Linear mixed-effects models

Peter Solymos and Subhash Lele

July 16, 2016 — Madison, WI — NACCB Congress

Continuous case with continuous measurement error (Neyman-Scott problem)

Some historical comments: Fisher and Pearson arguments about MOM and MLE; Fisher and Neyman arguments about testing and p-values. This is an example where Fisher is wrong about consistency and efficiency of the MLE.

Motivation

Animal breeding example (Linear mixed models, LMM): Consider the model underlying one way analysis of variance (ANOVA).

$Y_{ij} = \mu + \alpha_i + \varepsilon_{ij}$, $i = 1, 2, \ldots, n$ and $j = 1, 2$, $\varepsilon_{ij} \sim Normal(0, \sigma^2_\varepsilon)$.

There are two offsprings per individual. We want to know something about the genetic potential of the individuals so that we can turn some into hamburgers and some can have fun as studs/dams.

Parameters

These are the unknown quantities that we want to learn about from the data. In this model, the parameters are $(\mu, \alpha_1, \alpha_2, \ldots, \alpha_n, \sigma^2_\varepsilon)$:

- number of parameters: $n + 2$,
- number of observations: $2n$.

Question

It is easy to write down the likelihood function. What happens if we compute MLE for these parameters? These are familiar quantities from ANOVA, except for the estimate of the variance.

$$\hat{\mu} = \frac{1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{2} Y_{ij}$$

$$\hat{\alpha}_i = \frac{1}{2}(Y_{i1} + Y_{i2})$$

$$\hat{\sigma}^2_\varepsilon = \frac{1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{2} (Y_{ij} - \hat{\mu} - \hat{\alpha}_i)^2$$

One of the most important results from Neyman and Scott (1949) is that as the sample size increases, $\hat{\sigma}^2_\varepsilon \to \sigma^2_\varepsilon/2$. It is almost obvious that we cannot estimate consistently (although it is an unbiased estimator).

Increasing the sample size does not guarantee good estimators. What is important is that the information in the sample about the parameter should converge to infinity.

In this model, the number of parameters is increasing at the same rate as the sample size (the information). Hence we have limited information about any particular parameter. We are spending the information profligately. This kind of situation is **not unusual in practice**.

- *Logistic regression and multi-center clinical studies*: Each hospital has only a few patients and we want to combine information across the hospitals.
- *Combining data across large geographic areas in abundance surveys*: Random effect for the area and the effect of the covariates in the Poisson regression.

What can we do? We need more information but more data are not going to give us more information. In mathematics the solution always exists: *assume*!

These assumptions should, in effect, reduce the number of parameters. Hopefully we can reduce them to the level that information increases sufficiently faster than the number of parameters. Usually we make assumptions so that the final number of parameters is unchanging with the sample size but this is not necessary.

Smoothness assumptions:

- regression assumption,
- random effects assumption.

WARNING This has nothing to do with the Bayesian thinking. These are simply modeling assumptions. The effect of these assumptions (e.g. distribution of the latent variable) does not go away as we increase the sample size. On the other hand, as we have seen repeatedly, the effect of the prior (which is also an assumption) vanishes as the information in the data increases.

There is no such thing as "Bayesian model" or "Frequentist model". There are stochastic models; there are deterministic models; there are descriptive models. Some Bayesians claim that specification of the prior on the parameters is on the same level as specifying a stochastic model for the data. Hence they consider all hierarchical models as "Bayesian models".

We do not agree with this. As we have pointed out, the effect of the modeling assumption does not vanish as we increase the sample size, whereas the effect of the prior does.

Unknown quantities in the stochastic models: We have come across two different types of unknown quantities in the hierarchical models: **latent variables** and **parameters**.

- *Parameters*: These are quantities in the model that can be estimated with perfect certainty as the information in the data increases to infinity.
- *Latent variables*: No amount of data can determine these with certainty. The uncertainty does not go to zero.

Analogy with estimation and prediction in time series or regression: If we have large amount of data (and, the model is correct), then the standard error for the parameter estimates goes to zero but the prediction error does not. Latent variables in the hierarchical models are similar to the prediction of unobserved time points.

Imposing a distributional assumption is qualitatively same as imposing regression model. This is not a 'prior distribution' of any kind. This is a misleading terminology commonly used in the Bayesian literature.

Prior distributions are smoothness assumptions on the parameters and their effect goes to zero as the information increases.

```
set.seed(1234)
n <- 100
m <- 1
mu <- 2.5
```

```
sigma_sq <- 0.2^2
eps <- rnorm(n * m, mean = 0, sd = sqrt(sigma_sq))
tau_sq <- 0.5^2
alpha <- rnorm(n, mean = 0, sd = sqrt(tau_sq))
Y <- mu + alpha + eps
dim(Y) <- c(n, m)
summary(Y)
```

Data generation

```
##        V1
##  Min.   :1.098
##  1st Qu.:2.171
##  Median :2.470
##  Mean   :2.489
##  3rd Qu.:2.816
##  Max.   :3.877
```

Without a smoothing assumption:

```
library(dclone)
```

```
## Loading required package: coda

## Loading required package: parallel

## Loading required package: Matrix

## dclone 2.1-1      2016-01-11
```

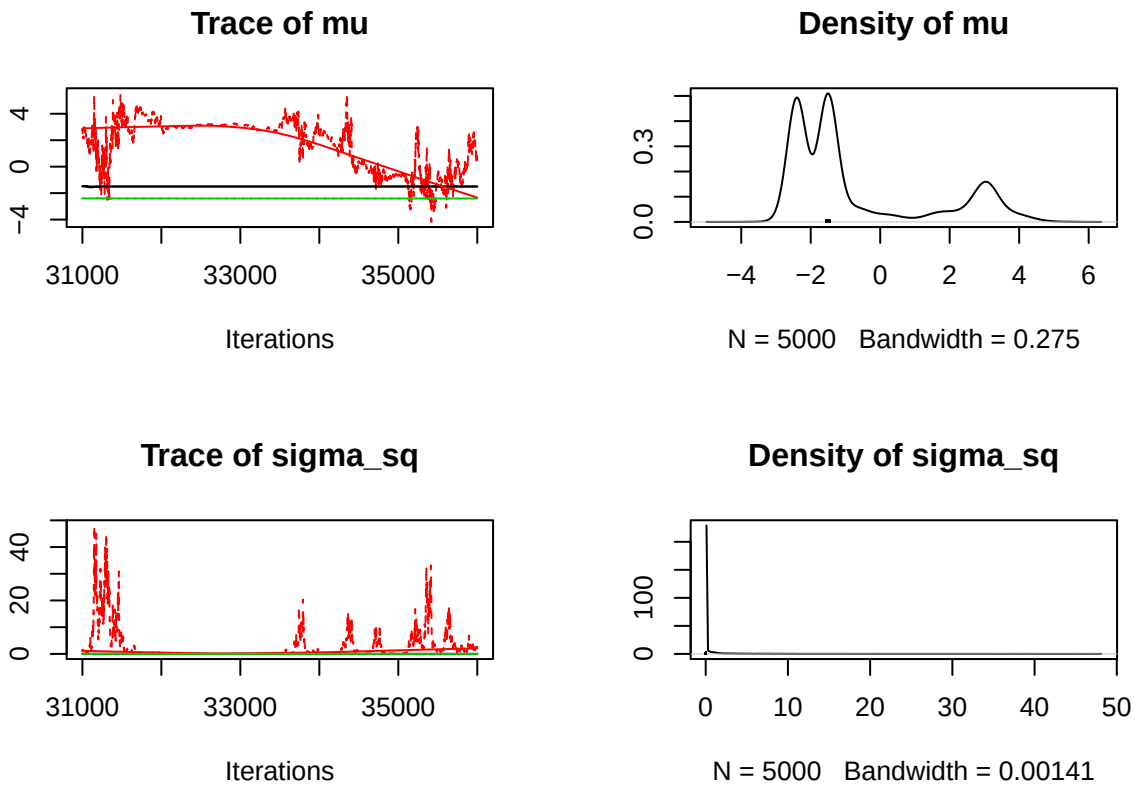
```
model <- custommodel("model {
    for (i in 1:n) {
        for (j in 1:m) {
            Y[i,j] ~ dnorm(mu + alpha[i], 1 / sigma_sq)
        }
        alpha[i] ~ dnorm(0, 0.001)
    }
    log_sigma ~ dnorm(0, 0.001)
    sigma_sq <- exp(log_sigma)^2
    mu ~ dnorm(0, 0.1)
}")
dat <- list(Y = Y, n = n, m = m)
fit <- jags.fit(data = dat, params = c("mu", "sigma_sq","alpha"),
    model = model, n.update = 30000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 102
##    Total graph size: 713
##
## Initializing model
```

