Inferência bayesiana para modelos com efeitos aleatórios em R

Elias, Edson, Ana, PJ

8 de agosto de 2007

Resumo

Neste trabalho estudamos a implementaç ao em R da inferência em modelos de efeitos aleatórios. A inferência é sempre baseada na função de verossimilhança dos dados. Fazemos estimação pontual e intervalar por máxima verossimilhança e bayesiana. No primeiro caso utizamos a função `optim()` para encontrar as estimativas e no segundo, utilizamos a função `MCMCmetrop1R()` do pacote **MCMCpack** para simular amostras da distribuição à *posteriori*.

1 Experimentos em blocos completos casualizados

Vamos considerar nesta seção o caso em que temos um experimento em blocos completos casualizados. Neste caso temos que a observaç ao $y_{i,j}$ da variável resposta Y é referente à repetição j no bloco i, que $i = 1, 2, ..., I$ e $j = 1, 2, ..., n_i$. Podemos dizer também que temos I indivíduos sendo observados e em cada um foram feitas n_i observações.

1.1 Especificação

Vamos considerar o efeito aleatório U de bloco tem distribuição Normal:

$$U \sim Normal(\mu, \tau^2) \ ,$$

em que μ é a média do efeito de bloco e τ^2 a variância. Também consideramos que a

$$Y_{ij} \sim Normal(U_j, \sigma^2) \ ,$$

A distribuição da variável resposta condicionada no efeito de bloco é obtida integrando em relaç ao ao efeito aleatório

$$[Y] = \int [Y, U] dU$$

Além disso, como os blocos são independentes, podemos fatorar a distribuição de Y em produtos de k distribuições condicionais

$$[Y] = [Y_1][Y_2]...[Y_k] \ ,$$

e a distribuição da resposta em cada bloco é

$$[Y_j] = \int [Y_j, u_j] du_j$$

que é igual à

$$[Y_j] = \int [Y_j|u_j][u_j] du_j$$

Definindo os parâmetros em R:

```
> I <- 10
> n.sample <- 20
> mu <- 10
> sigma <- 2
> tau <- 0.5
```

Simulando uma amostra:

```
> set.seed(1)
> block <- rep(1:I, each = n.sample)
> rand.eff <- rep(rnorm(I, mu, tau), each = n.sample)
> y <- rnorm(I * n.sample, rand.eff, sigma)
```

1.2 Função de verossimilhança

A distribuição conjunta $[Y, U]$ é um produto de duas densidades normais. Podemoas escrever a verossimilhança desse modelo e obter as estimativas de máxima verossimilhança em R utilizando a função `optim()`.

Inicialmente definimos a sub-função conjunta $[Y, U]$ na escala logaritma:

```
> dconj <- function(ran, obs, mu, s.ran, s.obs) dnorm(obs, ran,
+     s.obs) * dnorm(ran, mu, s.ran)
```

Vamos desenhar o gráfico da verossimilhança em função do efeito aleatório para um valor de θ e alguns cada valor de y: A contribuição de cada observação para a verossimilhança é a área abaixo sua respectiva curva. Fazendo para as cinco primeiras observações:

```
> plot(function(x, ...) dconj(x, obs = y[1], mu = 10, s.ran = 0.5,
+     s.obs = 2), 8, 12, ylim = c(0, 0.16), xlab = "Efeito aleatorio",
+     ylab = "Contribuicao para a verossimilhanca")
> for (i in 2:5) plot(function(x, ...) dconj(x, obs = y[i], mu = 10,
+     s.ran = 0.5, s.obs = 2), 8, 12, col = i, add = T)
> legend(8, 0.16, paste("y = ", format(y[1:5], dig = 2), sep = ""),
+     col = 1:5, lty = 1)
```

A função de verossimilhança de Y é obtida integrando em relação a U. Para isso, definimos uma malha de valores razoáveis para o efeito aletaório, avaliamos a função nesses valores.

```
> lvero <- function(theta, y, id, ran) sum(log(colSums(outer(ran,
+     y, dconj, mu = theta[1], s.ran = theta[2], s.obs = theta[3])) *
+     diff(ran[1:2])))
```

