Package ‘geex’

February 17, 2020

Type Package

Title An API for M-Estimation

Version 1.0.12

Author Bradley Saul [aut, cre],
Brian Barkley [ctb]

Maintainer Bradley Saul <bradleysaul@gmail.com>

Description Provides a general, flexible framework for estimating parameters and empirical sandwich variance estimator from a set of unbiased estimating equations (i.e., M-estimation in the vein of Stefanski & Boos (2002) <doi:10.1198/000313002753631330>). All examples from Stefanski & Boos (2002) are published in the corresponding Journal of Statistical Software paper <doi:10.18637/jss.v092.i02>. Also provides an API to compute finite-sample variance corrections.

Depends R (>= 3.3)

Imports Matrix (>= 1.2-6), rootSolve (>= 1.6.6), numDeriv (>= 2014.2-1), lme4 (>= 1.1-12), methods (>= 3.3)

Suggests testthat, knitr, dplyr, moments, sandwich, inferference, xtable, AER, ICSNP, MASS, gee, ivpack, saws, rmarkdown, geepack, covr, mvtnorm

URL https://github.com/bsaul/geex, https://bsaul.github.io/geex/

BugReports https://github.com/bsaul/geex/issues

License MIT + file LICENSE

LazyData TRUE

VignetteBuilder knitr

RoxygenNote 7.0.2

NeedsCompilation no

Repository CRAN

Date/Publication 2020-02-17 15:10:05 UTC

R topics documented:

`geex-package` *geex: M-estimation API*

Description

geex provides an extensible API for estimating parameters and their covariance from a set of estimating functions (M-estimation). M-estimation theory has a long history [see reference in the M-estimation bibliography: https://bsaul.github.io/geex/articles/articles/mestimation_bib.html. For an excellent introduction, see the primer by L.A. Stefanski and D.D. Boos, "The Calculus of M-estimation" (The American Statistician (2002), 56(1), 29-38) (http://www.jstor.org/stable/3087324).

Details

M-estimation encompasses a broad swath of statistical estimators and ideas including:

- the empirical "sandwich" variance estimator
- generalized estimating equations (GEE)
- many maximum likelihood estimators
- robust regression
- and many more

geex can implement all of these using a user-defined estimating function.

To learn more about geex, see the package vignettes: `browseVignettes(package = 'geex')`.

Goals

If you can specify a set of unbiased estimating equations, geex does the rest. The goals of geex are simply:

- To minimize the translational distance between a set of estimating functions and R code;
- To return numerically accurate point and covariance estimates from a set of unbiased estimating functions.

geex does not, by itself, necessarily aim to be fast nor precise. Such goals are left to the user to implement or confirm.

Author(s)

Maintainer: Bradley Saul <bradleysaul@gmail.com>

Other contributors:

- Brian Barkley [contributor]

References

Saul, Bradley C., and Michael G. Hudgens. (2020). "The Calculus of M-estimation in R with geex." Journal of Statistical Software 92(2), 1-15. doi:10.18637/jss.v092.i02

See Also

Useful links:

- https://github.com/bsaul/geex
- https://bsaul.github.io/geex/
- Report bugs at https://github.com/bsaul/geex/issues

`approx_control-class` *approx_control S4 class*

Description

EXPERIMENTAL. See example 7 in `vignette("01_additional_examples",package = "geex")` for usage.

Slots

`.FUN` a function which approximates an `estFUN`.

`.options` a list of options passed to `.FUN`.

`basic_control-class` *basic_control S4 class*

Description

A general class for defining a `function`, and the options passed to the function

Slots

`.FUN` a function

`.options` a list of options passed to `.FUN`

See Also

`root_control-class`, `deriv_control-class` `approx_control-class`

coef,geex-method — *Gets the parameter estimates from a geex object*

Description

Gets the parameter estimates from a geex object

Usage

```
## S4 method for signature 'geex'
coef(object)

## S4 method for signature 'geex_summary'
coef(object)
```

Arguments

object	a geex object

Examples

```
ex_eeFUN <- function(data){
 function(theta){
   with(data,
    c(Y1 - theta[1],
     (Y1 - theta[1])^2 - theta[2] ))
}}

results <- m_estimate(
 estFUN = ex_eeFUN,
 data  = geexex,
 root_control = setup_root_control(start = c(1,1)))

coef(results)
```

compute_pairwise_sum_of_list — *Compute the sum of a list of matrices to sum*

Description

Compute the sum of a list of matrices to sum

Usage

```
compute_pairwise_sum_of_list(.l, .w = NULL, .wFUN = NULL, ...)
```

Arguments

`.l`	a list of matrices
`.w`	a numeric vector of weights
`.wFUN`	a function of `i`, `j`, and (optionally) additional arguments
`...`	additional arguments passed to `.wFUN` Either `.w` or `.wFUN` must be specified but not both.