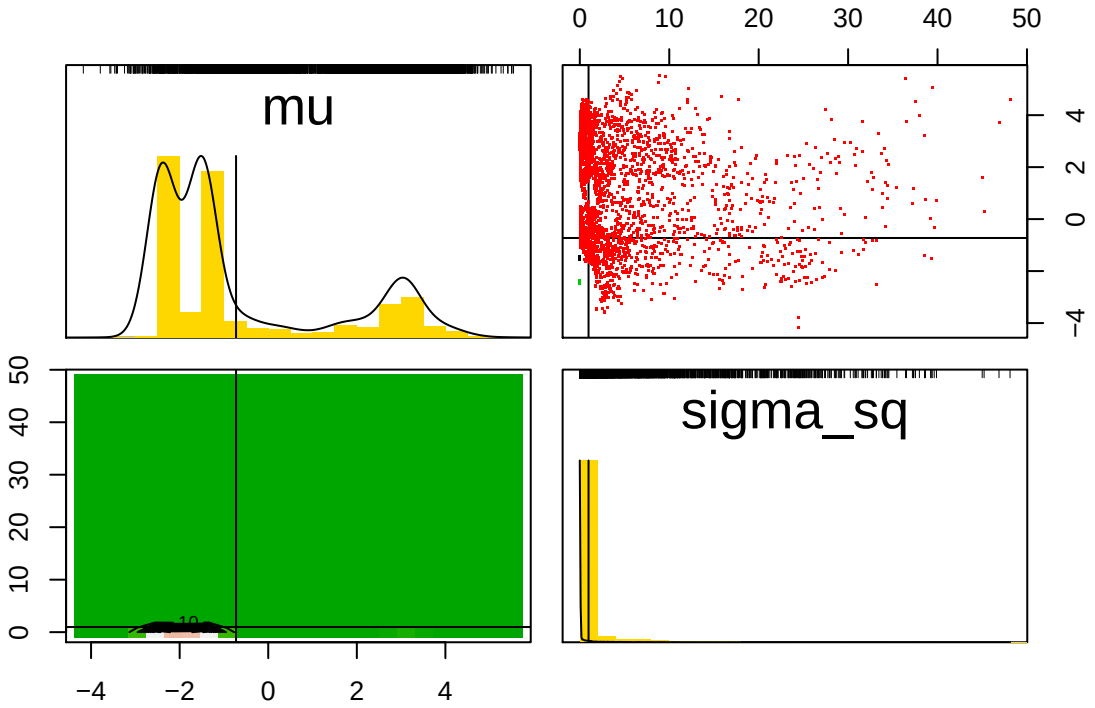
```r
summary(fit[,c("mu","sigma_sq")])
```

```
##
## Iterations = 31001:36000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##              Mean    SD Naive SE Time-series SE
## mu       -0.7265 2.073  0.01693         0.2361
## sigma_sq  0.9738 3.724  0.03041         0.3686
##
## 2. Quantiles for each variable:
##
##                2.5%        25%        50%      75%  97.5%
## mu       -2.410e+00 -2.403e+00 -1.500e+00 -0.02392  3.671
## sigma_sq  1.061e-13  1.295e-10  2.234e-07  0.01220 11.177
```

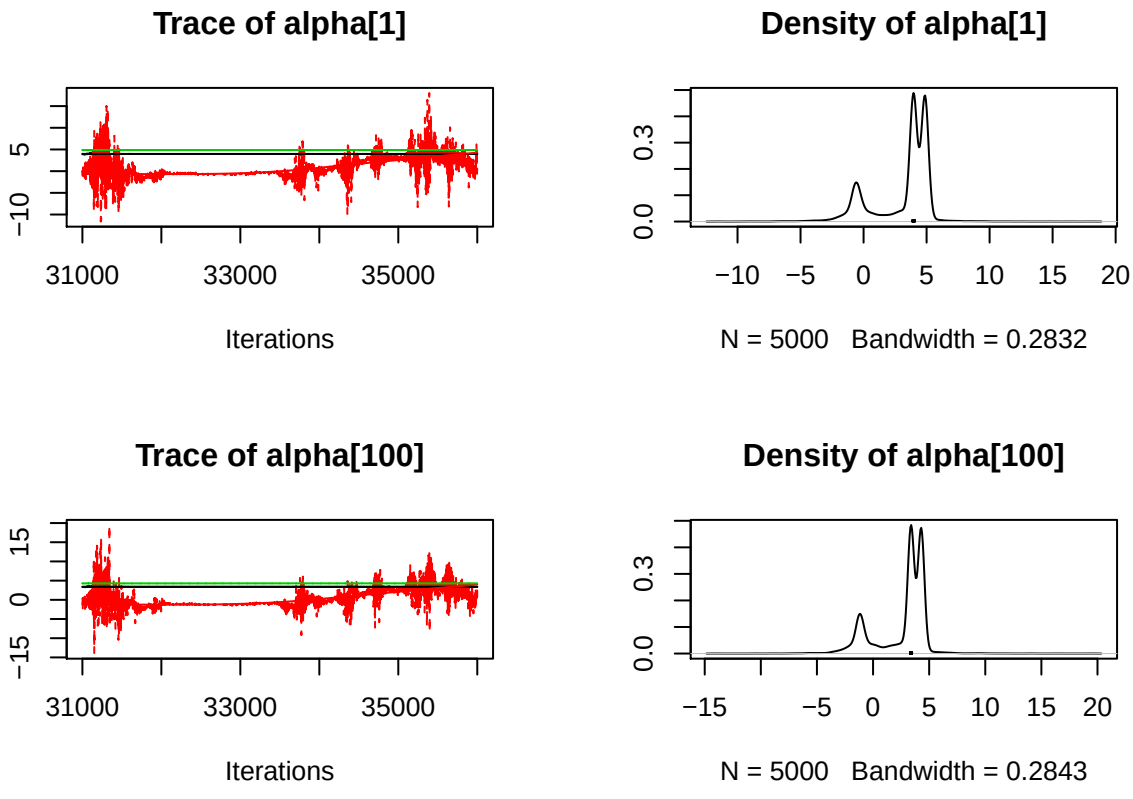
```r
plot(fit[,c("mu","sigma_sq")])
```




```r
pairs(fit[,c("mu","sigma_sq")])
```




```
plot(fit[,c("alpha[1]","alpha[100]")])
```



