avec le moins d'itérations possible)



Évaluation des MCMC - Représentativité

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).



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).



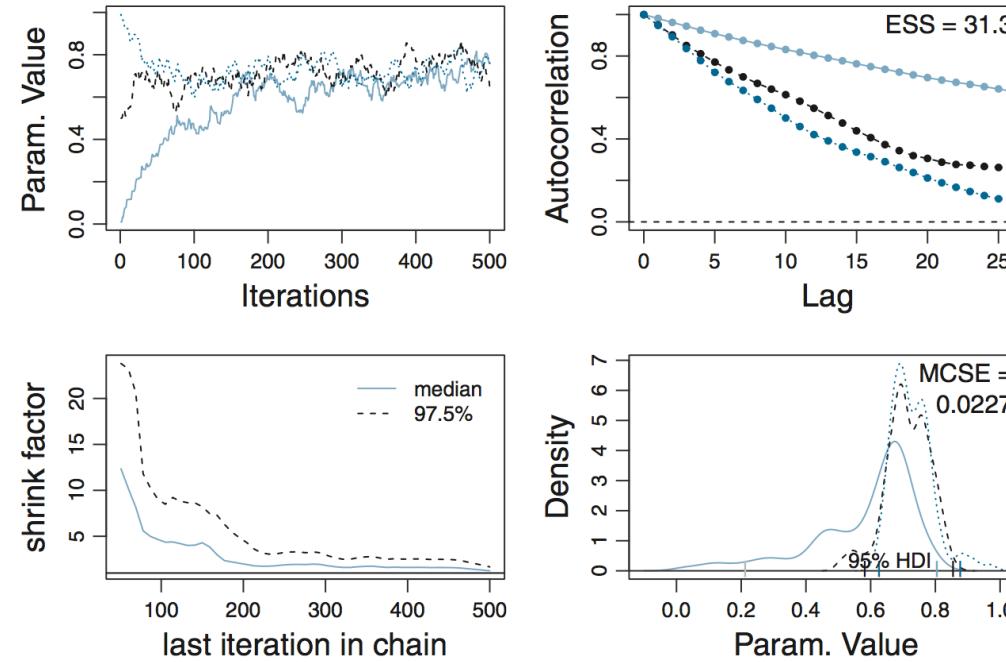
Évaluation des MCMC - Représentativité

- 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

- 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.

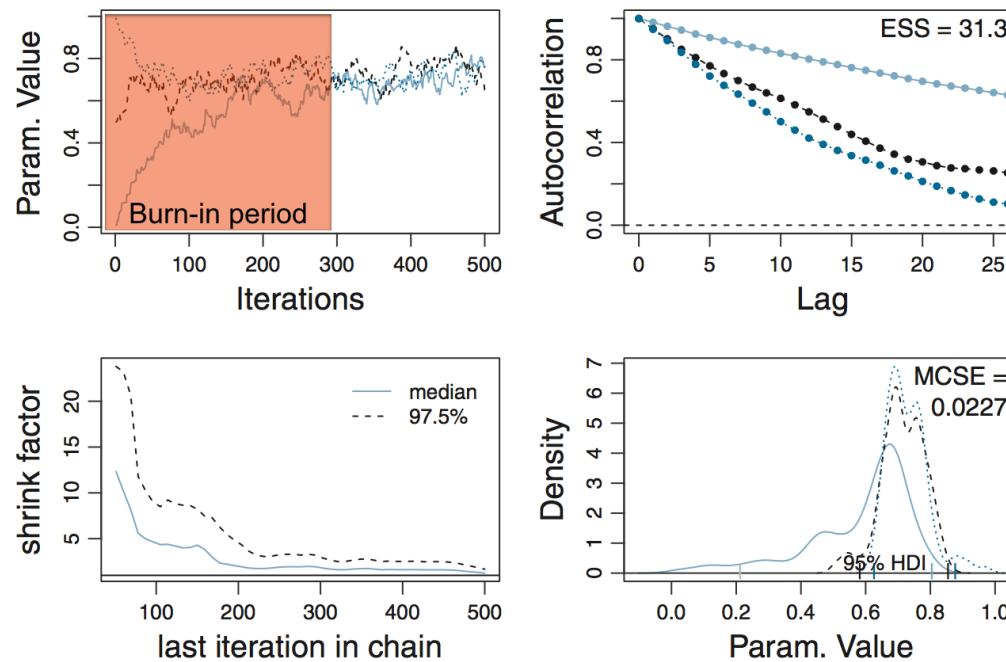


Évaluation des MCMC - Représentativité

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 (*burn-in* ou *warm-up*).

