Inférence bayesienne

Eric Marcon

12 février 2021

Résumé

Présentation pas à pas de la méthode. Adaptation d'un code original de Bruno Hérault.

Table des matières

1 Principes

D'après le théorème de Bayes, la vraisemblance du vecteur de paramètres d'un modèle sachant les données est proportionnelle à vraisemblance des paramètres (selon leur distribution *a priori*) multipliée par la vraisemblance des données (selon le modèle) sachant le vecteur de paramètres.

L'élément manquant pour écrire tous les termes du théorème de Bayes est la vraisemblance (intrinsèque) des données, qui n'est pas connue mais est constante et n'intervient donc pas dans l'inférence. La distribution du vecteur de paramètres est donc estimée en maximisant le produit des vraisemblances (en pratique, la somme de leurs logarithmes) des paramètres et des données.

2 Modèle

L'objectif est l'estimation des paramètre du modèle $y = ax + b + \epsilon$ où ϵ suit une loi normale.

Les données sont constituées par les vecteurs $\mathbf{Y}$ et $\mathbf{X}$.

Le modèle est une représentation de la réalité choisie par le modélisateur : ici, une relation linéaire entre $\mathbf{Y}$ et $\mathbf{X}$, avec une erreur gaussienne d'écart-type σ .

Les paramètres a, b et σ sont à estimer. On les regroupe classiquement dans un vecteur de paramètres appelé θ. Dans le cadre bayesien, le résultat de l'estimation sera la densité de probabilité de chaque paramètre, sachant les données.

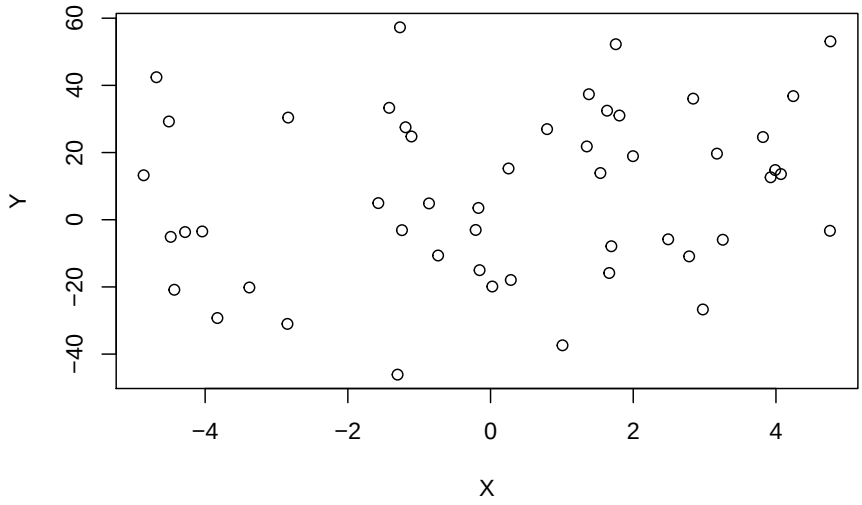
2.1 Fabrication des données

Pour illustrer le fonctionnement de l'inférence, des données sont d'abord simulées :

```
# Paramètres
a <- 10
b <- 2
sigma <- 20
n <- 50
# Données
X <- runif(n, min = -5, max = 5)
Y <- a + b * X + rnorm(n, 0, sigma)
```

Les données simulées sont présentées sur la figure :

```
plot(Y ~ X)
```



La variance du modèle est volontairement très grande pour que l'estimation des paramètres soit difficile.

3 Inférence

L'inférence de θ est faite en maximisant la vraisemblance de $\theta|\mathbf{Y}$, appelée distribution *a posteriori* de θ ou plus simplement *posterior*.

La vraisemblance de $\theta|\mathbf{Y}$ est proportionnelle au produit des vraisemblances de $\mathbf{Y}|\theta$ et de celle des paramètres estimés dans la distribution *a priori* (c'est-à-dire avant de connaître les données) de θ, appelée *prior*. Chacune des deux vraisemblances doit être écrite en fonction de θ.