With smoothing assumption:

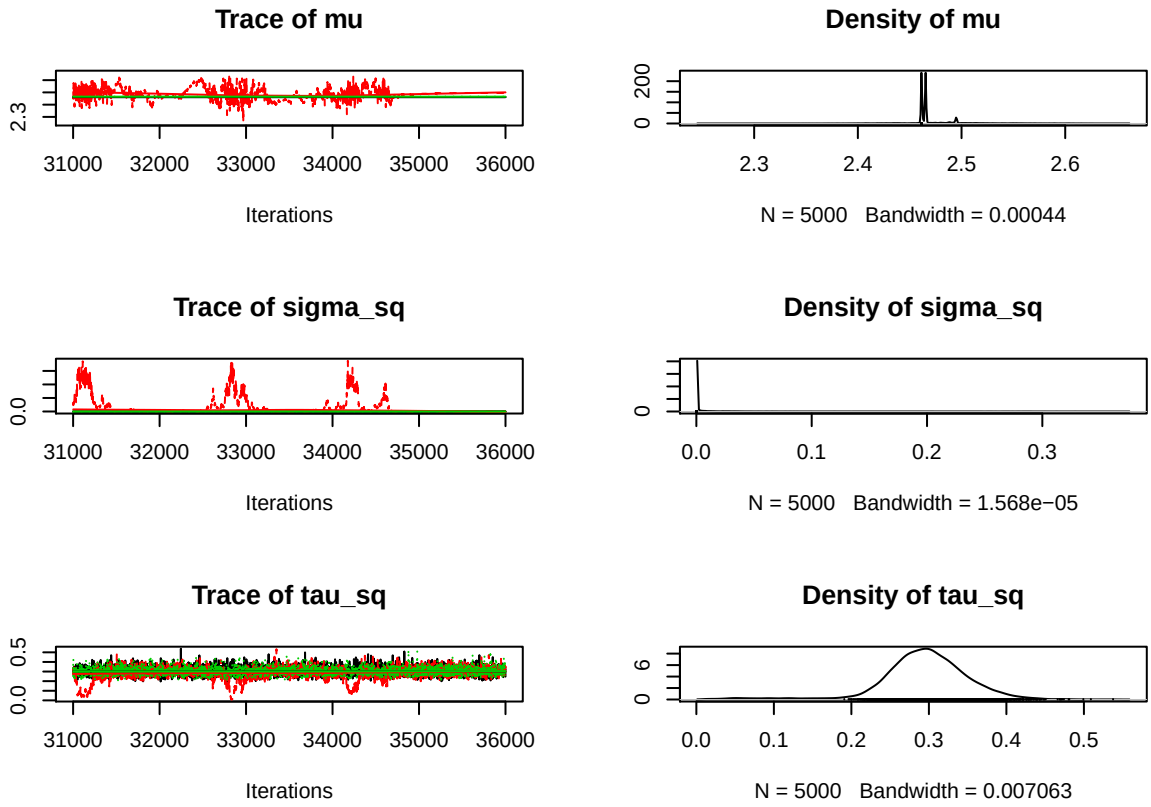
```
library(dclone)
model <- custommodel("model {
    for (i in 1:n) {
        for (j in 1:m) {
            Y[i,j] ~ dnorm(mu + alpha[i], 1 / sigma_sq)
        }
        alpha[i] ~ dnorm(0, 1 / tau_sq)
    }
    log_sigma ~ dnorm(0, 0.001)
    sigma_sq <- exp(log_sigma)^2
    mu ~ dnorm(0, 0.1)
    log_tau ~ dnorm(0, 0.001)
    tau_sq <- exp(log_tau)^2
}")
dat <- list(Y = Y, n = n, m = m)
fit <- jags.fit(data = dat,
    params = c("mu", "sigma_sq", "tau_sq", "alpha"),
    model = model, n.update = 30000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 103
##    Total graph size: 921
##
## Initializing model
```

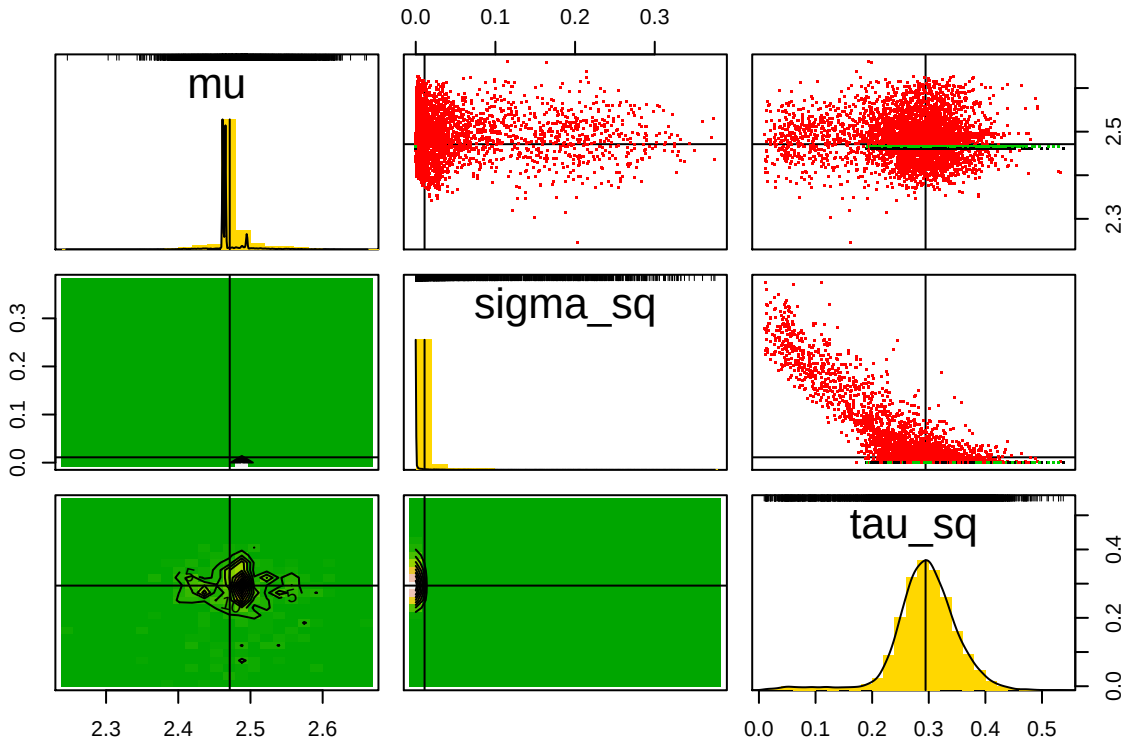
```
summary(fit[,c("mu","sigma_sq", "tau_sq")])
```

```
##
## Iterations = 31001:36000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean      SD  Naive SE Time-series SE
## mu       2.47128 0.02837 0.0002316       0.001890
## sigma_sq 0.01105 0.04030 0.0003291       0.005243
## tau_sq   0.29433 0.05843 0.0004771       0.004357
##
## 2. Quantiles for each variable:
##
##               2.5%       25%       50%       75%  97.5%
## mu       2.421e+00 2.461e+00 2.465e+00 2.4653036 2.5570
## sigma_sq 6.712e-21 2.269e-18 1.296e-15 0.0001356 0.1613
## tau_sq   1.393e-01 2.665e-01 2.964e-01 0.3276199 0.3981
```