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 (*burn-in* ou *warm-up*).

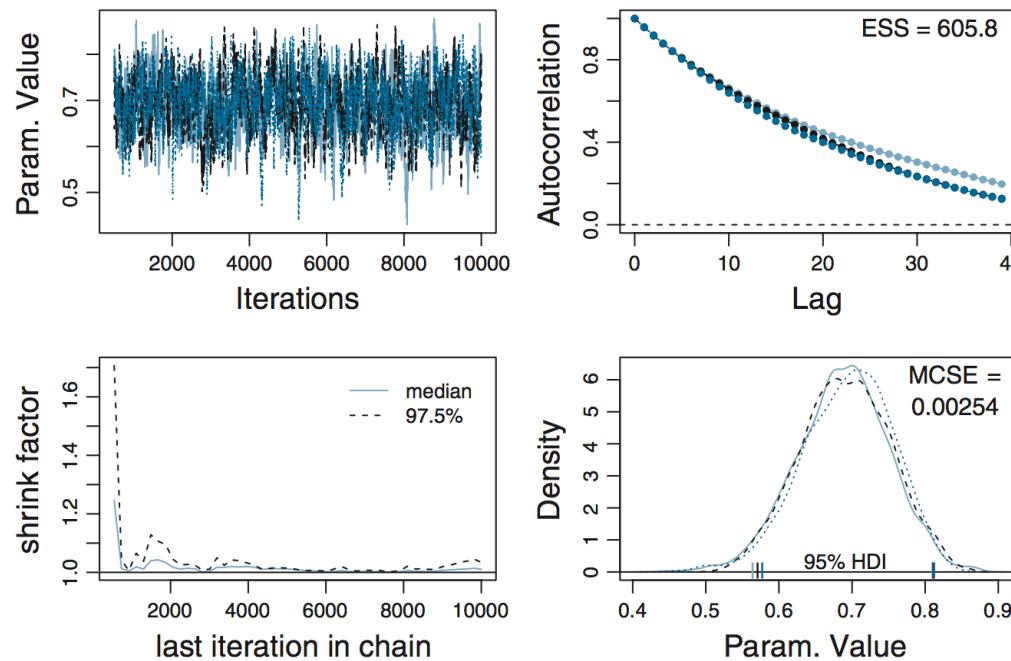


Évaluation des MCMC - Représentativité

Vérification numérique des chaînes : Le *shrink factor* (aussi connu comme \hat{R} ou R_{hat}) 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 1.01).

Évaluation des MCMC - Représentativité

Vérification numérique des chaînes : Le *shrink factor* (aussi connu comme \hat{R} ou R_{hat}) 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 1.01).



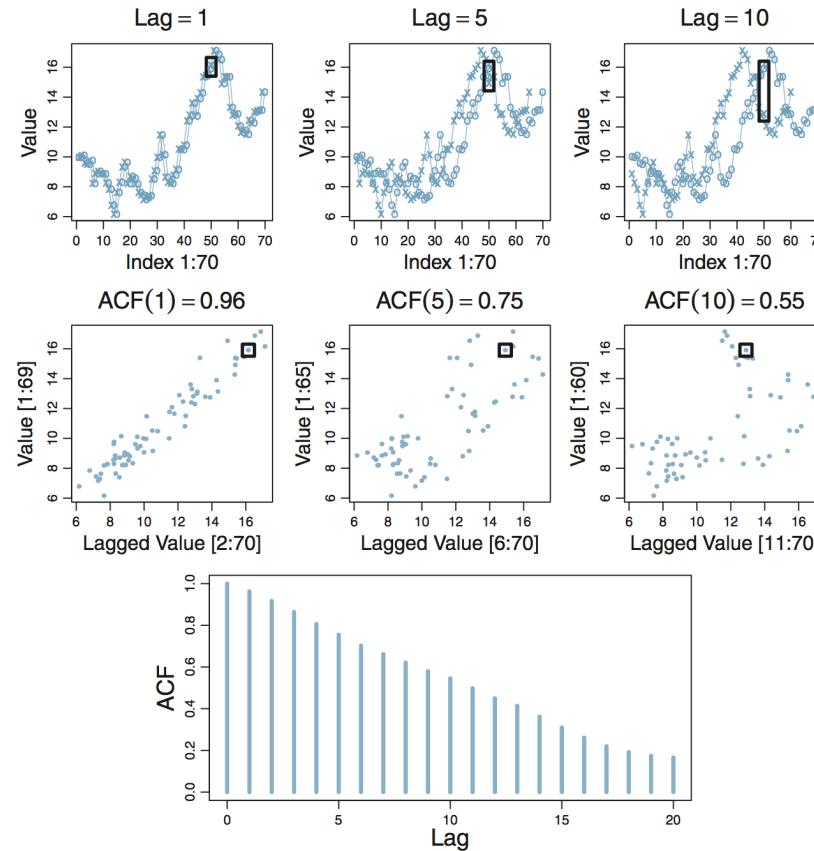
Évaluation des MCMC - Stabilité et précision

