miceFast-Usage

Maciej Nasinski

June 10, 2017

Loading packages and set.seed:

```
library(pacman)

p_load(Rcpp,
       mice,
       tidyverse,
       broom)

set.seed(1234)
```

Motivations

Missing data is a common problem. The easiest solution is to delete observations for which dependent variable is missing. But this will sometimes deteriorate quality of a project. Another solution will be to use methods such as multiple imputations to fill the missing data. Non missing independent variables could be used to approximate a missing observations for a dependent variable. R or Python language are comfortable for data manipulation but parallely brings slower computations. Languages such as C++ gives an opportunity to boost our applications or projects.

The presented miceFast module was built under Rcpp packages and the C++ library Armadillo. The Rcpp package offers functionality of exporting full C++ capabilities to the R environment. More precisely miceFast and corrData are offered. The first module offers capabilities of functions with a closed-form solution at the mice R package. The main upgrade is possibility of including a grouping variable and/or a weighting variable and C++ capabilities. The second module was made for purpose of presenting the miceFast usage and performance. It gives functionality of generating correlated data with a discrete, binomial or continuous dependent variable and continuous independent variables.

In the project was used the knowledge from mice and MASS R packages.

Example

Genereting Data

Loading corrData module:

```
Rcpp::sourceCpp("C:/Users/user/Desktop/Imputations/Imput/miceFast-projekt/corrData.cpp")
```

Available constructors:

new(corrData,nr_cat,n_obs,means,cor_matrix)

new(corrData,n_obs,means,cor_matrix)

where:

- nr_cat : number of categories for discrete dependent variable
- n_obs : number of observations
- means: center independent variables

- cor_mat : positive defined correlation matrix

relevant class methods:

- fill("type") : generating data

type - ("contin","binom","discrete")

Generating correlated data for all three possible data types of dependent variable

```
power = 5 # power of 10 - number of observations - should be adjusted to a computer capabilities

grs = 100 # grouping variable - number of groups

## generete example - data

##positive-defined correlation matrix

cors = matrix(c(1,0.6,0.7,0.4,0.4,0.5,0.35,
                NA,1,0.2,0.05,0.1,0.12,0.15,
                NA,NA,1,0.15,0.15,0.1,0.08,
                NA,NA,NA,1,0.12,0.15,0.1,
                NA,NA,NA,NA,1,0.15,0.2,
                NA,NA,NA,NA,NA,1,0.15,
                NA,NA,NA,NA,NA,NA,1),7,7,byrow = T)

cors[lower.tri(cors)] = t(cors)[lower.tri(cors)]

# automatic corr matrix - close to diagonal

#cors = stats::rWishart(100,10,diag(7))

#cors = apply(cors,1:2,mean)/10

#cors

##

model = new(corrData,10,10^power,rep(0,7),cors)

data_bin = model$fill("binom")
data_disc = model$fill("discrete")
data_con = model$fill("contin")

colnames(data_bin) = c("y","x1","x2","x3","x4","x5","group")
colnames(data_disc) = c("y","x1","x2","x3","x4","x5","group")
colnames(data_con) = c("y","x1","x2","x3","x4","x5","group")
```

Sampling 10% of observations - artificial missing values:

```
## NA index
index_NA = 1:nrow(data_con) %in% sample(1:nrow(data_con),10^(power-1))
```

A grouping variable:

```
#Grouping variable

data_disc[,7] = floor(pnorm(data_disc[,7])*grs)
```

```r
data_disc = data_disc[order(data_disc[,7]),] # sort by group

data_disc = cbind(data_disc,index_NA)

gr_disc = data_disc[,7]

index_NA = as.logical(data_disc[,8])# index_NA after sorting

#continous model

data_con[,7] = floor(pnorm(data_con[,7])*grs)

data_con = cbind(data_con,index_NA)

data_con = data_con[order(data_con[,7]),] # sort by group

gr_con = data_con[,7]

index_NA = as.logical(data_disc[,8])# index_NA after sorting
```

