Package 'WeMix'

December 21, 2022

Version 3.3.0

Date 2022-12-20

Title Weighted Mixed-Effects Models Using Multilevel Pseudo Maximum Likelihood Estimation

Maintainer Paul Bailey <pbailey@air.org>

Depends lme4, R (>= 3.5.0)

Imports numDeriv, statmod, Rmpfr, NPflow, Matrix, methods, minqa

Suggests testthat, knitr, rmarkdown, withr, tidyr, EdSurvey (>= 2.6.1)

Description Run mixed-effects models that include weights at every level. The WeMix package fits a weighted mixed model, also known as a multilevel, mixed, or hierarchical linear model (HLM). The weights could be inverse selection probabilities, such as those developed for an education survey where schools are sampled probabilistically, and then students inside of those schools are sampled probabilistically. Although mixed-effects models are already available in R, WeMix is unique in implementing methods for mixed models using weights at multiple levels. Both linear and logit models are supported. Models may have up to three levels.

License GPL-2

VignetteBuilder knitr

ByteCompile true

Note This publication was prepared for NCES under Contract No. ED-IES-12-D-0002 with American Institutes for Research. Mention of trade names, commercial products, or organizations does not imply endorsement by the U.S. government.

RoxygenNote 7.2.1

URL https://american-institutes-for-research.github.io/WeMix/

BugReports https://github.com/American-Institutes-for-Research/WeMix/issues

Encoding UTF-8

NeedsCompilation no

Author Emmanuel Sikali [pdr],
Paul Bailey [aut, cre],
Claire Kelley [aut],
Trang Nguyen [aut],
Huade Huo [aut],
Eric Buehler [ctb],
Christian Kjeldsen [ctb]

Repository CRAN

Date/Publication 2022-12-21 17:50:07 UTC

R topics documented:

`WeMix-package` *Estimate Weighted Mixed-Effects Models*

Description

The WeMix package estimates mixed-effects models (also called multilevel models, mixed models, or HLMs) with survey weights.

Details

This package is unique in allowing users to analyze data that may have unequal selection probability at both the individual and group levels. For linear models, the model is evaluated with a weighted version of the estimating equations used by Bates, Maechler, Bolker, and Walker (2015) in `lme4`. In the non-linear case, WeMix uses numerical integration (Gauss-Hermite and adaptive Gauss-Hermite quadrature) to estimate mixed-effects models with survey weights at all levels of the model. Note that `lme4` is the preferred way to estimate such models when there are no survey weights or weights only at the lowest level, and our estimation starts with parameters estimated in lme4. WeMix is intended for use in cases where there are weights at all levels and is only for use with fully nested data. To start using WeMix, see the vignettes covering the mathematical background of mixed-effects model estimation and use the `mix` function to estimate models. Use `browseVignettes(package="WeMix")` to see the vignettes.

References

Bates, D., Maechler, M., Bolker, B., & Walker, S. (2015). Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software, 67(1), 1-48. doi:10.18637/jss.v067.i01

Rabe-Hesketh, S., & Skrondal, A. (2006) Multilevel Modelling of Complex Survey Data. Journal of the Royal Statistical Society: Series A (Statistics in Society), 169, 805-827. https://doi.org/10.1111/j.1467-985X.2006.00426.x

Bates, D. & Pinheiro, J. C. (1998). Computational Methods for Multilevel Modelling. Bell labs working paper.

`mix` *Survey Weighted Mixed-Effects Models*

Description

Implements a survey weighted mixed-effects model using the provided formula.

Usage

```
mix(
  formula,
  data,
  weights,
  cWeights = FALSE,
  center_group = NULL,
  center_grand = NULL,
  max_iteration = 10,
  nQuad = 13L,
  run = TRUE,
  verbose = FALSE,
  acc0 = 120,
  keepAdapting = FALSE,
  start = NULL,
  fast = FALSE,
  family = NULL
)
```

Arguments

`formula`	a formula object in the style of `lme4` that creates the model.
`data`	a data frame containing the raw data for the model.
`weights`	a character vector of names of weight variables found in the data frame starts with units (level 1) and increasing (larger groups).
`cWeights`	logical, set to `TRUE` to use conditional weights. Otherwise, `mix` expects unconditional weights.
`center_group`	a list where the name of each element is the name of the aggregation level, and the element is a formula of variable names to be group mean centered; for example to group mean center gender and age within the group student: `list("student"= ~gender+age)`, default value of NULL does not perform any group mean centering.
`center_grand`	a formula of variable names to be grand mean centered, for example to center the variable education by overall mean of education: `~education`. Default is NULL which does no centering.

