

# Package ‘lmmpar’

July 22, 2025

**Title** Parallel Linear Mixed Model

**Version** 0.1.0

**Description** Embarrassingly Parallel Linear Mixed Model calculations spread across local cores which repeat until convergence.

**Encoding** UTF-8

**License** MIT + file LICENSE

**LazyData** TRUE

**Depends** R (>= 3.2.2)

**Imports** MASS, matrixcalc, mnormt, plyr, doParallel, bigmemory

**Suggests** testthat

**RoxygenNote** 6.0.1

**URL** <https://github.com/fulyagokalp/lmmpar>

**BugReports** <https://github.com/fulyagokalp/lmmpar/issues>

**NeedsCompilation** no

**Author** Fulya Gokalp Yavuz [aut, cre],  
Barret Schloerke [aut]

**Maintainer** Fulya Gokalp Yavuz <fulyagokalp@gmail.com>

**Repository** CRAN

**Date/Publication** 2017-08-03 15:17:37 UTC

## Contents

|                  |          |
|------------------|----------|
| lmmpar . . . . . | 2        |
| <b>Index</b>     | <b>4</b> |

lmmpar

*Parallel Linear Mixed Model***Description**

Embarrassingly Parallel Linear Mixed Model calculations spread across local cores which repeat until convergence. All calculations are currently done locally, but theoretically, the calculations could be extended to multiple machines.

**Usage**

```
lmmpar(Y, X, Z, subject, beta, R, D, sigma, maxiter = 500, cores = 8,
       verbose = TRUE)
```

**Arguments**

|         |                                                                                                                                       |
|---------|---------------------------------------------------------------------------------------------------------------------------------------|
| Y       | matrix of responses with observations/subjects on column and repeats for each observation/subject on rows. It is (m x n) dimensional. |
| X       | observed design matrices for fixed effects. It is (m*n x p) dimensional.                                                              |
| Z       | observed design matrices for random effects. It is (m*n x q) dimensional.                                                             |
| subject | vector of positions that belong to each subject.                                                                                      |
| beta    | fixed effect estimation vector with length p.                                                                                         |
| R       | variance-covariance matrix of residuals.                                                                                              |
| D       | variance-covariance matrix of random effects.                                                                                         |
| sigma   | initial sigma value.                                                                                                                  |
| maxiter | the maximum number of iterations that should be calculated.                                                                           |
| cores   | the number of cores. Why not to use maximum?!                                                                                         |
| verbose | boolean that defaults to print iteration context                                                                                      |

**Examples**

```
# Set up fake data
n <- 1000 # number of subjects
m <- 4    # number of repeats
N <- n * m # true size of data
p <- 15   # number of betas
q <- 2    # width of random effects

# Initial parameters
# beta has a 1 for the first value. all other values are ~N(10, 1)
beta <- rbind(1, matrix(rnorm(p, 10), p, 1))
R <- diag(m)
D <- matrix(c(16, 0, 0, 0.025), nrow = q)
sigma <- 1
```

```

# Set up data
subject <- rep(1:n, each = m)
repeats <- rep(1:m, n)

subj_x <- lapply(1:n, function(i) cbind(1, matrix(rnorm(m * p), nrow = m)))
X <- do.call(rbind, subj_x)
Z <- X[, 1:q]
subj_beta <- lapply(1:n, function(i) mnormt::rmnorm(1, rep(0, q), D))
subj_err <- lapply(1:n, function(i) mnormt::rmnorm(1, rep(0, m), sigma * R))

# create a known response
subj_y <- lapply(
  seq_len(n),
  function(i) {
    (subj_x[[i]] %*% beta) +
    (subj_x[[i]][, 1:q] %*% subj_beta[[i]]) +
    subj_err[[i]]
  }
)
Y <- do.call(rbind, subj_y)

# run the algorithm to recover the known betas
ans <- lmmpar(
  Y,
  X,
  Z,
  subject,
  beta = beta,
  R = R,
  D = D,
  cores = 1, # increase for faster computation
  sigma = sigma,
  verbose = TRUE
)
str(ans)

```

# Index

Impar, [2](#)