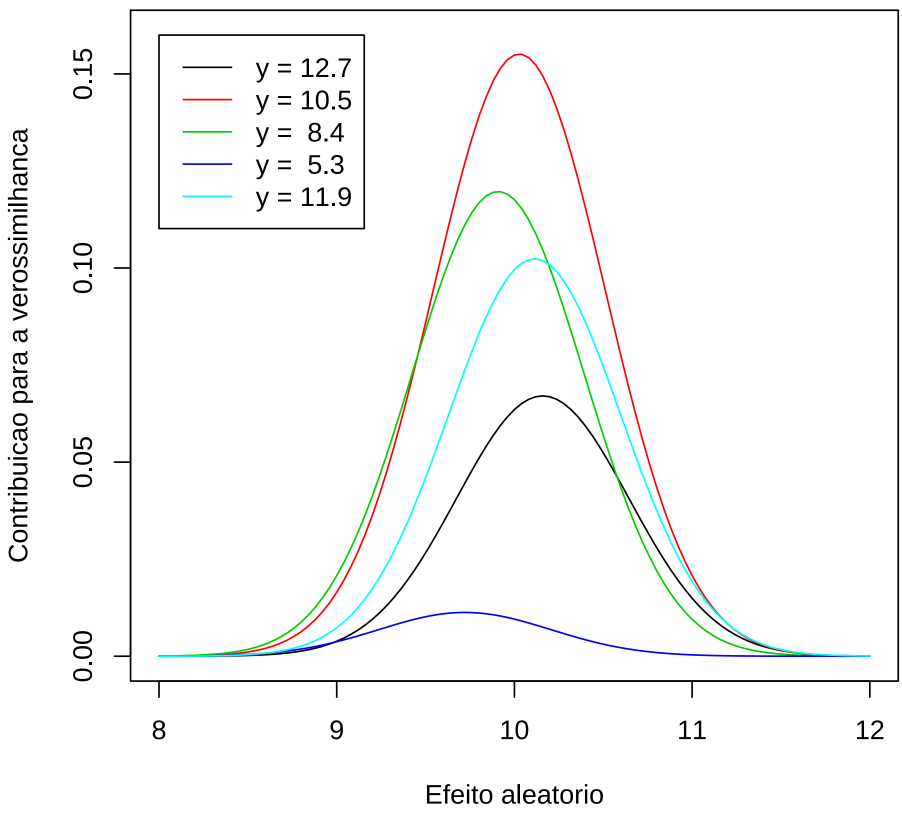
Encontrando estimativas de máxima verossimilhança, utilizando algoritmo de Nelder-Mead:

```
> gran <- seq(5, 15, len = 100)
> opt1 <- optim(c(10, 0.5, 2), lvero, y = y, id = block, ran = gran,
+     hessian = TRUE, control = list(fnscale = -1))
> opt1$conv

[1] 0

> opt1$par

[1] 10.1764872  0.8904584  1.7121407

> sum(exp(opt1$par[2:3])^2)

[1] 36.63587

> var(y)

[1] 3.743052

> sqrt(diag(solve(-opt1$hess)))

[1]  0.1364613 16.0987349  8.3733659
```

Utilizando algoritmo de BFGS:

Figura 1: Contribuição individual na verossimilhança

```
> plot(function(x, ...) dconj(x, obs = y[1], mu = 10, s.ran = 0.5,
+     s.obs = 2), 8, 12, ylim = c(0, 0.16), xlab = "Efeito aleatorio",
+     ylab = "Contribuicao para a verossimilhanca")
> for (i in 2:5) plot(function(x, ...) dconj(x, obs = y[i], mu = 10,
+     s.ran = 0.5, s.obs = 2), 8, 12, col = i, add = T)
> legend(8, 0.16, paste("y = ", format(y[1:5], dig = 2), sep = ""),
+     col = 1:5, lty = 1)
```



```
> opt1b <- optim(c(10, 0.5, 2), lvero, y = y, id = block, ran = gran,
+     hessian = TRUE, method = "B", control = list(fnscale = -1))
> opt1b$conv
```

```
[1] 0
```

```
> opt1b$par
```

```
[1] 10.1766754  0.4679942  1.8722495
```

```
> sum(exp(opt1b$par[2:3])^2)
```

```
[1] 44.83755
```

```
> var(y)
```

```
[1] 3.743052
```

```
> sqrt(diag(solve(-opt1b$hess)))
```

```
[1]   0.1364613 358.7780027  89.6813918
```

A verossimilhanąa pode ser escrita de outra maneira. Podemos considerar as observações y_i do indivíduo i tem distribuição Normal n_i variada, com vetor de médias u_i e matriz de covariância com os elementos da diagonal igual a $\sigma^2 + \tau^2$ e os elementos fora da diagonal igual a τ^2.

```
> require(mvtnorm)
```

```
[1] TRUE
```

```
> lvero2 <- function(par, y, id) {
+     n <- length(y)
+     ni <- table(id)
+     sum(sapply(unique(id), function(i) {
+         mc <- matrix(par[2]^2, ni[i], ni[i]) + diag(ni[i]) *
+             par[3]^2
+         dmvnorm(y[id == i], rep(par[1], ni[i]), mc, log = TRUE)
+     }))
+ }
> opt2 <- optim(c(10, 0.5, 2), lvero2, y = y, id = block, hessian = TRUE,
+     control = list(fnscale = -1))
> opt2$conv
```

```
[1] 0
```

```
> opt2$par
```

```
[1] 10.1769208  0.2514006  1.9132914
```

```
> sum(opt2$par[2:3]^2)
```

```
[1] 3.723886
```

```
> var(y)
```

```
[1] 3.743052
```

```
> sqrt(diag(solve(-opt2$hess)))
```

```
[1] 0.15691927 0.22218031 0.09814365
```

Utilizando a função `glmmPQL()` do pacote **MASS**.

```
> require(MASS)

[1] TRUE

> summary(aj.glmm <- glmmPQL(y ~ 1, ~1 | block, gaussian))

Linear mixed-effects model fit by maximum likelihood
 Data: NULL
  AIC BIC logLik
   NA  NA     NA

Random effects:
 Formula: ~1 | block
        (Intercept) Residual
StdDev:   0.2515545 1.913389

Variance function:
 Structure: fixed weights
 Formula: ~invwt
Fixed effects: y ~ 1
               Value Std.Error  DF t-value p-value
(Intercept) 10.17667 0.1573437 190  64.678       0

Standardized Within-Group Residuals:
       Min         Q1        Med         Q3        Max
-2.5675505 -0.6283639 -0.1256009  0.6548342  2.4318485

Number of Observations: 200
Number of Groups: 10
```

E notamos que tem resultados similares á abordagem por blocos.

2 Inferência bayesiana

Agora vamos tratar θ como variável aleatória. Vamos supor que á *priori* μ, τ^2 e σ^2 são independentes. É razoável considerar uma *priori* Normal para μ e *prioris* Gamma para τ^2 e σ^2.