`compute_sigma` *Compute empirical sandwich covariate estimator*

Description

Computes $\Sigma = A^{-1}B(A^{-1})^T$ with provided A and B matrices.

Usage

```
compute_sigma(A, B)
```

Arguments

`A`	a matrix, generally the `.A` slot in a `sandwich_components` object created in `estimate_sandwich_matrices`
`B`	a matrix, generally the `.B` slot in a `sandwich_components` object created in `estimate_sandwich_matrices`

Value

the `matrix` `Ainv` `%*%` `B` `%*%` `t(Ainv)`

Examples

```
A <- diag(2, nrow = 2, ncol = 2)
B <- matrix(4, nrow = 2, ncol = 2)
compute_sigma(A = A, B = B)
```

compute_sum_of_list — *Compute the sum of a list of matrices to sum*

Description

Compute the sum of a list of matrices to sum

Usage

```
compute_sum_of_list(.l, .w = numeric(0))
```

Arguments

`.l`	a list of matrices
`.w`	a numeric vector of weights

correction — *Creates a correct_control object*

Description

Creates a correct_control object

Usage

```
correction(FUN, ...)
```

Arguments

`FUN`	a correction to perform. `components` must be the first argument
`...`	additional arguments passed to `FUN`

Value

a `correct_control` object

Examples

```
correction(FUN = fay_bias_correction, b = 0.75)
```

correct_by *Correct sandwich components*

Description

Modifies the matrices in a sandwich_components object using the function and options in a correct_control object. The function correction is a utility for creating correct_control objects.

Usage

```
correct_by(.components, .correct_control)
```

Arguments

.components	an object of class sandwich_components
.correct_control	an object of class correct_control

Details

See the finite sample corrections vignette for further examples.

Value

the result of .FUN in .correct_control.

See Also

fay_bias_correction and fay_df_correction for corrections provided by geex

Examples

```
myee <- function(data){
   function(theta){
    c(data$Y1 - theta[1],
     (data$Y1 - theta[1])^2 - theta[2])
   }
 }
mybasis <- create_basis(
   estFUN = myee,
   data   = geexex)
mats <- estimate_sandwich_matrices(mybasis, .theta = c(5.04, 10.04))
correct_by(mats,
   .correct_control =  correction(fay_bias_correction, b = .75))
```

`correct_control-class` *correct_control S4 class*

Description

correct_control S4 class

Slots

`.FUN` a function which "corrects" a `sandwich_components` object. Usually a small-sample correction

`.options` a list of options passed to `.FUN`.

`create_basis` *Creates an m_estimation_basis object*

Description

Creates an m_estimation_basis object

Usage

```
create_basis(estFUN, data, units, outer_args, inner_args)
```

Arguments

`estFUN`	a function that takes in group-level data and returns a function that takes parameters as its first argument
`data`	a data.frame
`units`	an optional character string identifying the grouping variable in `data`
`outer_args`	a list of arguments passed to the outer (data) function of `estFUN`. (optional)
`inner_args`	a list of arguments passed to the inner (theta) function of `estFUN`. (optional)

Details

Either `data` or `split_data` must be provided

Value

a `m_estimation_basis`

Examples

```
myee <- function(data){
   function(theta){
    c(data$Y1 - theta[1],
     (data$Y1 - theta[1])^2 - theta[2])
   }
 }
mybasis <- create_basis(
   estFUN = myee,
   data   = geexex)
```

---

create_GFUN   *Creates a function that sums over psi functions*

Description

From a list of $\psi(O_i, \theta)$ for i = 1, ..., m, creates $G_m = \sum_i \psi(O_i, \theta)$, called GFUN. Here, $\psi(O_i, \theta)$ is the *inner* part of an estFUN, in that the data is fixed and G_m is a function of θ).

Usage

```
create_GFUN(object, ...)

## S4 method for signature 'm_estimation_basis'
create_GFUN(object)
```

Arguments

object	an object of class m_estimation_basis
...	additional arguments passed to other methods

Examples

```
myee <- function(data){
   function(theta){
    c(data$Y1 - theta[1],
     (data$Y1 - theta[1])^2 - theta[2])
   }
 }
mybasis <- create_basis(
   estFUN = myee,
   data   = geexex)
f <- grab_GFUN(create_GFUN(mybasis))

# Evaluate GFUN at mean and variance: should be close to zero
n <- nrow(geexex)
f(c(mean(geexex$Y1), var(geexex$Y1) * (n - 1)/n))
```

`create_psiFUN_list` *Creates list of psi functions*

Description

Creates the estimating function ($\psi(O_i, \theta)$) for each unit. That is, this function evaluates the outer function in `estFUN` for each independent unit and a returns the inner function in `estFUN`.