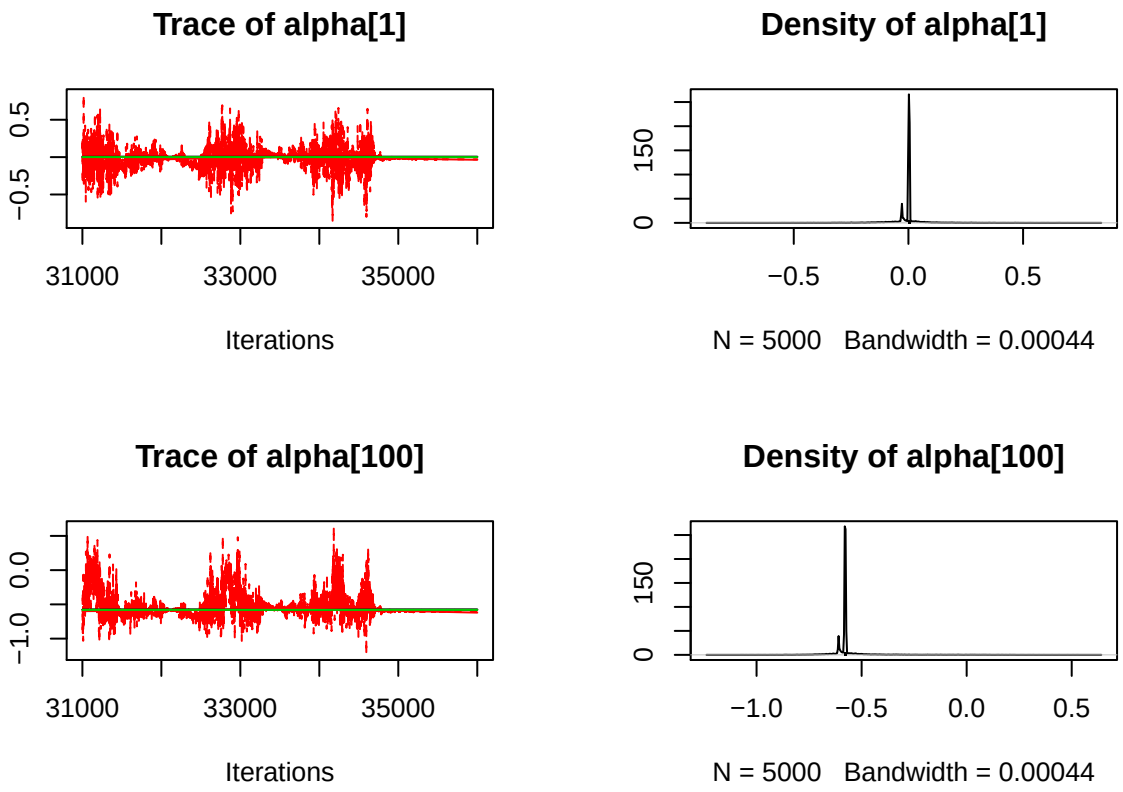
```
plot(fit[,c("mu","sigma_sq", "tau_sq")])
```



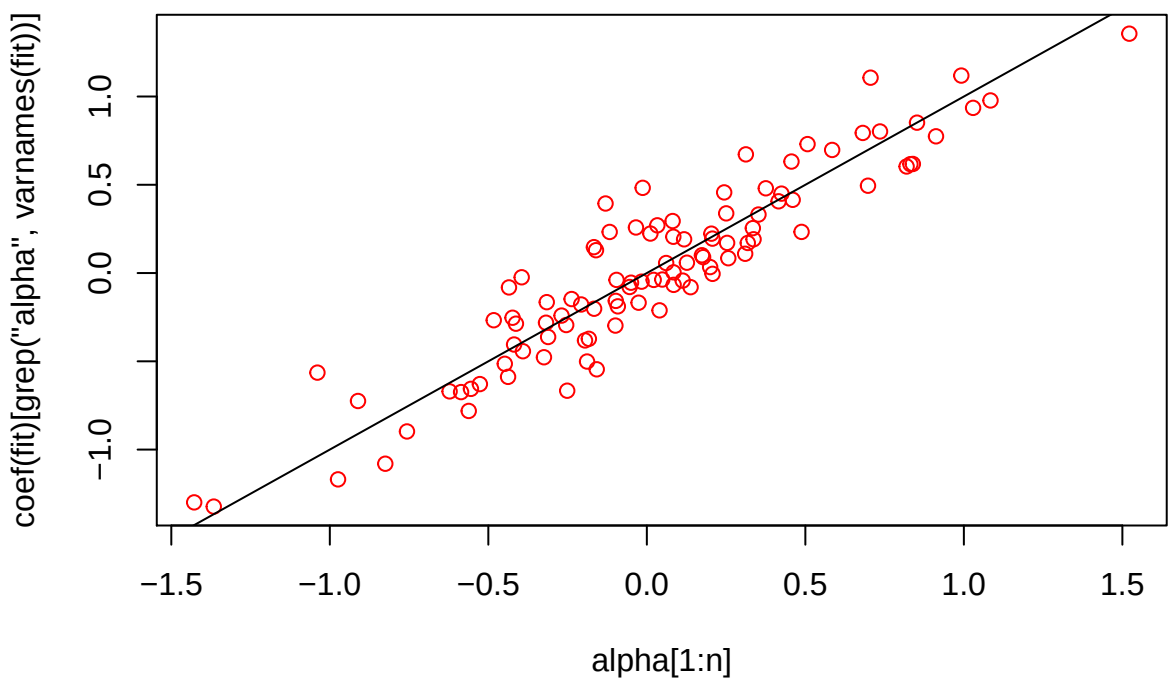

```
pairs(fit[,c("mu","sigma_sq", "tau_sq")])
```




```
plot(fit[,c("alpha[1]","alpha[100]")])
```



```
plot(alpha[1:n], coef(fit)[grep("alpha", varnames(fit))], col=2)
abline(0,1)
```