3.1 Vraisemblance de $\mathbf{Y}|\theta$

Les paramètres sont recherchés par un algorithme de proposition présenté plus loin. A chaque proposition, la vraisemblance de $\mathbf{Y}|\theta$ est calculée.

Le logarithme de la vraisemblance des données $\mathbf{Y}|\theta$ (sachant les paramètres) dépend du modèle d'erreur : étant donné $\mathbf{X}$ et θ, chaque valeur y_i de $\mathbf{Y}$ suit ici une loi normale d'espérance $a + bx_i$ et d'écart-type σ. Le logarithme de la vraisemblance de chaque valeur de $\mathbf{Y}$ est calculée par la fonction `dnorm`. La vraisemblance de $\mathbf{Y}$ est le produit des vraisemblances des y_i : la somme des logarithmes de vraisemblance des y_i est donc retournée.

```
ll_Y_theta <- function(Y, X, theta) {
    a <- theta[1]
    b <- theta[2]
```

```
    sigma <- theta[3]
    # Valeur prédite par le modèle
    prediction <- a + b * X
    # Log-vraisemblance de chaque valeur de y
    single_likelihoods <- dnorm(Y, mean = prediction,
        sd = sigma, log = TRUE)
    # Possibilité de NaN si sd proposé négatif par
    # exemple
    ll <- sum(single_likelihoods)
    # Vraisemblance nulle dans ce cas
    if (is.na(ll))
        ll <- -Inf
    return(ll)
}
```

Cette fonction doit être réécrite si le modèle est différent.

3.2 Vraisemblance *a priori* de θ

La distribution *a priori* des paramètres est choisie par le modélisateur. Elle peut correspondre à une connaissance d'expert ou au contraire être très peu informative.

Ici, la distribution de a et b est *a priori* uniforme entre -100 et 100 : la vraisemblance des paramètres sera toujours identique dans ces intervalles qui contiennent forcément toutes les valeurs proposables des paramètres étant donné l'ordre de grandeur des données. Ce sont donc des priors aussi peu informatifs que possible. L'écart-type de l'erreur du modèle est choisie dans une distribution uniforme entre 0 et 30, pour les mêmes raisons.

La somme des logarithmes de vraisemblance des paramètres est retournée.

```
ll_prior <- function(theta) {
    a <- theta[1]
    b <- theta[2]
    sigma <- theta[3]
    # Log-vraisemblance de chaque paramètre dans sa loi
    # a priori
    a_prior <- dunif(a, min = -100, max = 100, log = TRUE)
    b_prior <- dunif(b, min = -100, max = 100, log = TRUE)
    sigma_prior <- dunif(sigma, min = 0, max = 30,
        log = TRUE)
    return(a_prior + b_prior + sigma_prior)
}
```

Cette fonction doit être réécrite si le modèle est différent.

3.3 Vraisemblance de $\theta|\mathbf{Y}$

Le logarithme de la vraisemblance des paramètres sachant les données, $\theta|\mathbf{Y}$, est à une constante près la somme des logarithmes de vraisemblance des données et des paramètres :

```
ll_posterior <- function(Y, X, theta) {
    return(ll_Y_theta(Y, X, theta) + ll_prior(theta))
}
```

Cette fonction permet de calculer, pour toute proposition de θ, le logarithme de sa vraisemblance (à une constante près, simplement ignorée).

La construction de la vraisemblance permet de comprendre le poids relatif des données et du prior. Les log-vraisemblances sont sommées. Chaque observation y_i contribue pour un terme de la somme, de même que chaque paramètre :

- Augmenter le nombre d'observations diminue le poids relatif du prior ;
- Les priors peu informatifs dont la vraisemblance varie peu influent peu sur la recherche du maximum de vraisemblance.