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 précision

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).



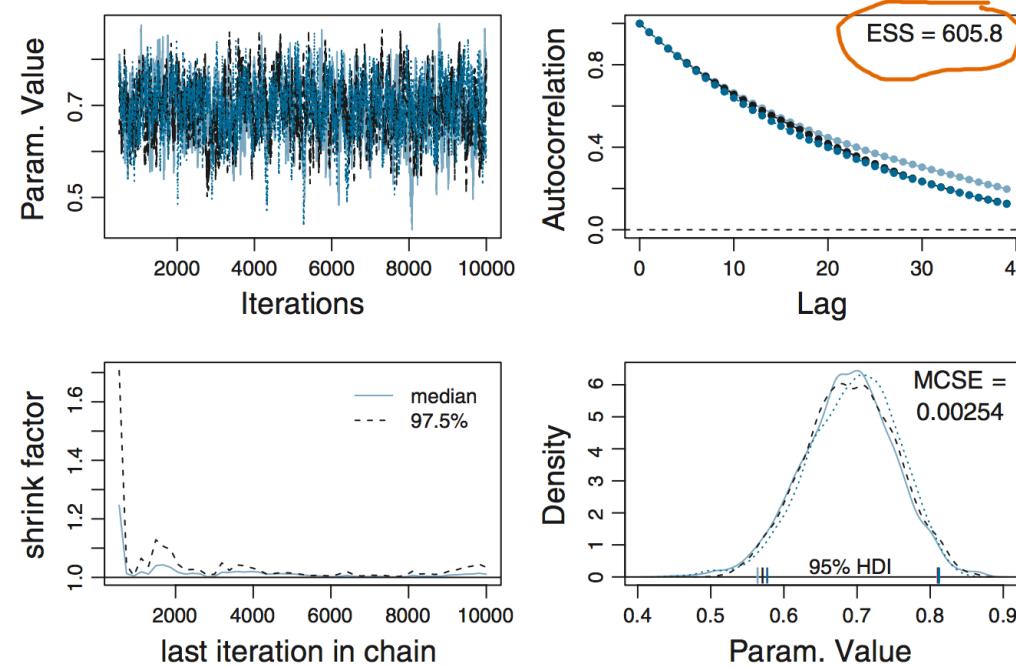
Évaluation des MCMC - Stabilité et précision

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 précision

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.