DC with the smoothing assumption:

```
model <- custommodel("model {
    for (k in 1:K) {
        for (i in 1:n) {
            for (j in 1:m) {
                Y[i,j,k] ~ dnorm(mu + alpha[i,k], 1 / sigma_sq)
            }
            alpha[i,k] ~ dnorm(0, 1 / tau_sq)
        }
    }
    log_sigma ~ dnorm(0, 0.001)
    sigma_sq <- exp(log_sigma)^2
    mu ~ dnorm(0, 0.1)
    log_tau ~ dnorm(0, 0.001)
    tau_sq <- exp(log_tau)^2
    sum <- sigma_sq + tau_sq
}")
dat <- list(Y = dcdim(array(Y, c(n, m, 1))), n = n,
    m = m, K = 1)
str(dat)
```

```
## List of 4
##  $ Y: dcdim [1:100, 1, 1] 2.47 2.32 2.75 1.78 2.17 ...
##   ..- attr(*, "drop")= logi TRUE
##   ..- attr(*, "class")= chr [1:2] "dcdim" "array"
##  $ n: num 100
##  $ m: num 1
##  $ K: num 1
```

```
K <- c(1, 10, 25)
dcfit1 <- dc.fit(data = dat,
    params = c("mu", "sigma_sq", "tau_sq"),
    model = model, n.iter = 1000,
    n.clones = K,
    unchanged = c("n", "m"), multiply = "K")
```

```
##
## Fitting model with 1 clone
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 103
##    Total graph size: 923
##
## Initializing model
##
##
## Fitting model with 10 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1000
##    Unobserved stochastic nodes: 1003
##    Total graph size: 9023
##
## Initializing model
##
##
## Fitting model with 25 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2500
##    Unobserved stochastic nodes: 2503
##    Total graph size: 22523
##
## Initializing model
```

```
## Warning in dclone::.dcFit(data, params, model, inits, n.clones, multiply =
## multiply, : chains convergence problem, see R.hat values
```

```
dcfit2 <- dc.fit(data = dat,
    params = c("mu", "sum"),
    model = model, n.iter = 1000,
```

```
    n.clones = K,
    unchanged = c("n", "m"), multiply = "K")
```

```
##
## Fitting model with 1 clone
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 103
##    Total graph size: 923
##
## Initializing model
##
##
## Fitting model with 10 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1000
##    Unobserved stochastic nodes: 1003
##    Total graph size: 9023
##
## Initializing model
##
##
## Fitting model with 25 clones
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2500
##    Unobserved stochastic nodes: 2503
##    Total graph size: 22523
##
## Initializing model
```

```
dcdiag(dcfit1)
```

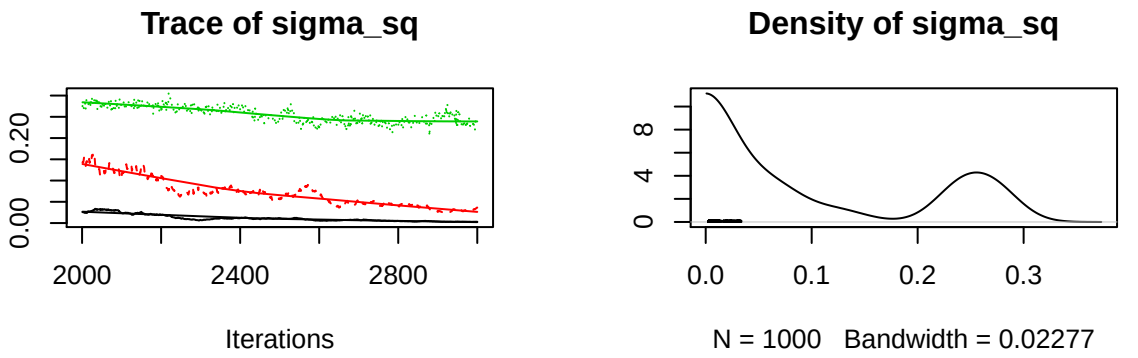
```
##   n.clones lambda.max  ms.error   r.squared    r.hat
## 1        1 0.02639509 0.1621144 0.018607549 2.023167
## 2       10 0.01611097 0.1418476 0.011262791 4.686486
## 3       25 0.02269569 0.1332986 0.006531604 6.694341
```

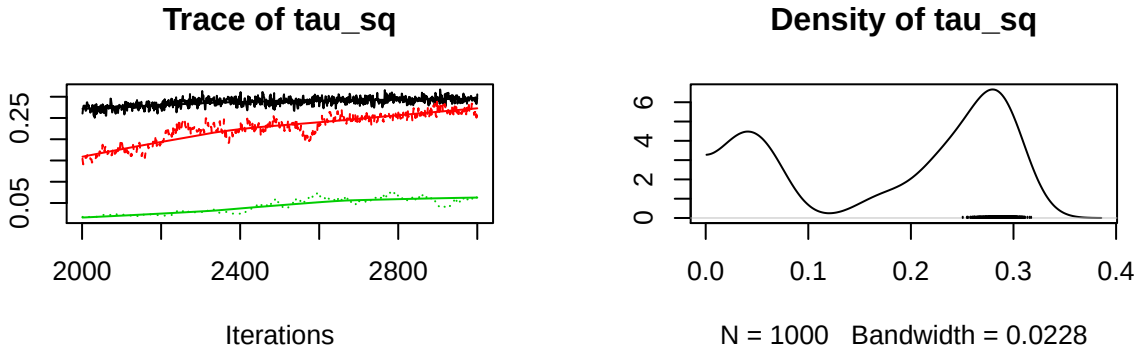
```
dcdiag(dcfit2)
```

```
##   n.clones   lambda.max    ms.error    r.squared    r.hat
```

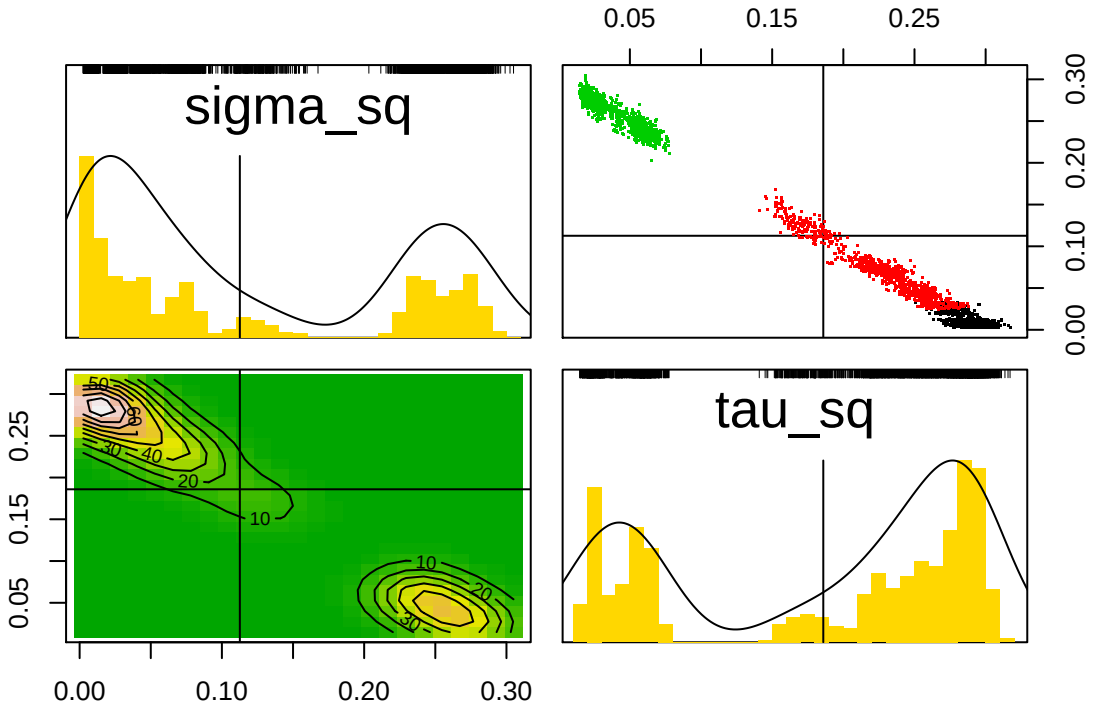
```
## 1         1 0.0021461939 0.238907833 0.0193474264 1.048283
## 2        10 0.0003087487 0.004744087 0.0011876787 1.009152
## 3        25 0.0001231966 0.005182170 0.0009563923 1.003428
```

```
plot(dcfit1[,c("sigma_sq", "tau_sq")])
```






```
pairs(dcfit1[,c("sigma_sq", "tau_sq")])
```