max_iteration	a optional integer, for non-linear models fit by adaptive quadrature which limits number of iterations allowed before quitting. Defaults to 10. This is used because if the liklihood surface is flat, models may run for a very long time without converging.
nQuad	an optional integer number of quadrature points to evaluate models solved by adaptive quadrature. Only non-linear models are evaluated with adaptive quadrature. See notes for additional guidelines.
run	logical; TRUE runs the model while FALSE provides partial output for debugging or testing. Only applies to non-linear models evaluated by adaptive quadrature.
verbose	logical, default FALSE; set to TRUE to print results of intermediate steps of adaptive quadrature. Only applies to non-linear models.
acc0	integer, the precision of mpfr, default 120. Only applies to non-linear models.
keepAdapting	logical, set to TRUE when the adaptive quadrature should adapt after every Newton step. Defaults to FALSE. FALSE should be used for faster (but less accurate) results. Only applies to non-linear models.
start	optional numeric vector representing the point at which the model should start optimization; takes the shape of c(coef, vars) from results (see help).
fast	logical; deprecated
family	the family; optionally used to specify generalized linear mixed models. Currently only binomial(link="logit") is supported.

Details

Linear models are solved using a modification of the analytic solution developed by Bates and Pinheiro (1998). Non-linear models are solved using adaptive quadrature following the method in STATA's GLAMMM (Rabe-Hesketh & Skrondal, 2006). For additional details, see the vignettes *Weighted Mixed Models: Adaptive Quadrature* and *Weighted Mixed Models: Analytical Solution* which provide extensive examples as well as a description of the mathematical basis of the estimation procedure and comparisons to model specifications in other common software. Notes:

- Standard errors of random effect variances are robust; see vignette for details.
- To see the function that is maximized in the estimation of this model, see the section on "Model Fitting" in the *Introduction to Mixed Effect Models With WeMix* vignette.
- When all weights above the individual level are 1, this is similar to a lmer and you should use lme4 because it is much faster.
- If starting coefficients are not provided they are estimated using lme4.
- For non-linear models, when the variance of a random effect is very low (<.1), WeMix doesn't estimate it, because very low variances create problems with numerical evaluation. In these cases, consider estimating without that random effect.
- The model is estimated by maximum likelihood estimation.
- To choose the number of quadrature points for non-linear model evaluation, a balance is needed between accuracy and speed; estimation time increases quadratically with the number of points chosen. In addition, an odd number of points is traditionally used. We recommend starting at 13 and increasing or decreasing as needed.

Value

object of class `WeMixResults`. This is a list with elements:

`lnlf`	function, the likelihood function.
`lnl`	numeric, the log-likelihood of the model.
`coef`	numeric vector, the estimated coefficients of the model.
`ranefs`	the group-level random effects.
`SE`	the standard errors of the fixed effects, robust if robustSE was set to true.
`vars`	numeric vector, the random effect variances.
`theta`	the theta vector.
`call`	the original call used.
`levels`	integer, the number of levels in the model.
`ICC`	numeric, the intraclass correlation coefficient.
`CMODE`	the conditional mean of the random effects.
`invHessian`	inverse of the second derivative of the likelihood function.
`ICC`	the interclass correlation.
`is_adaptive`	logical, indicates if adaptive quadrature was used for estimation.
`sigma`	the sigma value.
`ngroups`	the number of observations in each group.
`varDF`	the variance data frame in the format of the variance data frame returned by lme4.
`varVC`	the variance-covariance matrix of the random effects.
`cov_mat`	the variance-covariance matrix of the fixed effects.
`var_theta`	the variance covariance matrix of the theta terms.
`wgtStats`	statistics regarding weights, by level.
`ranefMat`	list of matrixes; each list element is a matrix of random effects by level with IDs in the rows and random effects in the columns.

Author(s)