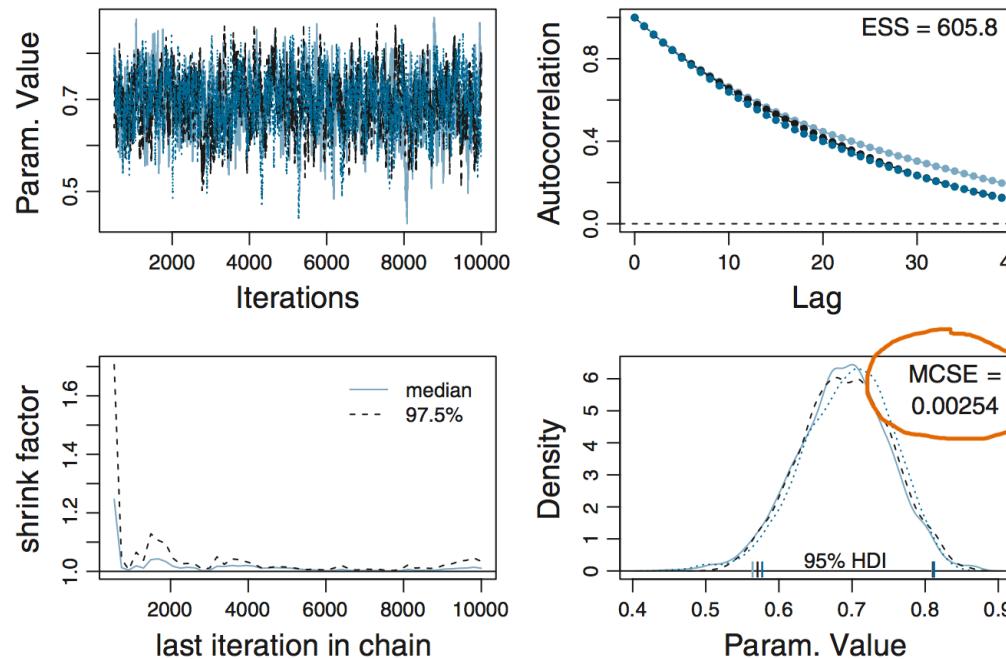
Évaluation des MCMC - Stabilité et précision

L'erreur standard d'un ensemble d'échantillons est donné : $SE = SD/\sqrt{N}$. 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 - Stabilité et précision

L'erreur standard d'un ensemble d'échantillons est donné : $SE = SD/\sqrt{N}$. 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



Évaluation des MCMC - Implémentation via brms

```
library(rethinking)
library(tidyverse)
library(brms)

data(Howell1)
d <- Howell1
d2 <- d %>% filter(age >= 18)

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

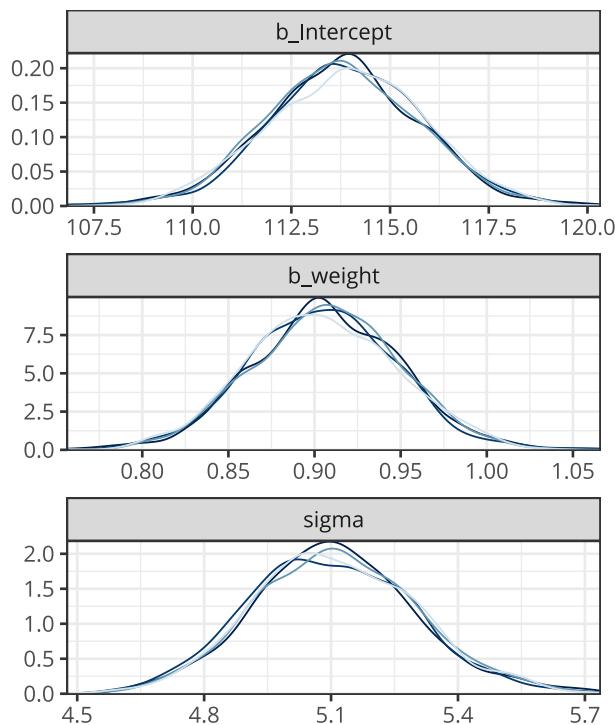
mod1 <- brm(
  height ~ 1 + weight,
  prior = priors,
  family = gaussian(),
  data = d2,
  chains = 4, # nombre de MCMCs
  iter = 2000, # nombre total d'itérations (par chaîne)
  warmup = 1000, # nombre d'itérations pour le warm-up
  thin = 1 # thinning (1 = no thinning)
)
```