Sem obter expressões anaíticas para a distribuições condicionais á *posteriori* de μ, τ^2 e σ^2, vamos simular amostras dessas distribuições. Utilizaremos o algorítmo de Metropolis. O Metropolis random walk esta implementado em C++ no pacote **MCMCpack** e pode ser utilizado via R com a função `MCMCmetrop1R()`. Nesse algoritmo, os valores propostos para todos os parâmetros são simulados de uma distribuição Normal multivariada. O vetor de médias utilizado a cada iteração é o valor atual dos parâmetros. A matriz de variância pode ser informada pelo usuário ou então é obtida do hessiano numérico calculado internamente na função. Neste caso, pode ser passado um vetor do tamanho do número de parâmetros para ajustar a variância de acordo com a taxa de aceitação.

Os parâmetros de variância são positivos e podemos simular de uma distribuição log-normal fazendo uma alteração na funcção de verossimilhança:

```
> lvero2p <- function(par, y, id) {
+     n <- length(y)
+     ni <- table(id)
+     sum(sapply(unique(id), function(i) {
+         mc <- matrix(exp(par[2])^2, ni[i], ni[i]) + diag(ni[i]) *
+             exp(par[3])^2
+         dmvnorm(y[id == i], rep(par[1], ni[i]), mc, log = TRUE)
+     }))
+ }
```

E podemos entao utilizar esta função para simular da distribuição á *posteriori*.

Não dispomos de muito conhecimento á *priori* acerca de θ, apenas sabemos que $\mu \in \Re$, τ^2 e σ^2 sâo estritamente positivos. Escolhemos as seguintes distribuições á *priori*:

$$\mu \sim Normal(0, 10000) \ ,$$

$$log(\tau^2) \sim Normal(0, 9) \ e$$

$$log(\sigma^2) \sim Normal(0, 9) \ .$$

Vizualizando em R:

O logaritmo da distribuição á priori em R pode ser escrito como:

```
> p.theta <- function(mu, tau2, sigma2) dnorm(mu, 0, 100, TRUE) +
+     dnorm(log(tau2), 0, 1, TRUE) + dnorm(log(sigma2), 0, 3, TRUE)
```

O logaritmo da distribuição á posteriori então é:

```
> dpost <- function(theta, y, id) p.theta(theta[1], exp(theta[2])^2,
+     exp(theta[3])^2) + lvero2p(theta, y, id)
```

Vizualização das distribuições á *posteriori* condicionais.

```
> par(mfrow = c(1, 3), mar = c(3, 3, 1, 1), mgp = c(1.7, 0.5, 0))
> plot(function(x) sapply(x, function(a) dpost(c(a, log(0.5), log(2)),
+     y, block)), 9, 11, n = 31, xlab = expression(mu), ylab = expression(L(mu/y,
+     tau, sigma)))
> plot(function(x) sapply(x, function(a) dpost(c(10, a, log(2)),
+     y, block)), -2, 0, n = 31, xlab = expression(exp(tau)), ylab = expression(L(exp(tau)/y,
+     mu, sigma)))
> plot(function(x) sapply(x, function(a) dpost(c(10, log(0.5),
+     a), y, block)), 0, 2, n = 31, xlab = expression(exp(sigma)),
+     ylab = expression(L(exp(sigma)/y, mu, tau)))
```

Uma idéia para a matriz de variância da distribuição da proposta é considerar a a matriz de variância estimada pela inversa da matriz hessiana, que pode ser obtida numericamente com a função `optim()`.

```
> opt2b <- optim(c(10, -1, 1), dpost, y = y, id = block, hessian = TRUE,
+     control = list(fnscale = -1))
> str(opt2b)

List of 6
 $ par        : num [1:3] 10.177 -0.665  0.643
 $ value      : num -425
 $ counts     : Named int [1:2] 78 NA
  ..- attr(*, "names")= chr [1:2] "function" "gradient"
 $ convergence: int 0
 $ message    : NULL
 $ hessian    : num [1:3, 1:3] -22.43898   0.00615   0.00421   0.00615 -10.05507 ...

> round(mcov <- solve(-opt2b$hess), 4)

       [,1]    [,2]    [,3]
[1,] 0.0446  0.0000  0.0000
[2,] 0.0000  0.0995 -0.0001
[3,] 0.0000 -0.0001  0.0026

> round(v.metrop <- diag(c(1.5, 1.7, 1)) %*% mcov %*% diag(c(1.5,
+     1.7, 1)), 4)
```

Figura 2: Distribuições á *priori*. O terceiro gráfico é um close so segundo

```
> par(mfrow = c(1, 3), mar = c(3, 3, 1, 1), mgp = c(1.7, 0.5, 0))
> plot(function(x) dnorm(x, 0, 100), -300, 300, xlab = expression(mu),
+     ylab = expression(p(mu)))
> plot(function(x) dnorm(log(x), 0, 3), 0, 300, xlab = expression(tau),
+     ylab = expression(p(tau)))
> plot(function(x) dnorm(log(x), 0, 3), 0, 5, xlab = expression(tau),
+     ylab = expression(p(tau)))
```