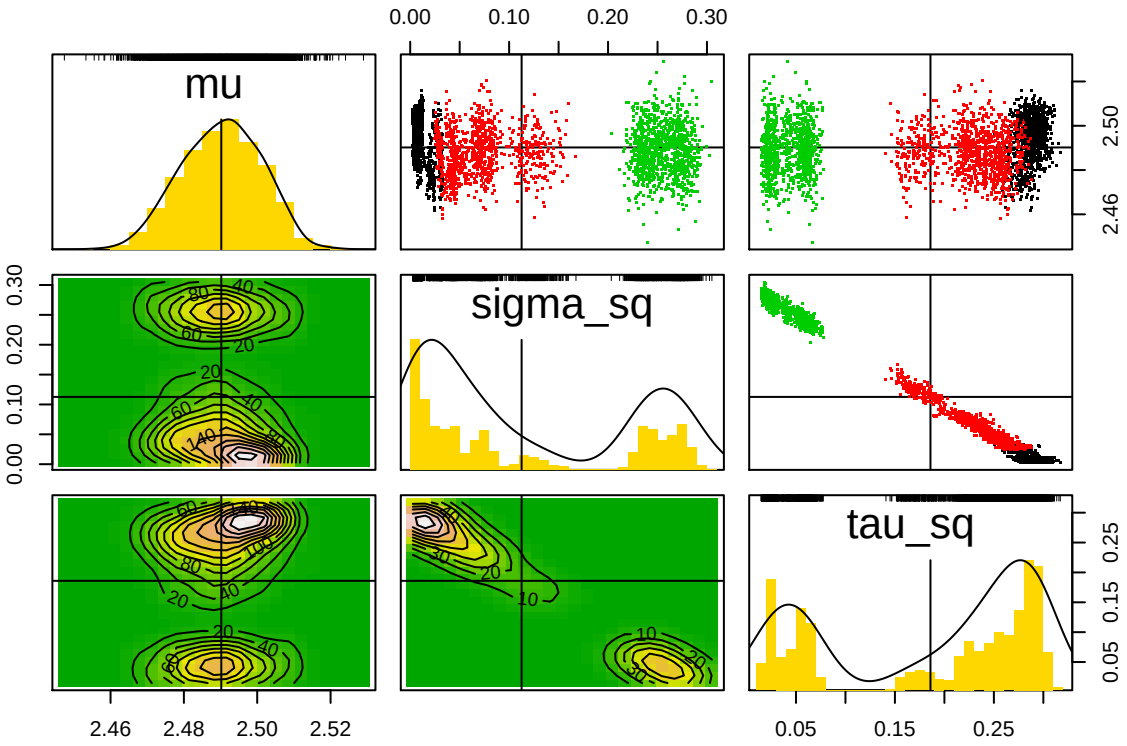

```
cov2cor(vcov(dcfit1))
```

```
##                  mu   sigma_sq     tau_sq
## mu        1.0000000 -0.1319759  0.1313767
## sigma_sq -0.1319759  1.0000000 -0.9971203
## tau_sq    0.1313767 -0.9971203  1.0000000
```

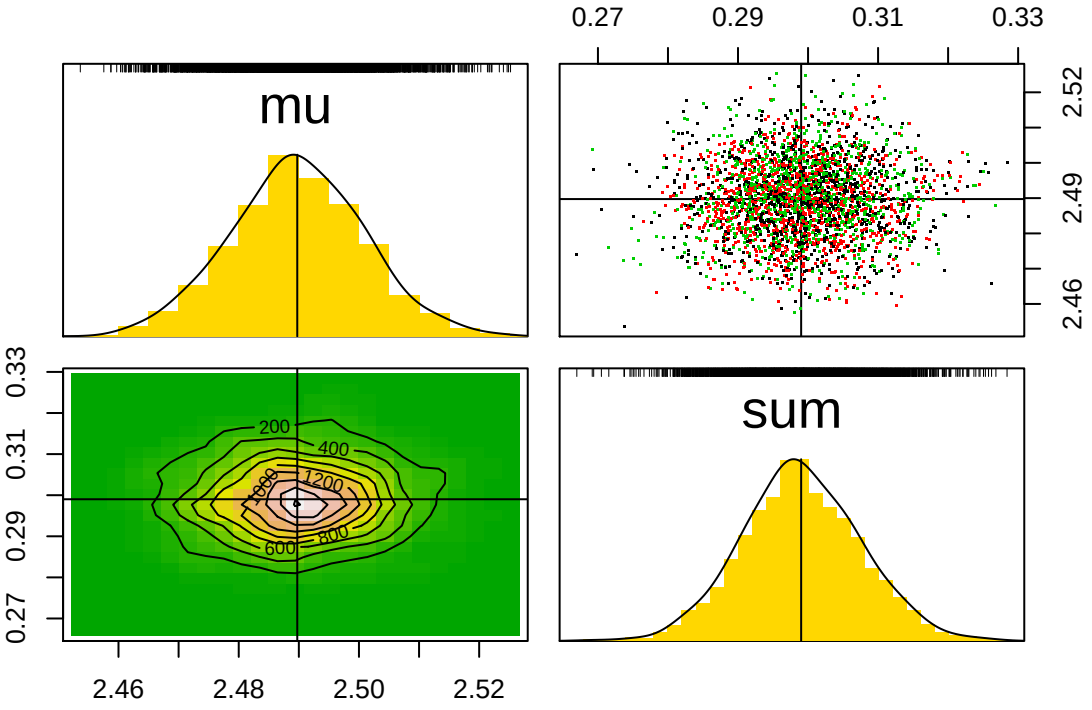
```
cov2cor(vcov(dcfit2))
```

```
##            mu       sum
## mu  1.0000000 0.0322487
## sum 0.0322487 1.0000000
```