3.4 Fonction de proposition

La fonction de proposition permet d'explorer l'espace des paramètres, c'est-à-dire toutes les valeurs possibles de θ. Ici, il s'agit d'un espace en trois dimensions correspondant aux valeurs possibles de a, b et σ. C'est un parallélépipède borné par les valeurs -100 et 100 pour a et b, et 0 et 30 pour σ : les autres valeurs sont interdites par la loi *a priori* qui leur donne une vraisemblance nulle.

La fonction la plus simple est une marche aléatoire :

```
proposal <- function(theta, sigma_prop) {
    return(rnorm(length(theta), mean = theta, sd = sigma_prop))
}
```

A partir d'une valeur de θ, une nouvelle valeur est retournée dans son voisinage. La différence de valeur de chaque paramètre est tiré dans une loi normale d'écart-type `sigma_prop` fixé par le modélisateur (appelé pas de la marche aléatoire). Si le pas est trop petit, l'exploration de l'espace des paramètres sera inutilement longue et pourra même rester bloquée dans des régions correspondant à des maximums locaux de vraisemblance. Si le pas est trop grand, l'exploration sera trop grossière pour détecter des régions intéressantes.

3.5 Chaine de Markov

L'inférence est réalisée par une chaîne de Markov. A partir d'une valeur initiale de θ, `theta_0`, une proposition d'un nouveau vecteur θ est faite. Le rapport des vraisemblances entre la proposition et la valeur originale de θ est calculé. La proposition est acceptée d'autant plus probablement que ce rapport est élevé. Un nombre aléatoire est tiré dans une loi uniforme entre 0 et 1 pour fixer un seuil d'acceptation. Si le rapport de vraisemblance est supérieur au seuil d'acceptation (c'est toujours le cas si la vraisemblance est améliorée) alors la proposition est acceptée. Si elle est refusée, θ ne change pas. L'opération est répétée un grand nombre de fois (`iterations`). Après un certain nombre d'itérations, `burn_in` (préchauffage), la chaîne de Markov est supposée converger vers une région où les paramètres vont se stabiliser avec une grande vraisemblance.

Les valeurs de θ et de log-vraisemblance, à chaque itération, sont enregistrées.

```
MetropolisMCMC <- function(Y, X, sigma_prop, theta_0,
    iterations) {
    # Stockage. Chaque ligne du tableau contient une
    # itération ; les colonnes contiennent theta
    chain <- matrix(nrow = iterations + 1, ncol = length(theta_0))
    # Noms des paramètres
    colnames(chain) <- names(theta_0)
    # Stockage des vraisemblances
    ll_data <- numeric(length = iterations + 1)
    # Valeurs initiales
    chain[1, ] <- theta_0
    ll_data[1] <- ll_Y_theta(Y, X, theta_0)
    # Initialisation d'une barre de progression
    pgb <- txtProgressBar(min = 0, max = iterations)
    # Chaîne de Markov
    for (i in 1:iterations) {
        # Proposition d'une valeur de theta
        theta_proposal <- proposal(chain[i, ], sigma_prop)
        # Rapport de vraisemblance entre la proposition et
        # la valeur précédente de theta
        l_ratio <- exp(ll_posterior(Y, X, theta_proposal) -
            ll_posterior(Y, X, chain[i, ]))
        # Acceptation ou non
        if (runif(1) < l_ratio) {
            chain[i + 1, ] <- theta_proposal
        } else {
            chain[i + 1, ] <- chain[i, ]
        }
        # Enregistrement de la vraisemblance de Y sachant
        # theta
        ll_data[i + 1] <- ll_Y_theta(Y, X, chain[i +
            1, ])
        setTxtProgressBar(pgb, i)
    }
    # Retour d'un tableau complet: theta et
    # vraisemblance
    return(cbind(chain, LogVraisemblance = ll_data))
}
```