Presenting Data - Continuous & Discrete:

```r
round(head(data_disc),3)
```

```
##      y     x1     x2     x3     x4     x5 group index_NA
## [1,] 5 -0.100  0.501 -0.171  0.095  0.038     0        0
## [2,] 1 -0.998 -1.996 -2.238  0.155  0.096     0        0
## [3,] 1 -1.100 -0.307 -1.105 -0.923  0.187     0        0
## [4,] 4 -0.096  0.160  0.004 -1.395 -0.041     0        0
## [5,] 9  1.360  2.388  0.000 -0.528  0.007     0        0
## [6,] 2 -1.924 -0.254  0.207 -0.174  1.226     0        0
```

```r
round(head(data_con),3)
```

```
##           y     x1     x2     x3     x4     x5 group index_NA
## [1,] -0.504  0.630 -0.631 -0.556  0.231  0.599     0        1
## [2,] -0.359 -0.493  0.878  0.054 -2.547  0.865     0        0
## [3,] -1.323 -1.357 -0.233  0.306  0.994 -1.217     0        0
## [4,] -1.054 -1.084 -0.496 -1.270 -0.662  1.554     0        0
## [5,] -2.589 -0.877 -0.693 -0.552 -0.111 -3.873     0        0
## [6,] -1.183 -0.670 -0.077 -2.404  0.342  0.300     0        0
```

```r
round(cor(data_disc),3)
```

```
##                y    x1     x2    x3     x4    x5 group index_NA
## y          1.000 0.582  0.673 0.387  0.384 0.481 0.329   -0.005
## x1         0.582 1.000  0.198 0.053  0.100 0.117 0.145    0.000
## x2         0.673 0.198  1.000 0.146  0.146 0.094 0.073   -0.004
## x3         0.387 0.053  0.146 1.000  0.121 0.149 0.098    0.000
## x4         0.384 0.100  0.146 0.121  1.000 0.148 0.193   -0.007
## x5         0.481 0.117  0.094 0.149  0.148 1.000 0.145    0.003
## group      0.329 0.145  0.073 0.098  0.193 0.145 1.000    0.001
## index_NA -0.005 0.000 -0.004 0.000 -0.007 0.003 0.001    1.000
```

```r
round(cor(data_con),3)
```

```
##                y    x1     x2    x3    x4     x5 group index_NA
## y          1.000 0.600  0.700 0.398 0.399  0.496 0.337   -0.002
## x1         0.600 1.000  0.199 0.052 0.100  0.117 0.145    0.001
## x2         0.700 0.199  1.000 0.146 0.151  0.097 0.076   -0.005
## x3         0.398 0.052  0.146 1.000 0.117  0.148 0.090    0.002
## x4         0.399 0.100  0.151 0.117 1.000  0.147 0.193    0.003
## x5         0.496 0.117  0.097 0.148 0.147  1.000 0.145   -0.005
## group      0.337 0.145  0.076 0.090 0.193  0.145 1.000    0.003
## index_NA -0.002 0.001 -0.005 0.002 0.003 -0.005 0.003    1.000
```

Imputations

Loading miceFast module:

```
sourceCpp("C:/Users/user/Desktop/Imputations/Imput/miceFast-projekt/miceFast.cpp")
```

Building miceFast objects - a simple model or with a grouping variable:

available constructors:

new(miceFast,y,x,index_NA)

new(miceFast,y,x,index_NA,grouping,sorted)

new(miceFast,y,x,index_NA,weights)

new(miceFast,y,x,index_NA,grouping,sorted,weights)

where:

- y : dependent variable - type vector
- x : independent variables - type matrix
- index_NA : vector of bool (or 0/1) where TRUE equal missing!!!
- grouping : vector of integers for grouping variable - you could build it form several discrete variables
- sorted : boolean (TRUE/FALSE) specifying if data is already sorted by a grouping variable
- weights: vector of weights for weighted linear regressions

relevant class methods:

- impute("model") - impute data
- imputeby("model") - impute data divide imputations by a grouping variable
- imputeW("model2") - impute data with weights
- imputebyW("model2") - impute data divide imputations by a grouping variable with weights
- get_models() - possible quantitative models for a certain type dependent variable
- sortby_g() - sort data by a grouping variable

model - ("lda","lm_pred","lm_bayes","lm_noise") model2 - ("lm_pred","lm_bayes","lm_noise")

- for simple mean use "lm_pred" and x=as.matrix(rep(1,"nrow"))

Base model:

Continuous data:

```
model = new(miceFast,data_con[,1],cbind(1,data_con[,c(2:7)]),index_NA)

#get availible predction models
model$get_models()
```

```
## [1] "lm_pred or lm_bayes or lm_noise"
```

```r
#implementing lm_pred
pred =  model$impute("lm_pred")

sum((pred-data_con[index_NA,1])^2)
```

```
## [1] 269.691
```

```r
head(cbind(pred,data_con[index_NA,1]))
```

```
##            [,1]        [,2]
## [1,] -0.8958526 -1.01191759
## [2,] -1.7952301 -1.86822635
## [3,] -0.9510549 -1.22299932
## [4,] -0.5218749 -0.76574770
## [5,] -0.7375411 -0.99739020
## [6,]  0.1719752  0.07865915
```

Discrete data:

```r
model = new(miceFast,data_disc[,1],data_disc[,c(2:7)],index_NA)

#get availible predction models
model$get_models()
```

```
## [1] "recommended lda or (lm_pred,lm_bayes,lm_noise - remember to round results if needed)"
```

```r
#implementing lda
pred =  model$impute("lda")

table(pred,data_disc[index_NA,1])
```

```
##
## pred   1   2   3   4   5   6   7   8   9  10
##   1  811  36   0   0   0   0   0   0   0   0
##   2  208 791 160   3   0   0   0   0   0   0
##   3    0 178 655 238  12   0   0   0   0   0
##   4    0   8 217 515 225  18   0   0   0   0
##   5    0   0  18 233 497 203  17   0   0   0
##   6    0   0   0  20 237 542 226  11   0   0
##   7    0   0   0   2  16 248 501 201   4   0
##   8    0   0   0   0   1  12 203 621 160   0
##   9    0   0   0   0   0   0   7 173 716 181
##   10   0   0   0   0   0   0   0   0  55 820
```

Using a grouping variable:

Continuous data:

```r
model = new(miceFast,data_con[,1],cbind(1,data_con[,c(2:6)]),index_NA,gr_con,TRUE)

#get availible predction models
model$get_models()
```

```
## [1] "lm_pred or lm_bayes or lm_noise"
```

```r
#implementing lm_pred
pred =  model$imputeby("lm_pred")

sum((pred-data_con[index_NA,1])^2)
```

```
## [1] 263.829
```

```
head(cbind(pred,data_con[index_NA,1]))
```

```
##             [,1]        [,2]
## [1,] -1.05010711 -1.01191759
## [2,] -1.90784372 -1.86822635
## [3,] -1.08251764 -1.22299932
## [4,] -0.65269462 -0.76574770
## [5,] -0.85897135 -0.99739020
## [6,]  0.04889113  0.07865915
```

Discrete data:

```
model = new(miceFast,data_disc[,1],data_disc[,c(2:6)],index_NA,gr_disc,TRUE)

#get availible predction models
model$get_models()
```

```
## [1] "recommended lda or (lm_pred,lm_bayes,lm_noise - remember to round results if needed)"
```

```
#implementing lda
pred =  model$imputeby("lda")

table(pred,data_disc[index_NA,1])
```

```
##
## pred   1   2   3   4   5   6   7   8   9  10
##    1  806  45   0   0   0   0   0   0   0   0
##    2  213 775 158   4   0   0   0   0   0   0
##    3    0 188 658 239  11   0   0   0   0   0
##    4    0   5 217 528 219  14   0   0   0   0
##    5    0   0  17 220 513 203  15   0   0   0
##    6    0   0   0  19 227 544 219   6   0   0
##    7    0   0   0   1  16 252 514 187   5   0
##    8    0   0   0   0   2  10 201 629 157   0
##    9    0   0   0   0   0   0   5 184 712 180
##    10   0   0   0   0   0   0   0   0  61 821
```

Additional functionality - weighted linear regressions

Weights:

```
weights = pnorm(data_con[,6])
```

Base model:

```
model = new(miceFast,data_con[,1],cbind(1,data_con[,c(2:5,7)]),index_NA,weights)

#get availible predction models
model$get_models()
```

```
## [1] "lm_pred or lm_bayes or lm_noise"
```

```
#implementing lm_pred
pred =  model$imputeW("lm_pred")

sum((pred-data_con[index_NA,1])^2)
```

```
## [1] 1521.995
```

```
head(cbind(pred,data_con[index_NA,1]))
```

```
##              [,1]        [,2]
## [1,] -0.15592592 -1.01191759
## [2,] -1.92630019 -1.86822635
## [3,] -0.86662707 -1.22299932
## [4,] -0.56782161 -0.76574770
## [5,] -0.48264027 -0.99739020
## [6,]  0.03187084  0.07865915
```

with grouping variable:

```
model = new(miceFast,data_con[,1],cbind(1,data_con[,c(2:5)]),index_NA,gr_con,TRUE,weights)

#get availible predction models
model$get_models()
```

```
## [1] "lm_pred or lm_bayes or lm_noise"
```

```
#implementing lm_pred
pred =  model$imputebyW("lm_pred")

sum((pred-data_con[index_NA,1])^2)
```

```
## [1] 1517.636
```

```
head(cbind(pred,data_con[index_NA,1]))
```

```
##             [,1]        [,2]
## [1,] -0.2918551 -1.01191759
## [2,] -2.0809335 -1.86822635
## [3,] -1.0029923 -1.22299932
## [4,] -0.6997534 -0.76574770
## [5,] -0.6421331 -0.99739020
## [6,] -0.1184150  0.07865915
```

Performance

Environment: MRO Intel MKL - i7 6700HQ and 24GB DDR4

If you are interested about the procedure of testing performance check performance_validity.R file.

Mice fast was compared with the mice package. For grouping option there was used a basic R looping and the popular dplyr package.

Summing up, miceFast offer a relevant boost of calculations for LDA and all models with grouping option'.

'It was tested for up to 100 independent variables and 1 million observations.

Plots for 1 million observations, 7 independent variables and 100 levels for a grouping variable:

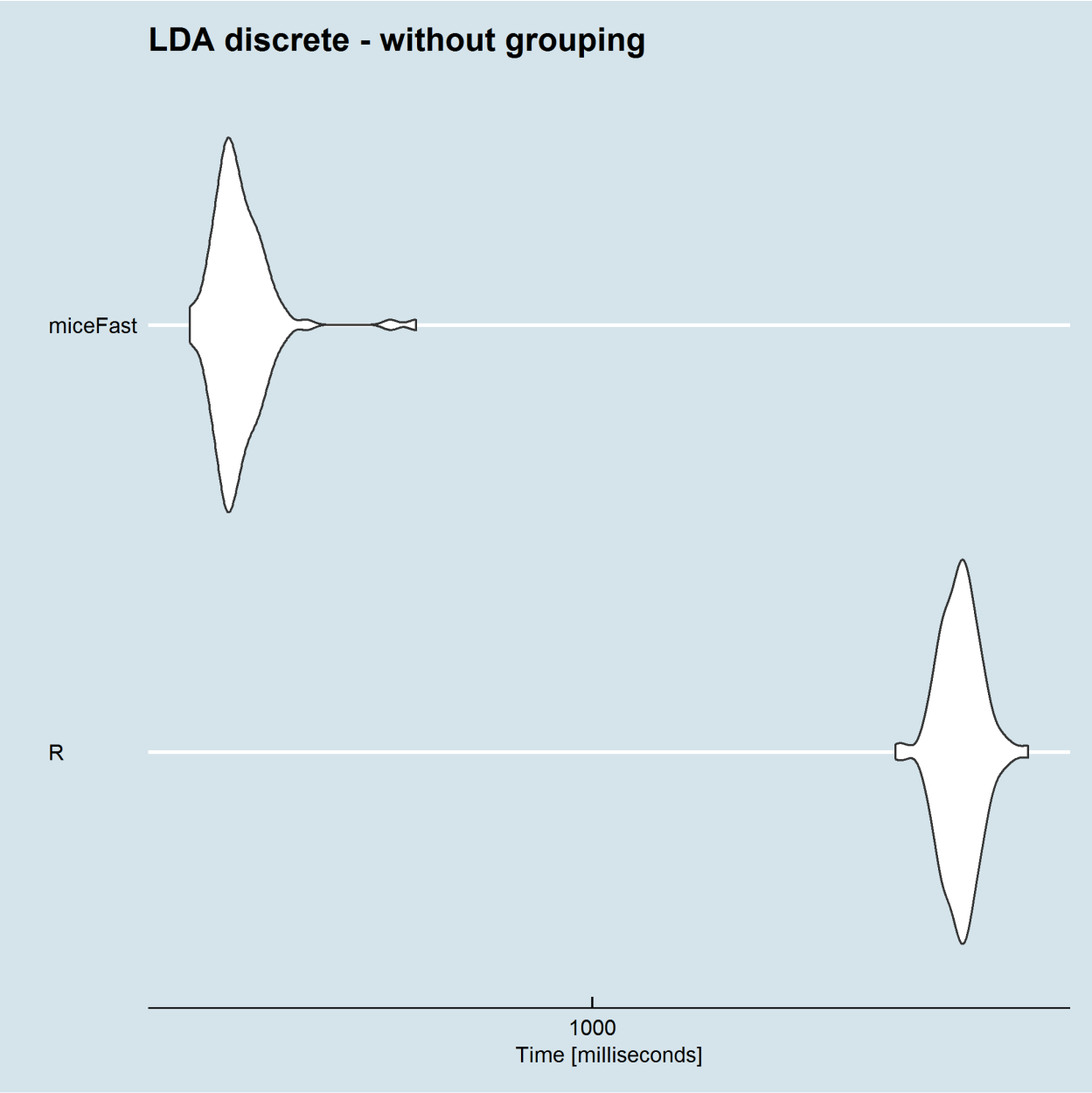
LDA discrete - without grouping
miceFast
R
1000
Time [milliseconds]

Figure 1: “”

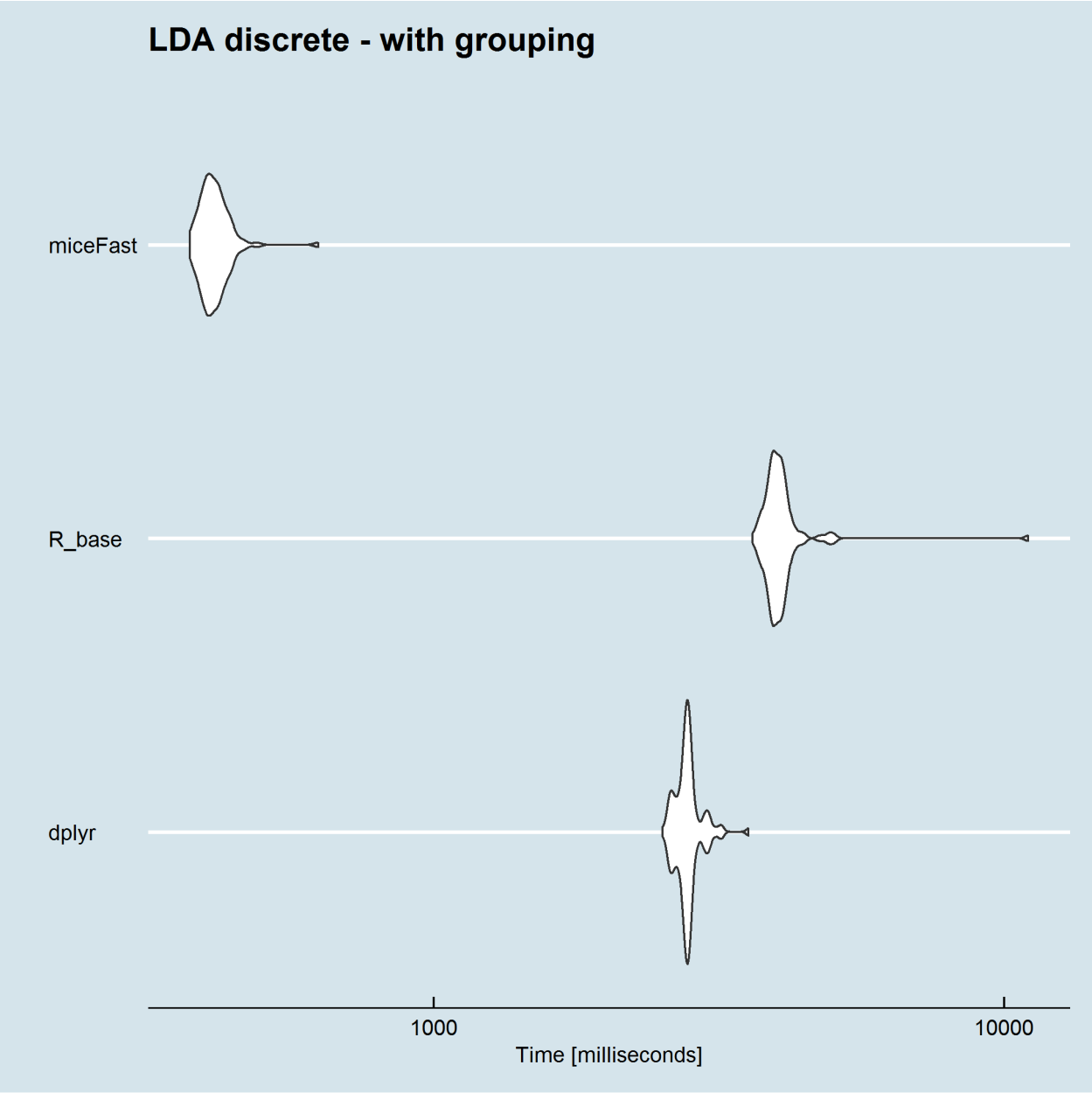
LDA discrete - with grouping
miceFast
R_base
dplyr
1000
10000
Time [milliseconds]