Usage

```
create_psiFUN_list(object, ...)

## S4 method for signature 'm_estimation_basis'
create_psiFUN_list(object)
```

Arguments

`object`	an object of class `m_estimation_basis`
`...`	additional arguments passed to other methods

Value

the `object` with the `.psiFUN_list` slot populated.

Examples

```
myee <- function(data){
   function(theta){
    c(data$Y1 - theta[1],
     (data$Y1 - theta[1])^2 - theta[2])
   }
 }
mybasis <- create_basis(
   estFUN = myee,
   data   = geexex)
psi_list <- grab_psiFUN_list(create_psiFUN_list(mybasis))

# A list of functions
head(psi_list)
```

deriv_control-class *deriv_control S4 class*

Description

deriv_control S4 class

Slots

`.FUN` a function which computes a numerical derivation. This functions first argument must the function on which the derivative is being compute. Defaults to `jacobian`.

`.options` a list of options passed to `.FUN`. Defaults to `list(method = 'Richardson')`

diagnose_roots *Diagnose roots of estimating equations*

Description

Computes the value of

$$G_m = sum_i psi(O_i, \hat{\theta})$$

, i.e., the estimating equations at `theta`. Used to verify that $G_m = 0$ (or close to 0).

Usage

```
diagnose_roots(GFUN, theta)
```

Arguments

`GFUN`	a function of theta
`theta`	parameter estimates to use in evaluating the estimating equations.

Value

a numeric vector

Examples

```
myee <- function(data){
   function(theta){
    c(data$Y1 - theta[1],
     (data$Y1 - theta[1])^2 - theta[2])
   }
 }

mest <- m_estimate(
```

```
   estFUN = myee,
   data   = geexex,
   root_control = setup_root_control(start = c(1, 1)))

f <- grab_GFUN(mest@basis)
# Should be close to zero
diagnose_roots(GFUN = f, theta = roots(mest))
```

---

estimate_GFUN_roots *Estimate roots for a set of estimating equations*

Description

Using the rootFUN specified by the user (defaults to multiroot), this function estimates the roots of the equations:

$$G_m = sum_i psi(O_i, \hat{\theta}) = 0$$

Usage

```
estimate_GFUN_roots(.basis)
```

Arguments

.basis	an object of class m_estimation_basis

Details

This is primilary an internal function used within m_estimate, but it is exported for use in debugging and development.

For an example of how to use a different rootFUN, see the root solver vignette, vignette('geex_root_solvers',package = 'geex').

Value

the output of the rootFUN function

Examples

```
myee <- function(data){
  function(theta){
    c(data$Y1 - theta[1],
     (data$Y1 - theta[1])^2 - theta[2])
   }
 }

# Start with a basic basis
mybasis <- create_basis(
  estFUN = myee,
```

```
  data   = geexex)

# Add a control for the root solver
mycontrol <- new('geex_control', .root = setup_root_control(start = c(1, 1)))
mybasis@.control <- mycontrol

# Now estimate roots of GFUN
roots <- estimate_GFUN_roots(mybasis)
roots
```

---

estimate_sandwich_matrices   *Estimate component matrices of the empirical sandwich covariance estimator*

Description

For a given set of estimating equations computes the 'meat' (B_m in Stefanski and Boos notation) and 'bread' (A_m in Stefanski and Boos notation) matrices necessary to compute the covariance matrix.

Usage

```
estimate_sandwich_matrices(.basis, .theta)
```

Arguments

.basis	basis an object of class m_estimation_basis
.theta	vector of parameter estimates (i.e. estimated roots)

Details

For a set of estimating equations ($\sum_i \psi(O_i, \theta) = 0$), this function computes:

$$A_i = \partial \psi(O_i, \theta)/\partial \theta$$

$$A = \sum_i A_i$$

$$B_i = \psi(O_i, \theta)\psi(O_i, \theta)^T$$

$$B = \sum_i B_i$$

where all of the above are evaluated at $\hat{\theta}$. The partial derivatives in A_i numerically approximated by the function defined in deriv_control.

Note that $A = \sum_i A_i$ and not $\sum_i A_i/m$, and the same for B.

Value

a `sandwich_components` object

References

Stefanski, L. A., & Boos, D. D. (2002). The calculus of m-estimation. The American Statistician, 56(1), 29-38.