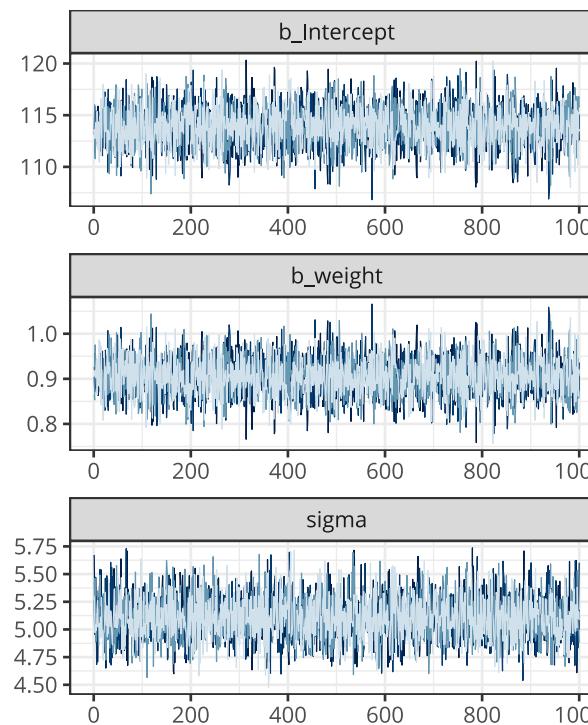
Évaluation des MCMC - Implémentation via brms

```
# 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"),
  theme = theme_bw(base_size = 16, base_family = "Open Sans")
)
```



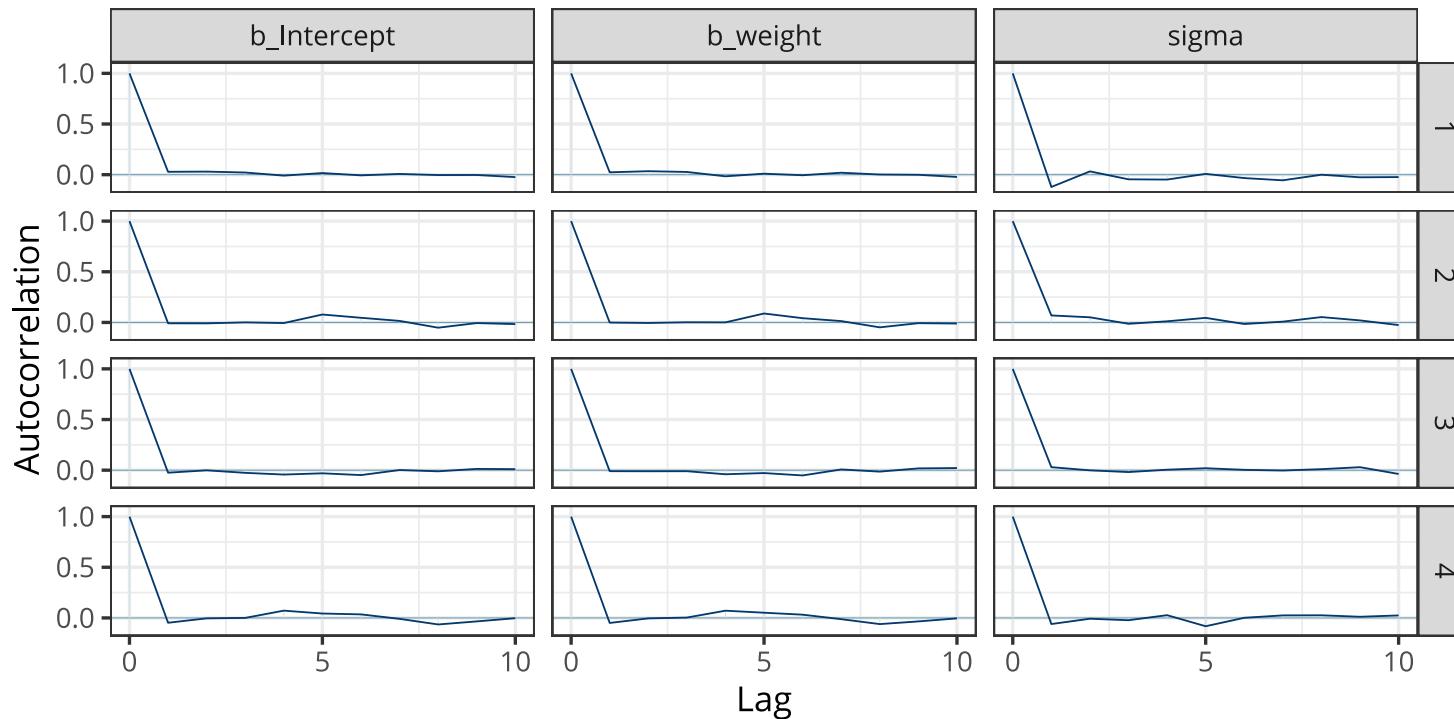
Chain
— 1
— 2
— 3
— 4



Chain
— 1
— 2
— 3
— 4

Évaluation des MCMC - Implémentation via brms

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



Évaluation des MCMC - Implémentation via brms

```
summary(mod1)
```

Family: gaussian
Links: mu = identity; sigma = 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