Figure 2: “”

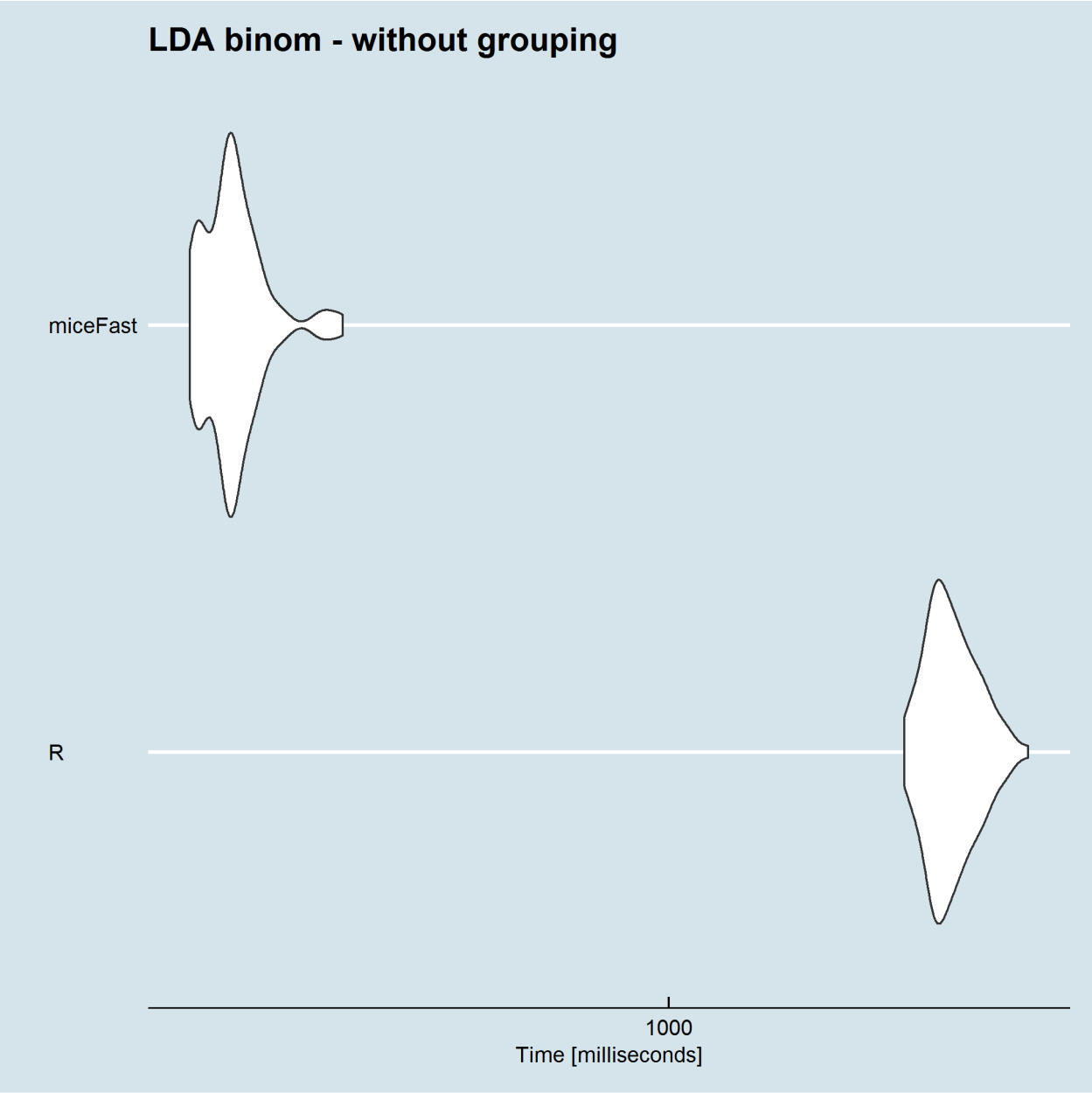
LDA binom - without grouping
miceFast
R
1000
Time [milliseconds]

Figure 3: “”