3.6 Exécution

Le nombre d'itérations doit être grand. La taille des pas de la marche aléatoire est affaire d'expérience : les écarts-types de la marche aléatoire doivent être augmentés pour diminuer le taux d'acceptation (objectif : 30%). La valeur de départ des paramètres est choisie aléatoirement dans l'espace des possibles, en cohérence avec la distribution *a priori*. Elle doit être possible (la vraisemblance des paramètres ne doit pas être nulle).

```
# Nombre de pas de la chaîne
iterations <- 1e+06
# Ecart-type de la marche aléatoire (pour chaque
# paramètre)
sigma_prop <- c(2, 2, 2)
# Valeur initiale des paramètres
theta_0 <- c(runif(2, min = -100, max = 100), runif(1,
    min = 0, max = 30))
names(theta_0) <- c("Intercept", "Pente", "EcartType")
# Lancement de la chaîne de Markov
chain <- MetropolisMCMC(Y, X, sigma_prop, theta_0,
    iterations)
```

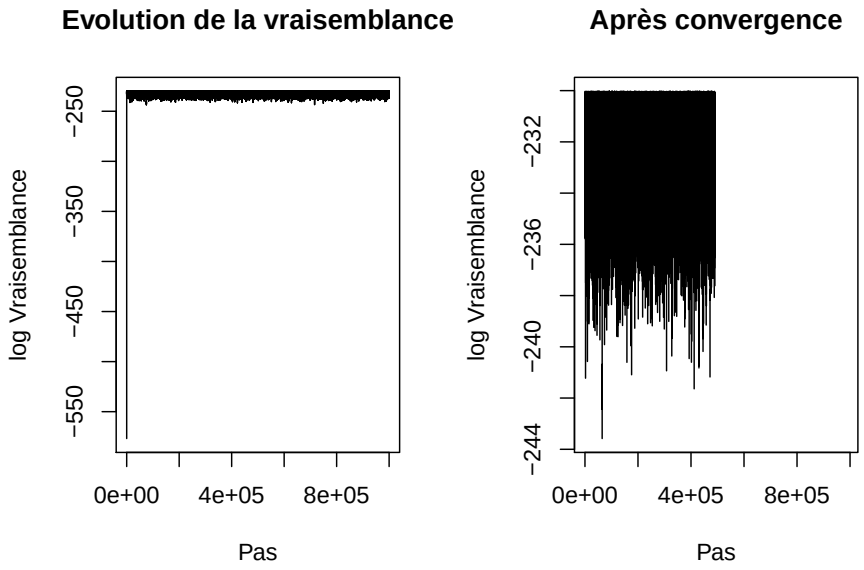
```
## ==================================================
```

4 Résultats

4.1 Vérification de la convergence

La vraisemblance des données doit augmenter pendant le préchauffage et se stabiliser ensuite. Les données de préchauffage (ici, les 10000 premières valeurs) sont éliminées.

```
# Evolution de la vraisemblance
par(mfrow = c(1, 2))
plot(chain[, length(theta_0) + 1], type = "l", main = "Evolution de la vraisemblance",
    xlab = "Pas", ylab = "log Vraisemblance")
burn_in <- 10000
plot(chain[-(1:burn_in), length(theta_0) + 1], type = "l",
    main = "Après convergence", xlab = "Pas", ylab = "log Vraisemblance")
```




```
# Taux d'acceptation de la chaine de Markov
(acceptance <- 1 - mean(duplicated(chain[-(1:burn_in),
    ])))
```