Paul Bailey, Claire Kelley, and Trang Nguyen

Examples

```
## Not run:
library(lme4)

data(sleepstudy)
ss1 <- sleepstudy

# Create weights
ss1$W1 <- ifelse(ss1$Subject %in% c(308, 309, 310), 2, 1)
ss1$W2 <- 1
```

```
# Run random-intercept 2-level model
two_level <- mix(Reaction ~ Days + (1|Subject), data=ss1, weights=c("W1", "W2"))

#Run random-intercept 2-level model with group-mean centering
grp_centered <- mix(Reaction ~ Days + (1|Subject), data=ss1,
                    weights = c("W1", "W2"),
                    center_group = list("Subject" = ~Days))

#Run three level model with random slope and intercept.
#add group variables for 3 level model
ss1$Group <- 3
ss1$Group <- ifelse(as.numeric(ss1$Subject) %% 10 < 7, 2, ss1$Group)
ss1$Group <- ifelse(as.numeric(ss1$Subject) %% 10 < 4, 1, ss1$Group)
# level-3 weights
ss1$W3 <- ifelse(ss1$Group == 2, 2, 1)

three_level <- mix(Reaction ~ Days + (1|Subject) + (1+Days|Group), data=ss1,
                   weights=c("W1", "W2", "W3"))

# Conditional Weights
# use vignette example
library(EdSurvey)

#read in data
downloadPISA("~/", year=2012)
cntl <- readPISA("~/PISA/2012", countries="USA")
data <- getData(cntl,c("schoolid","pv1math","st29q03","sc14q02","st04q01",
                       "escs","w_fschwt","w_fstuwt"),
                omittedLevels=FALSE, addAttributes=FALSE)

# Remove NA and omitted Levels
om <- c("Invalid", "N/A", "Missing", "Miss", NA, "(Missing)")
for (i in 1:ncol(data)) {
  data <- data[!data[,i] %in% om,]
}

#relevel factors for model
data$st29q03 <- relevel(data$st29q03, ref="Strongly agree")
data$sc14q02 <- relevel(data$sc14q02, ref="Not at all")

# run with unconditional weights
m1u <- mix(pv1math ~ st29q03 + sc14q02 +st04q01+escs+ (1|schoolid), data=data,
           weights=c("w_fstuwt", "w_fschwt"))
summary(m1u)

# conditional weights
data$pwt2 <- data$w_fschwt
data$pwt1 <- data$w_fstuwt / data$w_fschwt

# run with conditional weights
m1c <- mix(pv1math ~ st29q03 + sc14q02 +st04q01+escs+ (1|schoolid), data=data,
            weights=c("pwt1", "pwt2"), cWeights=TRUE)
```

```
summary(m1c)
# the results are, up to rounding, the same in m1u and m1c, only the calls are different


## End(Not run)
```

`waldTest` *Mixed Model Wald Tests*

Description

This function calculates the Wald test for either fixed effects or variance parameters.

Usage

```
waldTest(fittedModel, type = c("beta", "Lambda"), coefs = NA, hypothesis = NA)
```

Arguments

`fittedModel`	a model of class `WeMixResults` that is the result of a call to `mix`
`type`	a string, one of "beta" (to test the fixed effects) or "Lambda" (to test the variance-covariance parameters for the random effects)
`coefs`	a vector containing the names of the coefficients to test. For `type="beta"` these must be the variable names exactly as they appear in the fixed effects table of the summary. For `type="Lambda"` these must be the names exactly as they appear in the theta element of the fitted model.
`hypothesis`	the hypothesized values of beta or Lambda. If `NA` (the default) 0 will be used.

Details

By default this function tests against the null hypothesis that all coefficients are zero. To identify which coefficients to test use the name exactly as it appears in the summary of the object.

Value

Object of class `WeMixWaldTest`. This is a list with the following elements:

`wald`	the value of the test statistic.
`p`	the p-value for the test statistic. Based on the probabilty of the test statistic under the chi-squared distribution.
`df`	degrees of freedom used to calculate p-value.
`H0`	The vector (for a test of beta) or matrix (for tests of Lambda) containing the null hypothesis for the test.
`HA`	The vector (for a test of beta) or matrix (for tests of Lambda) containing the alternative hypothesis for the test (i.e. the values calculated by the fitted model being tested.)

Examples

```
## Not run:
library(lme4) #to use the example data
sleepstudyU <- sleepstudy
sleepstudyU$weight1L1 <- 1
sleepstudyU$weight1L2 <- 1
wm0 <- mix(Reaction ~ Days + (1|Subject), data=sleepstudyU,
            weights=c("weight1L1", "weight1L2"))
wm1 <- mix(Reaction ~ Days + (1 +Days|Subject), data=sleepstudyU,
           weights=c("weight1L1", "weight1L2"))

waldTest(wm0, type="beta")  #test all betas
#test only beta for days
waldTest(wm0, type="beta", coefs="Days")
#test only beta for intercept against hypothesis that it is 1
waldTest(wm0, type="beta", coefs="(Intercept)", hypothesis=c(1))

waldTest(wm1,type="Lambda")  #test all values of Lambda
 #test only some Lambdas.The names are the same as names(wm1$theta)
waldTest(wm1,type="Lambda", coefs="Subject.(Intercept)")
#specify  test values
waldTest(wm1,type="Lambda", coefs="Subject.(Intercept)", hypothesis=c(1))

## End(Not run)
```

Index