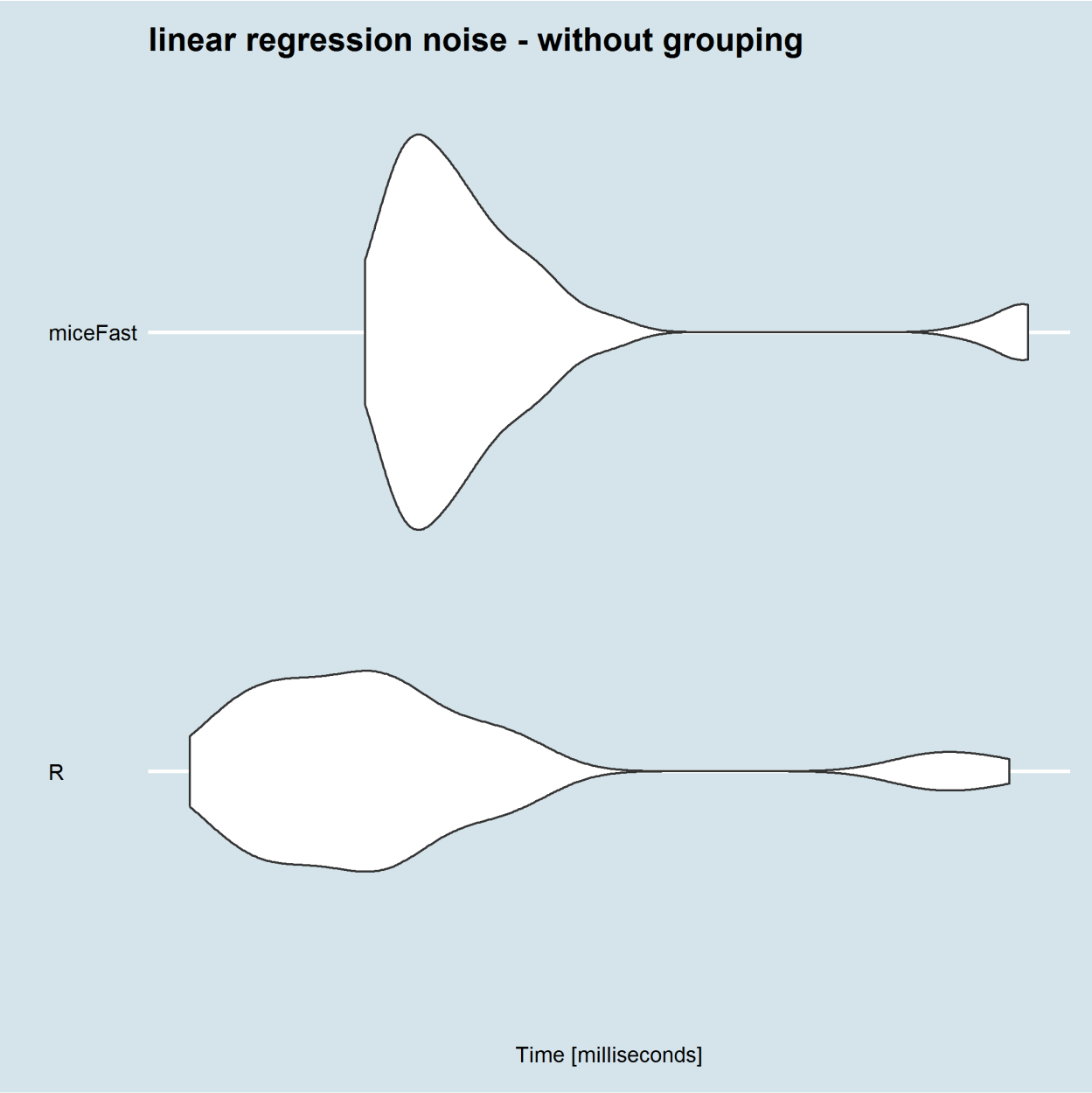
linear regression noise - without grouping
miceFast
R
Time [milliseconds]

Figure 4: “”