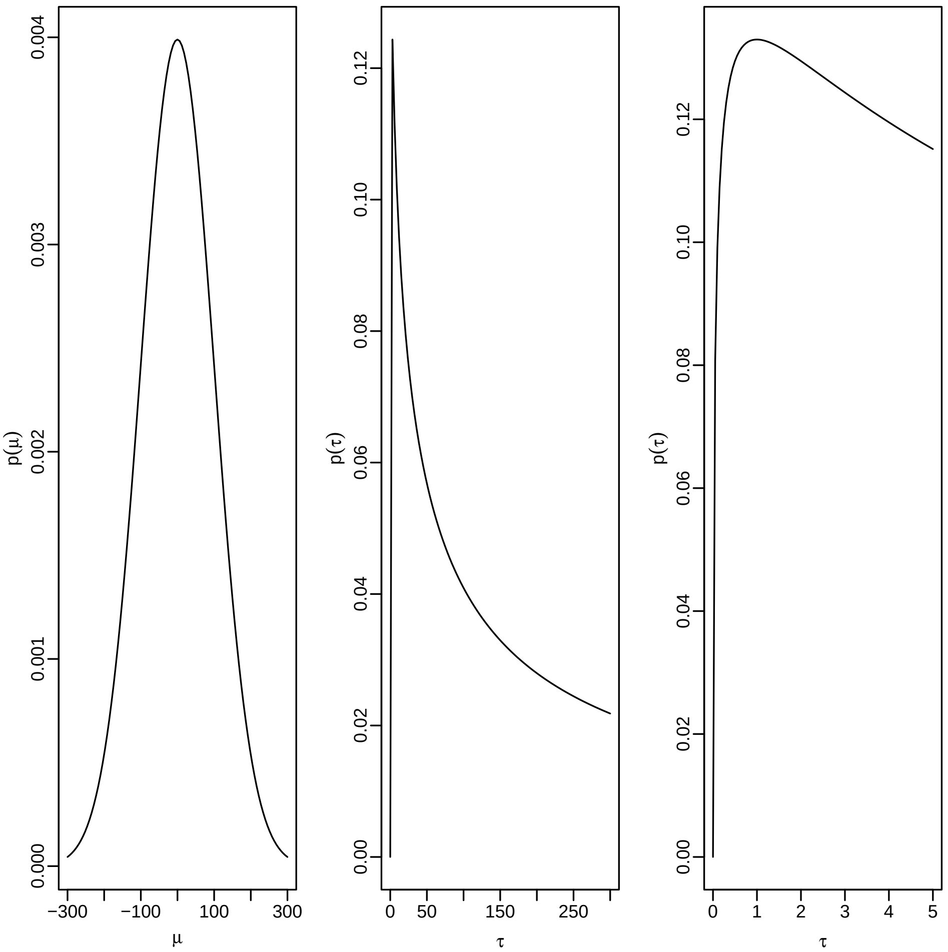
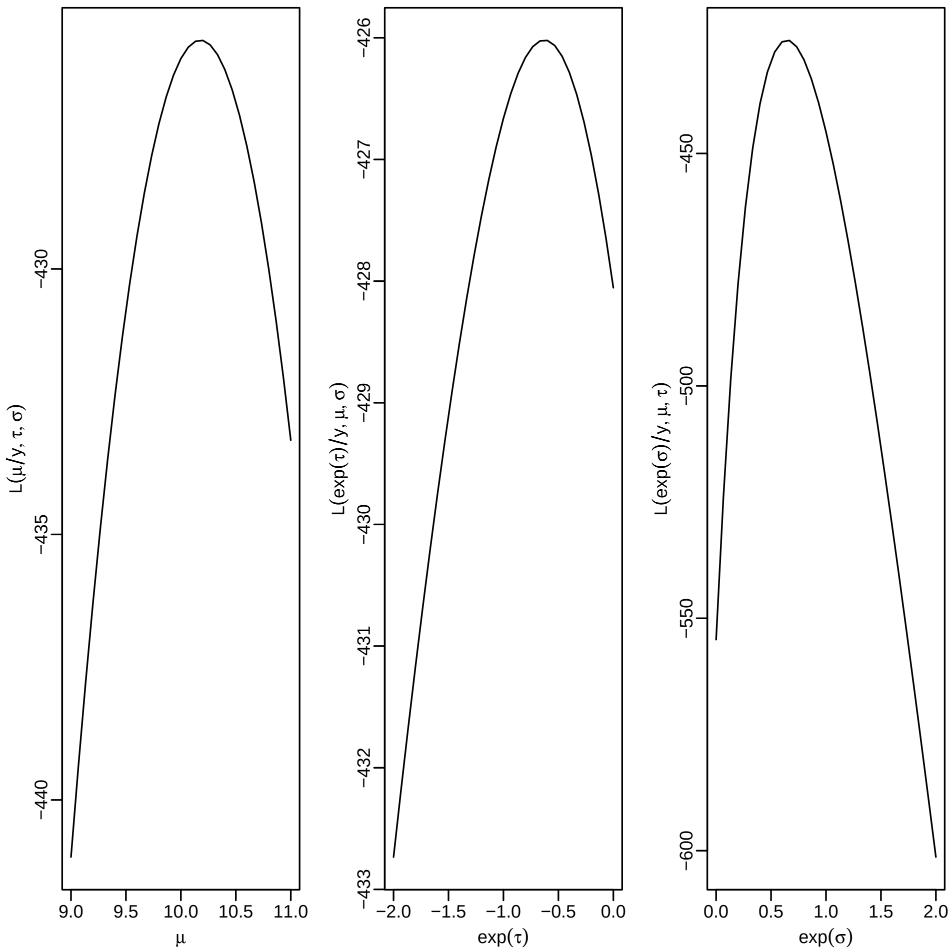