Population-Level Effects:

	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
Intercept	113.85	1.93	110.11	117.60	1.00	4112	2918
weight	0.91	0.04	0.82	0.99	1.00	3962	3012

Family Specific Parameters:

	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
sigma	5.11	0.19	4.74	5.51	1.00	4124	3036

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).



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 extrêmes (par exemple pour calculer un intervalle de crédibilité).



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).

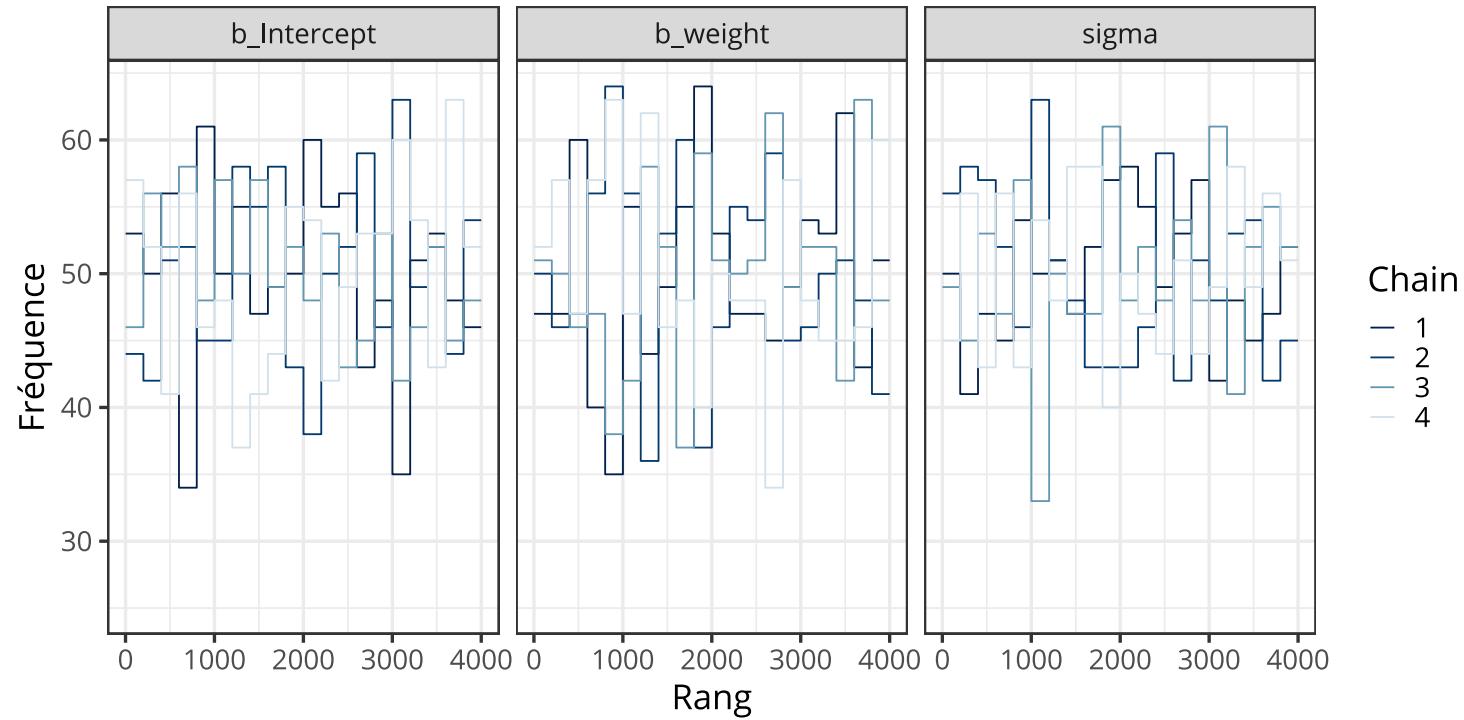
Tail-ESS donne le minimum de l'ESS calculé pour les quantiles à 5% et 95% (i.e., pour les queues de la distribution des échantillons normalisés par leur rang). Cette valeur doit être élevée si nous accordons de l'importance à l'estimation des valeurs extrêmes (par exemple pour calculer un intervalle 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 (*samplers*) pour obtenir des échantillons issus de la distribution postérieure (non-normalisée). Ces échantillons peuvent ensuite être utilisés pour calculer différentes statistiques sur la distribution postérieure (e.g., moyenne, médiane, HDI).