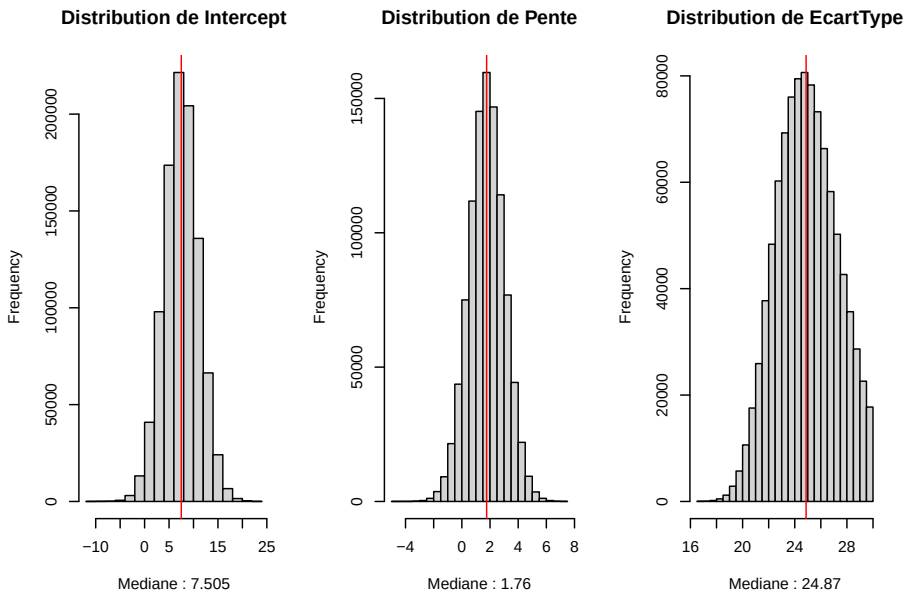
```
## [1] 0.431744
```

4.2 Distribution des paramètres *a posteriori*

La distribution des paramètres est le résultat de l'inférence.

```
par(mfrow = c(1, length(theta_0)))
for (i in 1:length(theta_0)) {
    Titre <- paste("Distribution de", names(theta_0)[i])
```

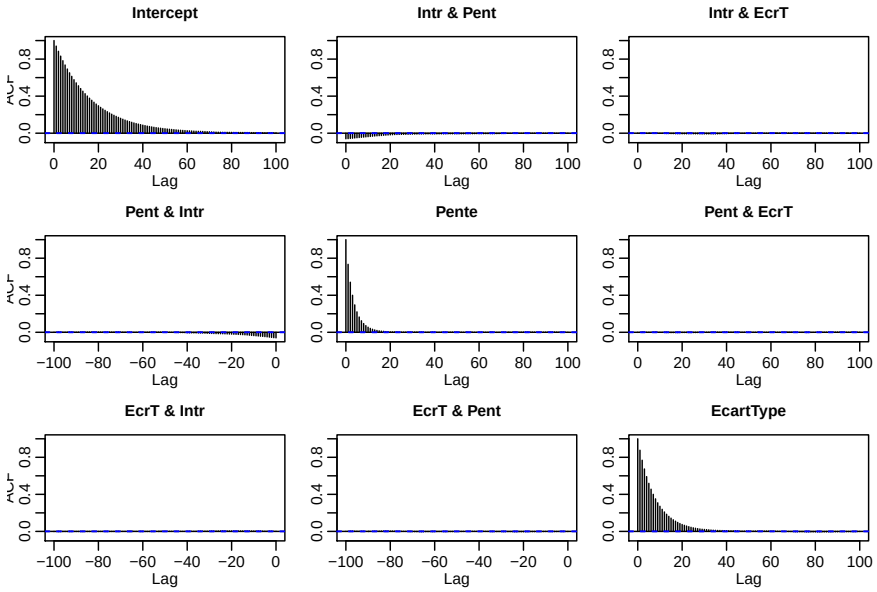
```
    Mediane <- format(median(chain[-(1:burn_in), i]),
        digits = 4)
    hist(chain[-(1:burn_in), i], xlab = paste("Mediane :",
        Mediane), main = Titre)
    abline(v = Mediane, col = "red")
}
```