Figura 3: Vizualizando a distribuições á posteriori condicionais

```
> par(mfrow = c(1, 3), mar = c(3, 3, 1, 1), mgp = c(1.7, 0.5, 0))
> plot(function(x) sapply(x, function(a) dpost(c(a, log(0.5), log(2)),
+     y, block)), 9, 11, n = 31, xlab = expression(mu), ylab = expression(L(mu/y,
+     tau, sigma)))
> plot(function(x) sapply(x, function(a) dpost(c(10, a, log(2)),
+     y, block)), -2, 0, n = 31, xlab = expression(exp(tau)), ylab = expression(L(exp(tau)/y,
+     mu, sigma)))
> plot(function(x) sapply(x, function(a) dpost(c(10, log(0.5),
+     a), y, block)), 0, 2, n = 31, xlab = expression(exp(sigma)),
+     ylab = expression(L(exp(sigma)/y, mu, tau)))
```



```
       [,1]    [,2]    [,3]
[1,] 0.1003  0.0001  0.0000
[2,] 0.0001  0.2874 -0.0002
[3,] 0.0000 -0.0002  0.0026
```

Neste caso, utilizar o argumento `V=v.metrop`, é o mesmo que utilizer `V=NULL` e `tune=c(1.5,1.5,0.7)`. Simulando da distribuição á posteriori:

```
> library(MCMCpack)
> post <- MCMCmetrop1R(dpost, theta.init = c(10, log(0.5), log(2)),
+     100, 3000, 10, verbose = 500, V = v.metrop, y = y, id = block)

MCMCmetrop1R iteration 1 of 3100
theta =
  -0.01414
   0.00000
  -0.01414
function value = -426.03902
Metropolis acceptance rate = 0.00000

MCMCmetrop1R iteration 501 of 3100
theta =
  10.26151
  -1.02940
   0.62403
function value = -426.05409
Metropolis acceptance rate = 0.28743

MCMCmetrop1R iteration 1001 of 3100
theta =
  10.27093
  -1.55244
   0.67731
function value = -428.77488
Metropolis acceptance rate = 0.29870

MCMCmetrop1R iteration 1501 of 3100
theta =
  10.08112
  -0.97336
   0.74496
function value = -427.64203
Metropolis acceptance rate = 0.30779

MCMCmetrop1R iteration 2001 of 3100
theta =
   9.86129
  -1.02817
   0.62181
function value = -427.60040
Metropolis acceptance rate = 0.31734

MCMCmetrop1R iteration 2501 of 3100
theta =
  10.00326
  -0.20807
   0.66498
```

```
function value = -426.55987
Metropolis acceptance rate = 0.31467

MCMCmetrop1R iteration 3001 of 3100
theta =
  10.39963
  -0.38801
   0.58255
function value = -426.76886
Metropolis acceptance rate = 0.30890



@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate was 0.30903
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Visualizando as amostras da distribuição á *posteriori*.

Calculando quantidades de interesse:

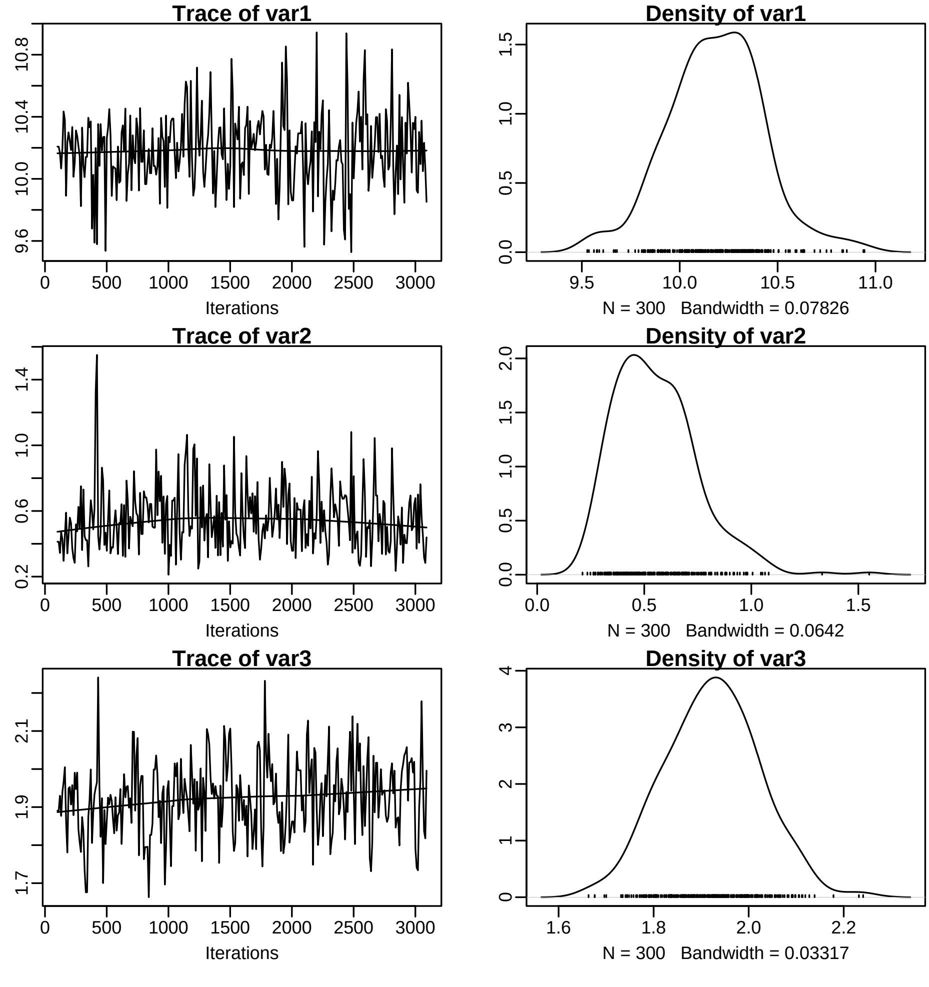
```
> summary(postb)