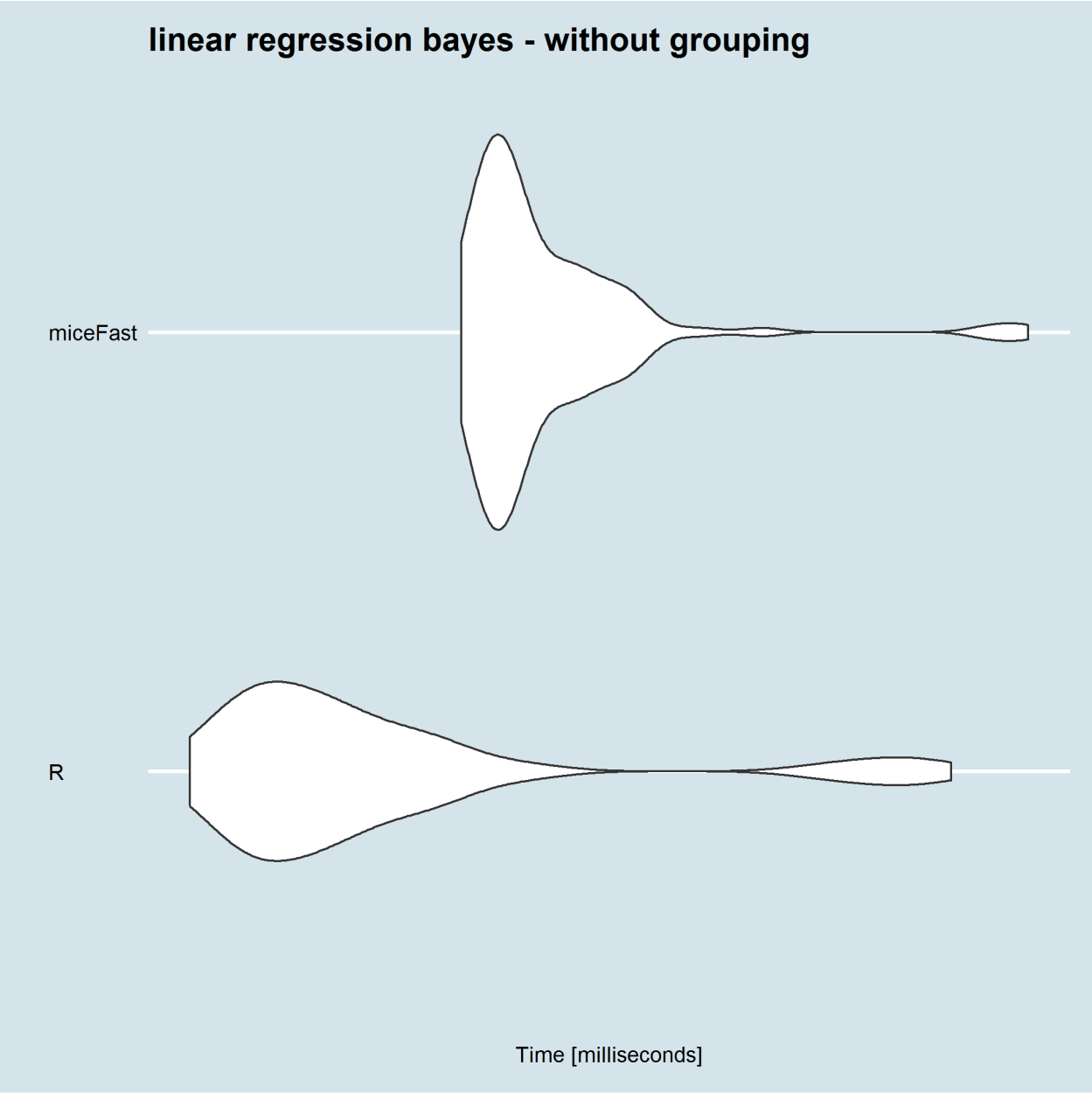
linear regression bayes - without grouping
miceFast
R
Time [milliseconds]

Figure 5: “”

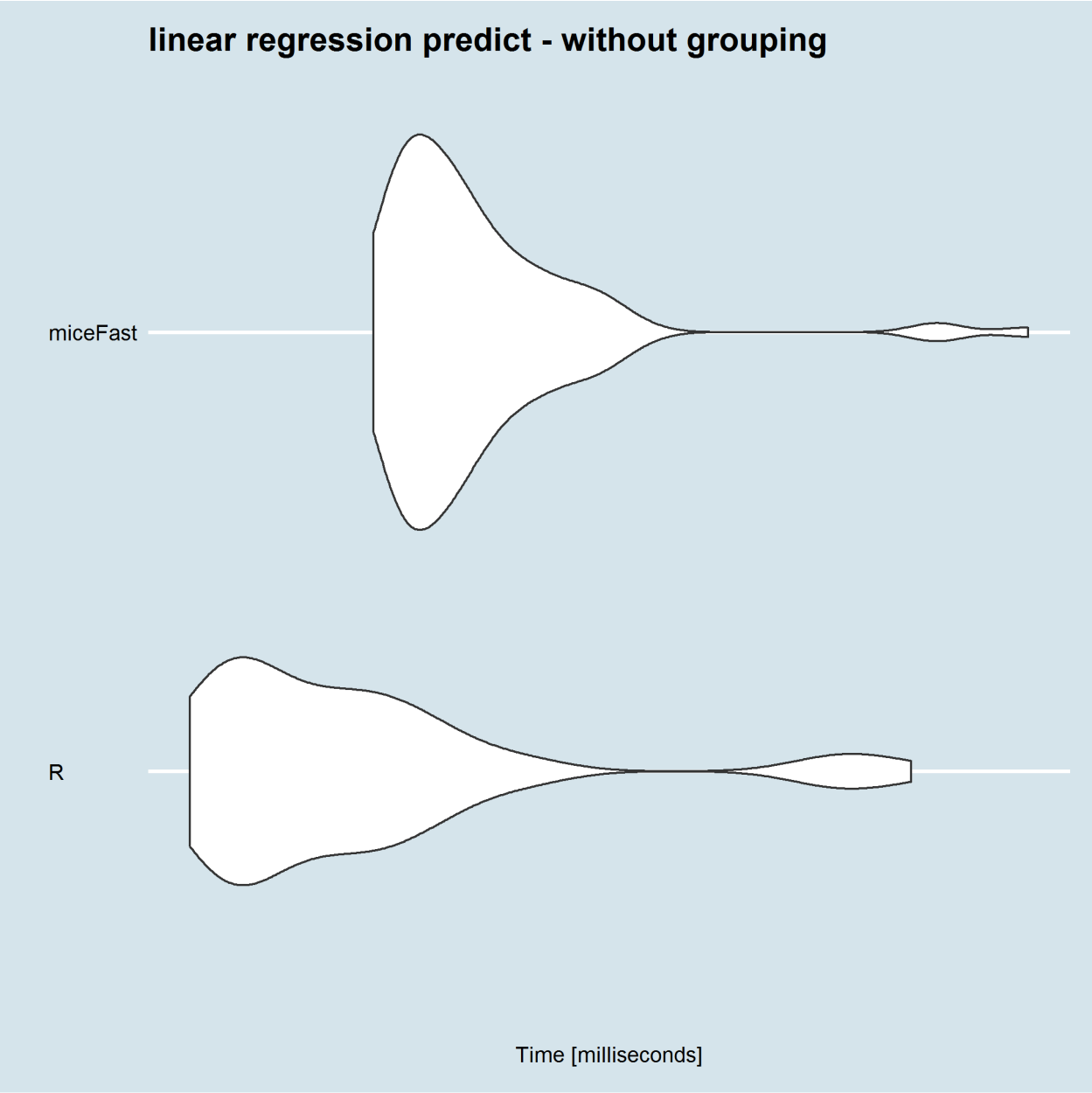
linear regression predict - without grouping
miceFast
R
Time [milliseconds]