4.3 Autocorrélation dans la chaîne

L'observation de l'autocorrélation entre les valeurs des paramètres permet de décider d'éclaircir les résultats de la chaîne de Markov. Ici, l'autocorrélation est faible pour l'intercept et nulle pour les autres paramètres à un décalage de 50 pas.

```
acf(chain[-(1:burn_in), -ncol(chain)], lag.max = 100)
```



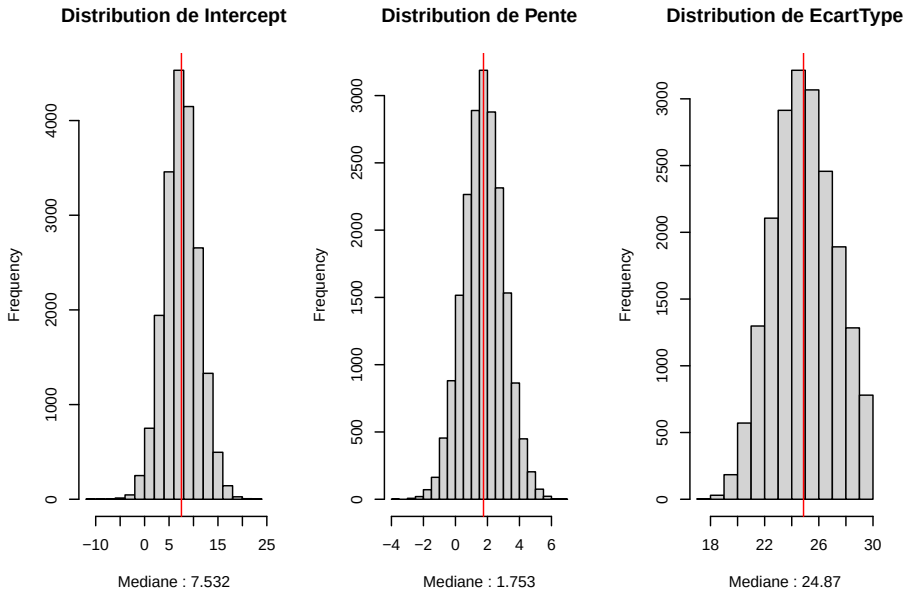
La distribution des paramètres est mieux estimée en ne retenant qu'un pas sur 50.

```
# Distribution a posteriori. Elimination du
# prechauffage et de la vraisemblance, éclaircie.
posterior <- chain[-(1:burn_in), -ncol(chain)][seq(1,
    iterations - burn_in, by = 50), ]
```

4.4 Nouvelle distribution des paramètres *a posteriori*

La distribution après éclaircie est peu modifiée.

```
par(mfrow = c(1, length(theta_0)))
for (i in 1:length(theta_0)) {
    Titre <- paste("Distribution de", names(theta_0)[i])
    Mediane <- format(median(posterior[, i]), digits = 4)
    hist(posterior[, i], xlab = paste("Mediane :",
        Mediane), main = Titre)
    abline(v = Mediane, col = "red")
}
```



Les valeurs médianes peuvent être comparées à l'estimation fréquentiste :

```
summary(flm <- lm(Y ~ X))
```

```
##
## Call:
## lm(formula = Y ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -51.296 -19.163  -1.079  20.082  52.032
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    7.483      3.483   2.148   0.0368
## X              1.760      1.232   1.429   0.1596
##
## (Intercept) *
## X
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 24.58 on 48 degrees of freedom
## Multiple R-squared:  0.04079,    Adjusted R-squared:  0.02081
## F-statistic: 2.041 on 1 and 48 DF,  p-value: 0.1596
```

4.5 Corrélation entre les paramètres

La corrélation entre les paramètres est calculée à partir de leur distribution.

```
cor(posterior)
```

```
##               Intercept       Pente     EcartType
## Intercept  1.0000000000 -0.04814976 -0.0002391943
## Pente     -0.0481497621  1.00000000 -0.0137218378
## EcartType -0.0002391943 -0.01372184  1.0000000000
```