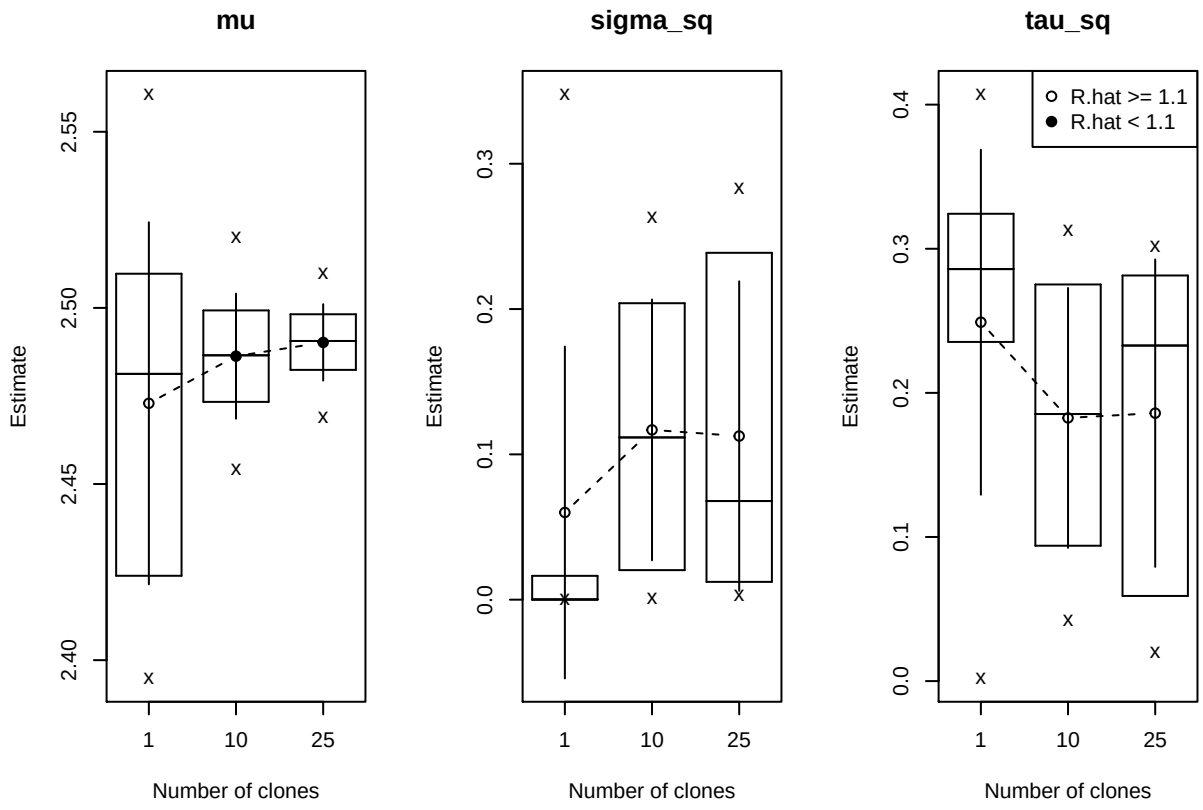
```
pairs(dcfit1)
```



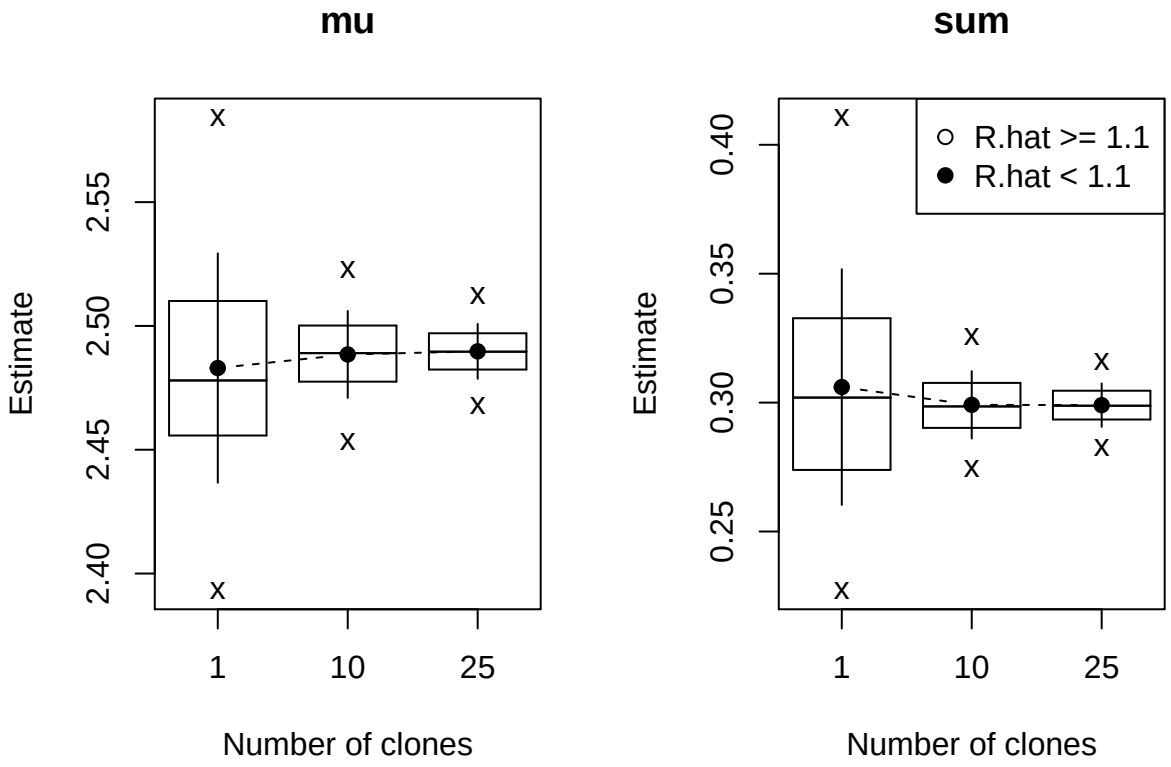

```
pairs(dcfit2)
```




```
plot(dctable(dcfit1))
```




```
plot(dctable(dcfit2))
```



```
coef(dcfit1)
```

```
##        mu  sigma_sq    tau_sq
## 2.4902057 0.1126094 0.1859192
```

```
coef(dcfit2)
```

```
##        mu       sum
## 2.4897385 0.2990147
```

```
c(mu=mu, sigma_sq=sigma_sq, tau_sq=tau_sq, sum=sigma_sq + tau_sq)
```

```
##       mu sigma_sq   tau_sq      sum
##     2.50     0.04     0.25     0.29
```

```
## m=1 leads to error:
## Error: number of levels of each grouping factor must be < number of observations
if (m > 1) {
    library(lme4)
    g <- rep(1:n, m)
    Yvec <- as.numeric(Y)
    mod.lm <- lmer(Yvec ~ 1 + (1|g))
    summary(mod.lm)
    plot(alpha[1:n], ranef(mod.lm)$g[,1], col = 2)
    abline(0, 1)
}
```

- Run things with `m = 1`, check diagnostics, estimates
- Run things with `m = 2`, check diagnostics, estimates

How do we predict α_i based on data cloning? `alpha` is now a vector of length `K`. We need a separate run for prediction:

```
model <- custommodel("model {
    for (i in 1:n) {
        for (j in 1:m) {
            Y[i,j] ~ dnorm(mu + alpha[i], 1 / sigma_sq)
        }
        alpha[i] ~ dnorm(0, 1 / tau_sq)
    }
    param[1:3] ~ dmnorm(cf[1:3], V[1:3,1:3])
    mu <- param[1]
    log_sigma <- param[2]
    sigma_sq <- exp(log_sigma)^2
    log_tau <- param[3]
    tau_sq <- exp(log_tau)^2
}")
## we need parameters on log scale
## calculate covariance matrix by hand
## create a matrix of the posterior samples
pos <- as.matrix(dcfit1)
head(pos)
```

```
##            mu   sigma_sq    tau_sq
## [1,] 2.491132 0.02562298 0.2604130
## [2,] 2.497418 0.02585292 0.2681768
## [3,] 2.498921 0.02575770 0.2751055
## [4,] 2.496136 0.02560646 0.2791320
## [5,] 2.490273 0.02456243 0.2777545
## [6,] 2.496997 0.02450219 0.2740195
```

```
pos[,"sigma_sq"] <- log(sqrt(pos[,"sigma_sq"]))
pos[,"tau_sq"] <- log(sqrt(pos[,"tau_sq"]))
colnames(pos)[2:3] <- c("log_sigma", "log_tau")
head(pos)
```

```
##            mu log_sigma    log_tau
## [1,] 2.491132 -1.832133 -0.6727432
## [2,] 2.497418 -1.827666 -0.6580543
## [3,] 2.498921 -1.829511 -0.6453003
## [4,] 2.496136 -1.832455 -0.6380352
## [5,] 2.490273 -1.853269 -0.6405088
## [6,] 2.496997 -1.854496 -0.6472780
```

```
(V <- cov(pos) * nclones(dcfit1))
```

```
##                     mu   log_sigma     log_tau
## mu         0.002939742 -0.04600054  0.01352784
## log_sigma -0.046000538 13.21438385 -6.92684678
## log_tau    0.013527839 -6.92684678  5.45395558
```

```
(cf <- colMeans(pos))
```

```
##         mu  log_sigma    log_tau
##  2.4902057 -1.4725911 -0.9981597
```

```
dat <- list(Y = Y, n = n, m = m,
    cf = cf, V = solve(V)) # precision matrix
pred <- jags.fit(data = dat,
    params = c("alpha"),
    model = model)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 101
##    Total graph size: 1232
##
## Initializing model
```