Figure 6: “”

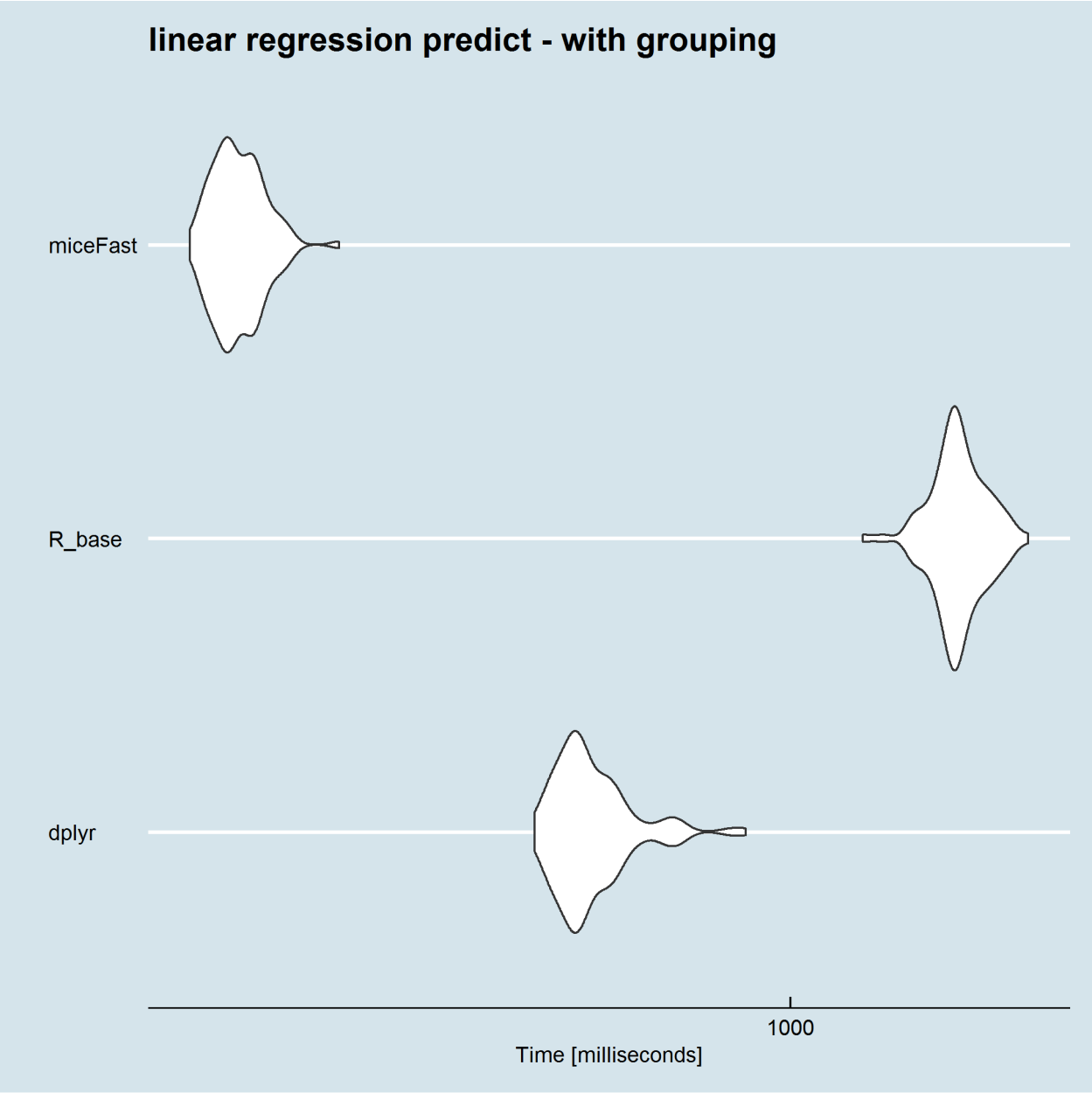
linear regression predict - with grouping
miceFast
R_base
dplyr
1000
Time [milliseconds]

Figure 7: “”