Iterations = 101:3091
Thinning interval = 10
Number of chains = 1
Sample size per chain = 300

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

        Mean      SD Naive SE Time-series SE
[1,] 10.1830 0.24374 0.014072       0.015243
[2,]  0.5545 0.19348 0.011171       0.013431
[3,]  1.9234 0.09919 0.005727       0.008697

2. Quantiles for each variable:

       2.5%     25%     50%     75%   97.5%
var1 9.6662 10.0332 10.1920 10.3427 10.7036
var2 0.2776  0.4117  0.5262  0.6656  0.9802
var3 1.7378  1.8599  1.9262  1.9911  2.1093
```

HPD intervalo

```
> ic <- HPDinterval(postb)
> ic

         lower      upper
var1 9.5889554 10.6367461
var2 0.2486845  0.9465142
var3 1.7408136  2.1133505
attr(,"Probability")
[1] 0.95

> apply(postb, 2, median)

     var1      var2      var3
10.191998  0.526184  1.926199
```

Figura 4: Traço e densidades das amostras da distribuição á posteriori

```
> postb <- post
> postb[, 2] <- exp(post[, 2])
> postb[, 3] <- exp(post[, 3])
> par(mar = c(3, 3, 1, 1), mgp = c(1.7, 0.5, 0))
> plot(postb)
```



2.1 Utilizando o princípio de aumento de dados

Uma alternativa que pode ser utilizada em lugar de integrar os efeitos aleatórios, é utilizar o princípio de aumento de dados e simulá-los. Neste caso temos que escrever uma função para $[y/u, \tau^2, \sigma^2]$, que recebe como argumentos o vetor u, $log(\tau)$ e $log(\sigma)$. Esta função deve retornar $[\theta][u/tau][y/u, sigma]$.

```
> post.aug <- function(pars, y, id) {
+     nb <- length(unique(id))
+     mu <- mean(pars[1:nb])
+     p.theta(mu, exp(pars[nb + 1])^2, exp(pars[nb + 2])^2) + sum(dnorm(pars[1:nb],
+         mu, exp(pars[nb + 1]), log = TRUE)) + sum(dnorm(y, pars[id],
+         exp(pars[nb + 2]), log = TRUE))
+ }
> opt3 <- optim(c(rep(10, 10), 1, 1), post.aug, y = y, id = block,
+     hess = T, control = list(fnscale = -1))
> opt3$conv
```

```
[1] 1
```

```
> c(mean(opt3$par[1:10]), exp(opt3$par[11:12]))
```

```
[1] 10.3160490  0.3977733  1.9229344
```

Matriz de variancia para algoritmo de Metropolis:

```
> v3.metrop <- diag(c(rep(0.6, 10), 1, 1)) %*% solve(-opt3$hess) %*%
+     diag(c(rep(0.6, 10), 1, 1))
> 100 * round(v3.metrop, 3)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
 [1,]  3.9  0.1  0.4 -0.3 -0.1  1.2  1.5  0.2 -0.3   0.3  -3.5   0.1
 [2,]  0.1  3.6  0.3  0.7  0.6 -0.1 -0.2  0.5  0.7   0.4   2.0   0.0
 [3,]  0.4  0.3  3.4  0.3  0.3  0.5  0.5  0.3  0.3   0.4  -0.4   0.0
 [4,] -0.3  0.7  0.3  4.5  1.1 -0.9 -1.4  0.7  1.4   0.4   5.6  -0.1
 [5,] -0.1  0.6  0.3  1.1  3.9 -0.5 -0.8  0.6  1.1   0.4   3.9  -0.1
 [6,]  1.2 -0.1  0.5 -0.9 -0.5  4.9  2.5  0.0 -0.9   0.3  -6.6   0.2
 [7,]  1.5 -0.2  0.5 -1.4 -0.8  2.5  6.4 -0.1 -1.4   0.3  -9.5   0.3
 [8,]  0.2  0.5  0.3  0.7  0.6  0.0 -0.1  3.5  0.7   0.4   1.7   0.0
 [9,] -0.3  0.7  0.3  1.4  1.1 -0.9 -1.4  0.7  4.5   0.4   5.8  -0.1
[10,]  0.3  0.4  0.4  0.4  0.4  0.3  0.3  0.4  0.4   3.4   0.2   0.0
[11,] -3.5  2.0 -0.4  5.6  3.9 -6.6 -9.5  1.7  5.8   0.2  30.4  -0.6
[12,]  0.1  0.0  0.0 -0.1 -0.1  0.2  0.3  0.0 -0.1   0.0  -0.6   0.3
```

```
> post3 <- MCMCmetrop1R(post.aug, c(rep(10, 10), exp(0.5), exp(2)),
+     1000, 10000, 10, verbose = 1000, V = v3.metrop, y = y, id = block)
```