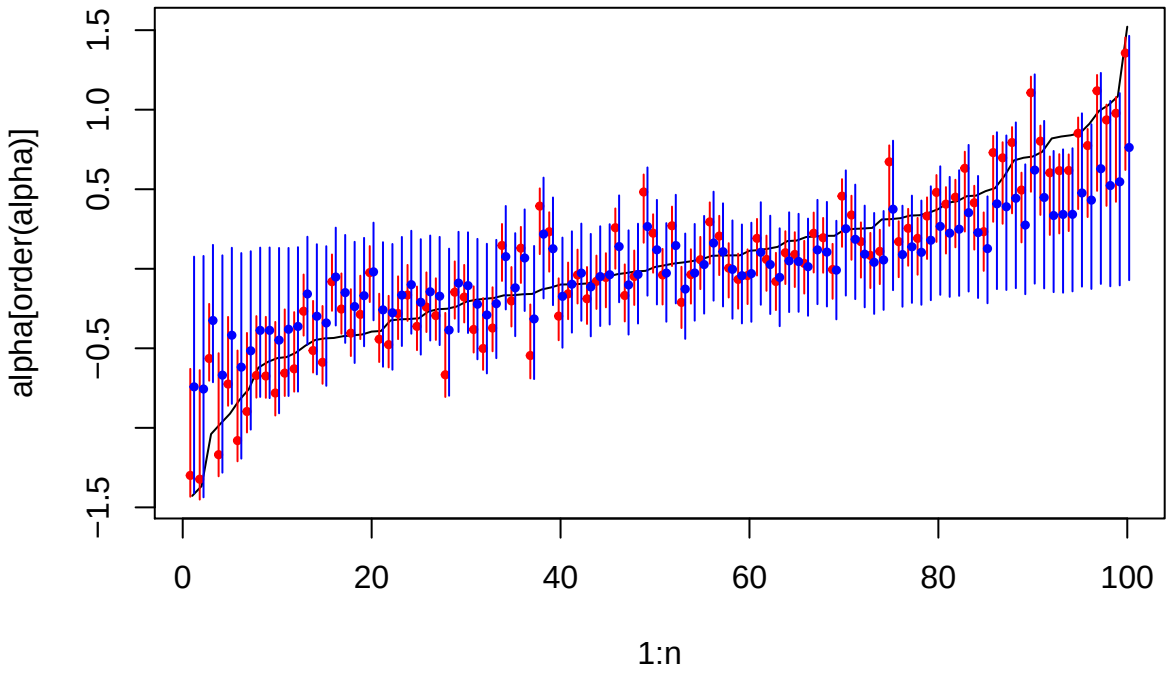
```
alpha_b <- cbind(est=coef(fit)[grep("alpha", varnames(fit))],
    t(quantile(fit[,grep("alpha", varnames(fit))],
    probs = c(0.025, 0.975))))
alpha_dc <- cbind(est=coef(pred),
    t(quantile(pred, probs = c(0.025, 0.975))))
head(alpha_b)
```

```
##                  est        2.5%      97.5%
## alpha[1] -0.00457169 -0.18890176  0.1566044
## alpha[2] -0.14725760 -0.31369421  0.0464140
## alpha[3]  0.27026841  0.01701318  0.3908752
## alpha[4] -0.66649206 -0.80599839 -0.2774934
## alpha[5] -0.28685120 -0.44224024 -0.0434310
## alpha[6]  0.20568151 -0.03985084  0.3330830
```

```
head(alpha_dc)
```

```
##                   est       2.5%     97.5%
## alpha[1] -0.008481782 -0.3180337 0.3018383
## alpha[2] -0.090082937 -0.3973713 0.2323804
## alpha[3]  0.145996538 -0.2177137 0.4651833
## alpha[4] -0.385513471 -0.7982691 0.1265031
## alpha[5] -0.169143152 -0.4872084 0.1942887
## alpha[6]  0.107438805 -0.2368748 0.4114261
```

```
plot(1:n, alpha[order(alpha)], type = "l",
    ylim = range(alpha, alpha_b, alpha_dc))
points(1:n - 0.2, alpha_b[order(alpha),1],
    col = 2, pch = 19, cex = 0.5)
segments(x0 = 1:n - 0.2, x1 = 1:n - 0.2,
    y0 = alpha_b[order(alpha),2],
    y1 = alpha_b[order(alpha),3], col = 2)
points(1:n + 0.2, alpha_dc[order(alpha),1],
    col=4, pch = 19, cex = 0.5)
segments(x0 = 1:n + 0.2, x1 = 1:n + 0.2,
    y0 = alpha_dc[order(alpha),2],
    y1 = alpha_dc[order(alpha),3], col = 4)
```




```
table(rowSums(sign(alpha - alpha_b[,-1])))
```

```
##
## -2  0  2
## 12 70 18
```

```
table(rowSums(sign(alpha - alpha_dc[,-1])))
```

```
##
## -2  0  2
##  6 87  7
```

```
model <- custommodel("model {
    for (i in 1:n) {
        for (j in 1:m) {
            Y[i,j] ~ dnorm(mu + alpha[i], 1 / sigma_sq)
        }
        alpha[i] ~ dnorm(0, 1 / tau_sq)
    }
    sigma_sq ~ dunif(0.001, 5)
    mu ~ dnorm(10, 1)
    tau_sq ~ dgamma(0.01, 0.01)
}")
dat <- list(Y = Y, n = n, m = m)
fit2 <- jags.fit(data = dat,
    params = c("mu", "sigma_sq", "tau_sq", "alpha"),
    model = model, n.update = 30000)
```

```
## Compiling model graph
```

```
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 103
##    Total graph size: 613
##
## Initializing model
```

```
alpha_b2 <- cbind(est=coef(fit2)[grep("alpha", varnames(fit))],
    t(quantile(fit2[,grep("alpha", varnames(fit))],
    probs = c(0.025, 0.975))))

plot(1:n, alpha[order(alpha)], type = "l",
    ylim = range(alpha, alpha_b, alpha_dc))
points(1:n - 0.2, alpha_b[order(alpha),1],
    col = 2, pch = 19, cex = 0.5)
segments(x0 = 1:n - 0.2, x1 = 1:n - 0.2,
    y0 = alpha_b[order(alpha),2],
    y1 = alpha_b[order(alpha),3], col = 2)
points(1:n + 0.2, alpha_b2[order(alpha),1],
    col=3, pch = 19, cex = 0.5)
segments(x0 = 1:n + 0.2, x1 = 1:n + 0.2,
    y0 = alpha_b2[order(alpha),2],
    y1 = alpha_b2[order(alpha),3], col = 3)
```