Résumé du cours

Nous avons introduit et discuté l'utilisation d'échantillonneurs (*samplers*) pour obtenir des échantillons issus de la distribution postérieure (non-normalisée). Ces échantillons peuvent ensuite être utilisés pour calculer différentes statistiques sur 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... L'algorithme Gibbs accélère la convergence mais nécessite qu'on l'on puisse calculer la distribution conditionnelle des paramètres. De plus, cet algorithme ne fonctionne pas bien lorsqu'il existe de fortes corrélations entre les différents paramètres...



Résumé du cours

Nous avons introduit et discuté l'utilisation d'échantillonneurs (*samplers*) pour obtenir des échantillons issus de la distribution postérieure (non-normalisée). Ces échantillons peuvent ensuite être utilisés pour calculer différentes statistiques sur 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... L'algorithme Gibbs accélère la convergence mais nécessite qu'on l'on puisse calculer la distribution conditionnelle des paramètres. 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 que les deux précédents et moins d'échantillons seront nécessaires pour approcher la distribution postérieure.



Résumé du cours

Nous avons introduit et discuté l'utilisation d'échantillonneurs (*samplers*) pour obtenir des échantillons issus de la distribution postérieure (non-normalisée). Ces échantillons peuvent ensuite être utilisés pour calculer différentes statistiques sur 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... L'algorithme Gibbs accélère la convergence mais nécessite qu'on l'on puisse calculer la distribution conditionnelle des paramètres. 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 que les deux précédents et 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`.

```
library(rethinking)
library(tidyverse)

data(rugged)
d <- rugged
d$log_gdp <- log(d$rgdppc_2000)
df1 <- d[complete.cases(d$rgdppc_2000), ]
str(df1[, 1:5])

'data.frame': 170 obs. of 5 variables:
 $ isocode   : Factor w/ 234 levels "ABW", "AFG", "AGO", ...: 3 5 8 9 10 12 13 14 15 16 ...
 $ isonum    : int  24 8 784 32 51 28 36 40 31 108 ...
 $ country   : Factor w/ 234 levels "Afghanistan", ...: 6 2 219 9 10 8 12 13 14 34 ...
 $ rugged    : num  0.858 3.427 0.769 0.775 2.688 ...
 $ rugged_popw: num  0.714 1.597 0.316 0.22 0.934 ...
```



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. Examinez ensuite les estimations de ce modèle (interprétation, diagnostiques des MCMCs).

$$\log(gdp_i) \sim \text{Normal}(\mu_i, \sigma_i)$$

$$\mu_i = \alpha + \beta_1 \cdot \text{rugged}_i + \beta_2 \cdot \text{continent}_i + \beta_3 \cdot (\text{rugged}_i \cdot \text{continent}_i)$$

$$\alpha \sim \dots$$

$$\beta_1, \beta_2, \beta_3 \sim \dots$$



Proposition de réponse

```
priors2 <- c(  
  set_prior("normal(0, 100)", class = "Intercept"),  
  set_prior("normal(0, 10)", class = "b"),  
  set_prior("exponential(0.01)", class = "sigma")  
)  
  
mod2 <- brm(  
  log_gdp ~ 1 + rugged * cont_africa,  
  prior = priors2,  
  family = gaussian(),  
  data = df1,  
  chains = 4, # nombre de MCMCs  
  iter = 2000, # nombre total d'itérations (par chaîne)  
  warmup = 1000 # nombre d'itérations pour le warm-up  
)
```



Proposition de réponse

```
summary(mod2)
```

Family: gaussian
Links: mu = identity; sigma = 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

Population-Level Effects:

	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
Intercept	9.22	0.14	8.94	9.50	1.00	2798	2875
rugged	-0.20	0.08	-0.36	-0.05	1.00	2594	2744
cont_africa	-1.95	0.23	-2.42	-1.49	1.00	2675	2674
rugged:cont_africa	0.39	0.14	0.13	0.66	1.00	2505	2568

Family Specific Parameters:

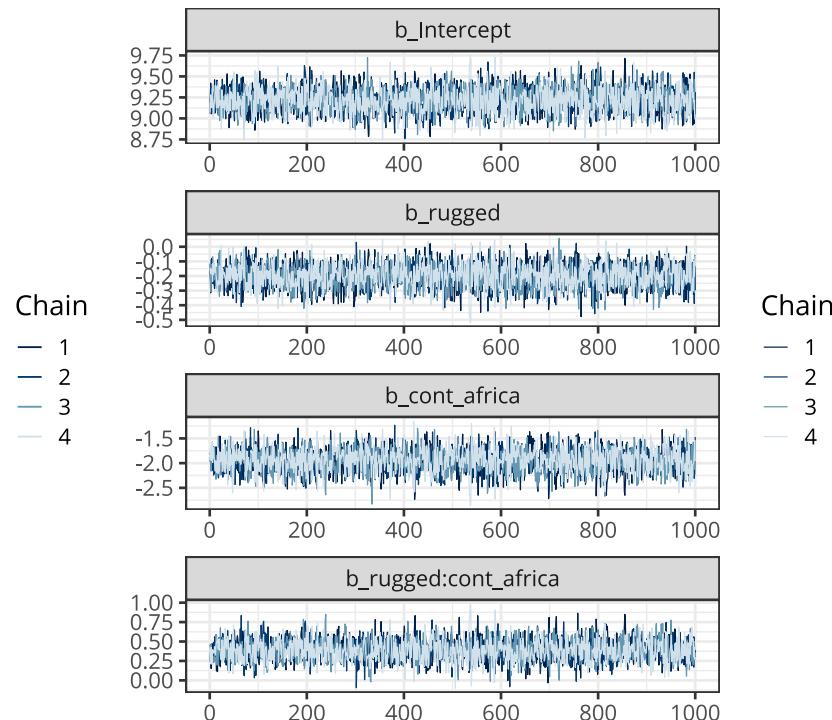
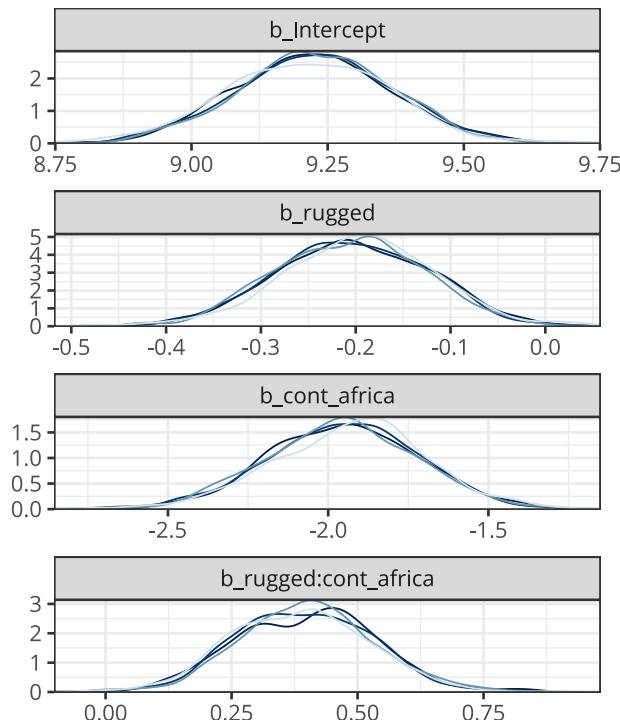
	Estimate	Est.Error	l-95% CI	u-95% CI	Rhat	Bulk_ESS	Tail_ESS
sigma	0.95	0.05	0.86	1.06	1.00	4043	3002

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

```
plot(  
  x = mod2, combo = c("dens_overlay", "trace"), pars = "^.b.",  
  theme = theme_bw(base_size = 16, base_family = "Open Sans")  
)
```



Proposition de réponse

pairs (mod2)