```
MCMCmetrop1R iteration 1 of 11000
theta =
   0.00000
  -0.00000
   0.00000
  -0.00000
-1713.31075
  -0.00000
   0.00000
  -0.00000
  -0.01414
   0.00000
  -0.01414
```

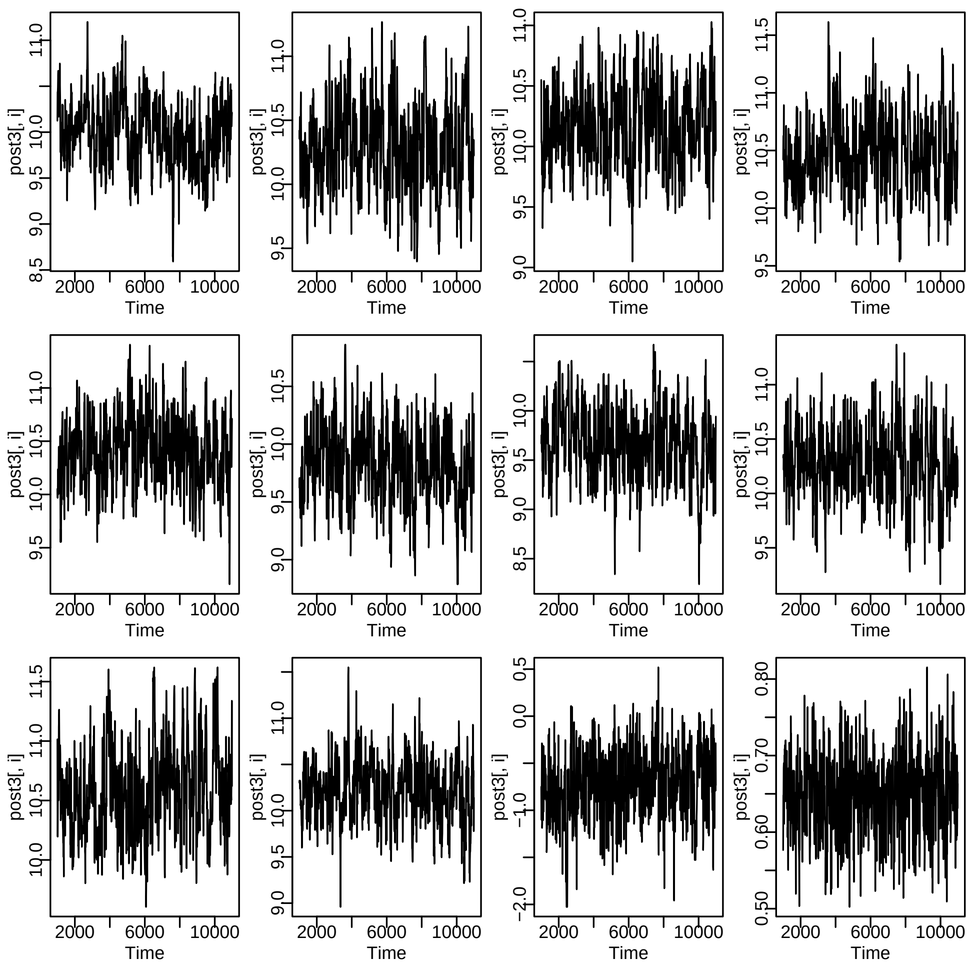
```
  -0.01414
function value = -1713.31075
Metropolis acceptance rate = 0.00000

MCMCmetrop1R iteration 1001 of 11000
theta =
  10.12434
  10.41838
  10.54969
  10.42146
  10.05218
   9.62504
   9.67071
  10.33090
  11.01454
  10.30293
  -0.75050
   0.57655
function value = -426.38045
Metropolis acceptance rate = 0.29071

MCMCmetrop1R iteration 2001 of 11000
theta =
  10.00829
  10.51529
  10.73715
  10.28088
  10.31178
  10.18514
  10.49596
  10.34145
  10.58750
   9.79961
  -1.14135
   0.67576
function value = -428.98403
Metropolis acceptance rate = 0.24638

MCMCmetrop1R iteration 3001 of 11000
theta =
  10.27119
  10.03806
  10.56863
  10.29362
  10.72731
  10.27141
   9.32997
  10.30782
  10.61740
  10.35125
  -0.24438
   0.62977
function value = -428.83046
Metropolis acceptance rate = 0.22393

MCMCmetrop1R iteration 4001 of 11000
```

```
theta =
   9.49603
  10.13323
  10.10406
  11.00363
  10.62964
   9.85903
   9.40657
  10.28468
  11.42737
   9.69555
  -0.19002
   0.58324
function value = -430.70106
Metropolis acceptance rate = 0.22169

MCMCmetrop1R iteration 5001 of 11000
theta =
  10.27759
   9.85215
   9.90916
  10.96316
  10.22471
   9.68595
   9.69962
   9.98951
  10.25767
  10.02058
  -0.78318
   0.65902
function value = -426.99995
Metropolis acceptance rate = 0.22116

MCMCmetrop1R iteration 6001 of 11000
theta =
  10.35227
   9.99328
   9.90485
  10.46453
  10.35436
  10.38270
   9.53822
  10.27179
   9.92280
   9.67681
  -1.11879
   0.68664
function value = -429.61415
Metropolis acceptance rate = 0.21930

MCMCmetrop1R iteration 7001 of 11000
theta =
   9.81911
  10.58465
   9.56815
  10.42309
```

```
   10.35070
    9.67833
    9.67584
   10.41247
   10.86629
   10.41437
   -0.55372
    0.64041
function value = -426.40410
Metropolis acceptance rate = 0.22025

MCMCmetrop1R iteration 8001 of 11000
theta =
    9.94505
    9.99481
    9.95141
   10.62551
   10.13815
    9.69882
   10.04120
   10.30957
   10.43784
   10.76454
   -1.04278
    0.65954
function value = -426.47617
Metropolis acceptance rate = 0.21985

MCMCmetrop1R iteration 9001 of 11000
theta =
    9.75849
    9.61678
    9.77571
    9.95850
   10.34015
    9.78959
    9.40833
   10.18448
   10.21287
    9.85227
   -1.10662
    0.66010
function value = -426.59667
Metropolis acceptance rate = 0.22031

MCMCmetrop1R iteration 10001 of 11000
theta =
   10.52135
    9.93791
   10.06254
   10.39936
   10.57509
    9.19575
    8.83181
    9.63382
   11.39611
```

Figura 5: Efeitos aleatórios simulados e parâmetros de variância

```
> par(mfrow = c(3, 4), mar = c(2, 2, 1, 0.5), mgp = c(1.2, 0.2,
+     0))
> for (i in 1:ncol(post3)) plot.ts(post3[, i])
```



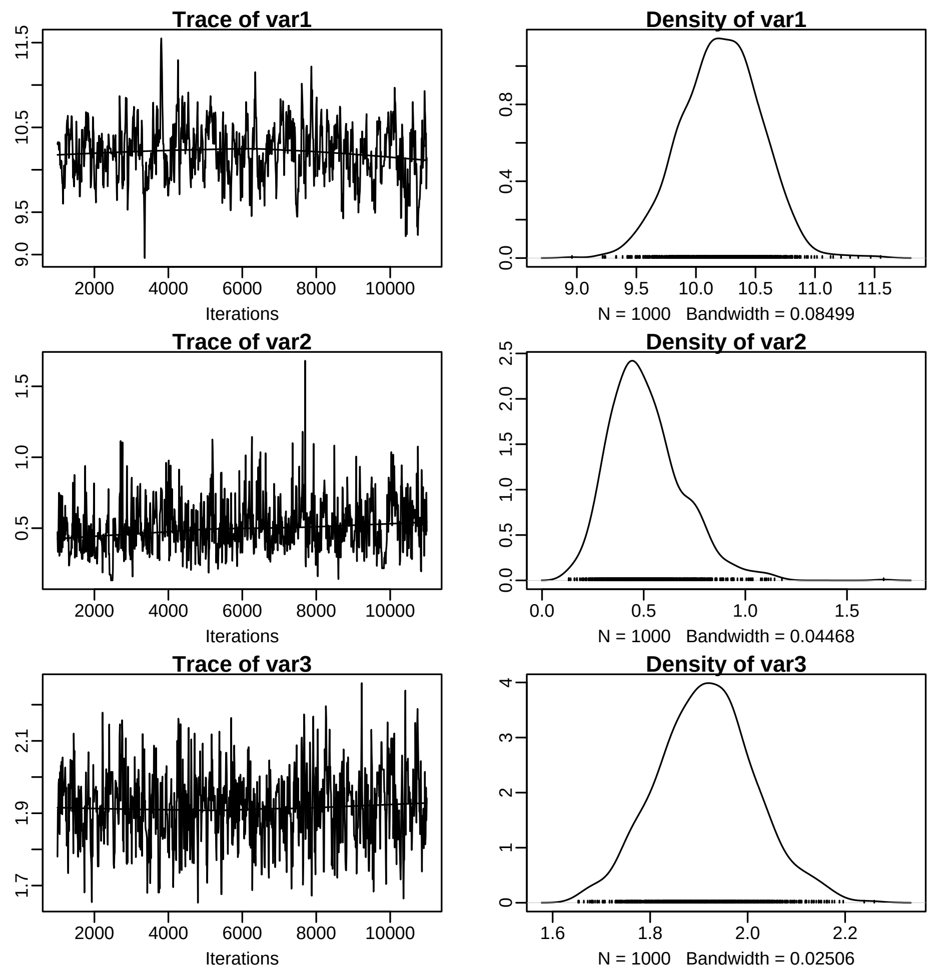
```
  10.32445
  -0.19328
   0.72353
function value = -434.08929
Metropolis acceptance rate = 0.21738



@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The Metropolis acceptance rate was 0.21900
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
> post3b <- post3[, 10:12]
> post2b <- colMeans(post3[, 1:10])
> post3b[, 2] <- exp(post3[, 11])
> post3b[, 3] <- exp(post3[, 12])
```

Estatísticas dos parâmetros de interesse:

Figura 6: Traç e densidade das amostras á posteriori dos parâmetros de interesse

```
> par(mar = c(3, 3, 1, 1), mgp = c(1.7, 0.5, 0))
> plot(post3b)
```



```
> summary(post3b)

Iterations = 1001:10991
Thinning interval = 10
Number of chains = 1
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

       Mean     SD Naive SE Time-series SE
[1,] 10.211 0.3342 0.010568       0.030046
[2,]  0.511 0.1822 0.005763       0.011075
[3,]  1.915 0.0991 0.003134       0.004919

2. Quantiles for each variable:

       2.5%     25%    50%     75%   97.5%
var1 9.5567 10.0015 10.204 10.4293 10.8306
var2 0.2201  0.3833  0.482  0.6082  0.9372
var3 1.7305  1.8484  1.913  1.9745  2.1205

> HPDinterval(post3b)

         lower     upper
var1 9.5733493 10.834197
var2 0.1718794  0.857262
var3 1.7305092  2.120564
attr(,"Probability")
[1] 0.95

> apply(post3b, 2, median)

      var1       var2       var3
10.2042354  0.4820479  1.9129738
```