Examples

```
myee <- function(data){
  function(theta){
    c(data$Y1 - theta[1],
     (data$Y1 - theta[1])^2 - theta[2])
   }
 }

# Start with a basic basis
mybasis <- create_basis(
  estFUN = myee,
  data   = geexex)

# Now estimate sandwich matrices
estimate_sandwich_matrices(
 mybasis, c(mean(geexex$Y1), var(geexex$Y1)))
```

`estimating_function-class` *estimating_function S4 class*

Description

estimating_function S4 class

Slots

`.estFUN` the estimating function.

`.outer_args` a named `list` of arguments passed to the outer function of `.estFUN`. Should *not* include the `data` argument.

`.inner_args` a named `list` of arguments passed to the inner function of `.estFUN`. Should *not* include the `theta` argument.

fay_bias_correction *Correct sandwich variance estimator byFay's bias correction*

Description

Computes the bias corrected sandwich covariance matrix described in Fay and Graubard (2001). See `vignette("05_finite_sample_corrections",package = "geex")` for further information.

Usage

```
fay_bias_correction(components, b = 0.75)
```

Arguments

components	an object of class `sandwich_components`
b	a numeric value < 1. Defaults to 0.75 as in Fay.

Value

a corrected covariance matrix

References

Fay, M. P., & Graubard, B. I. (2001). Small-Sample adjustments for Wald-type tests using sandwich estimators. Biometrics, 57(4), 1198-1206

Examples

```
# This example demonstrates usage of the corrections, not a meaningful application
myee <- function(data){
 function(theta){
   c(data$Y1 - theta[1],
   (data$Y1 - theta[1])^2 - theta[2])
  }
}

results <- m_estimate(
   estFUN = myee,
   data = geexex,
   root_control = setup_root_control(start = c(1,1)),
   corrections  = list(
     bias_correction_.1 = correction(fay_bias_correction, b = .1),
     bias_correction_.3 = correction(fay_bias_correction, b = .3))
   )

get_corrections(results)
```

fay_df_correction *Correct sandwich variance inference by Fay's degrees of freedom correction*

Description

Computes the degrees of freedom correction described in Fay and Graubard (2001). See `vignette("05_finite_sample_cor` `= "geex")` for further information.

Usage

```
fay_df_correction(components, b = 0.75, L, version)
```

Arguments

`components`	an object of class `sandwich_components`
`b`	a numeric value < 1. Defaults to 0.75 as in Fay.
`L`	a k x p matrix where p is the dimension of theta
`version`	either 1 or 2, corresponding to hat(d) or tilde(d), respectively

Value

a scalar corresponding to the estimated degrees of freedom

References

Fay, M. P., & Graubard, B. I. (2001). Small-Sample adjustments for Wald-type tests using sandwich estimators. Biometrics, 57(4), 1198-1206

Examples

```
# This example demonstrates usage of the corrections, not a meaningful application
myee <- function(data){
 function(theta){
   c(data$Y1 - theta[1],
   (data$Y1 - theta[1])^2 - theta[2])
  }
}

results <- m_estimate(
   estFUN = myee,
   data = geexex,
   root_control = setup_root_control(start = c(1,1)),
   corrections  = list(
     df_correction1 = correction(fay_df_correction,
                        b = .75, L = c(0, 1), version = 1 ),
     df_correction2 = correction(fay_df_correction,
```

```
                                b = .75, L = c(0, 1), version = 2 ))
    )

get_corrections(results)
```

geex-class *geex S4 class*

Description

geex S4 class

Slots

call the m_estimate call

basis a m_estimation_basis object

rootFUN_results the results of call to the root finding algorithm function

sandwich_components a sandwich_components object

GFUN the function of which the roots are computed.

corrections a list of correction performed on sandwich_components

estimates a numeric vector of parameter estimates

vcov the empirical sandwich variance matrix

geexex *Dataset used to illustrate Stefanski and Boos examples.*

Description

The data used to illustrate examples 1-9 of Stefanski and Boos (2002).

Format

a dataset with 9 variables and 100 observations

- Y1 rnorm(mean = 5, sd = 4)
- Y2 rnorm(mean = 2, sd = 1)
- X1 rgamma(shape =5)
- Y3 2 + 3*X1 + 1*rnorm(0, 1)
- W1 X1 + 0.25 * rnorm(0, 1)
- Z1 2 + 1.5*X1 + 1*rnorm(0, 1)
- X2 0 for first 50 observation, 1 for rest
- Y4 0.1 + 0.1*X1 + 0.5*X2 + rnorm(0, 1)
- Y5 rbinom(prob = plogis(0.1 + 0.1*X1 + 0.5*X2))

References

Stefanski, L. A., & Boos, D. D. (2002). The calculus of m-estimation. The American Statistician, 56(1), 29-38.

---

geex_control-class *geex_control S4 class*

Description

An object which control all the basic_control objects necessary to perform M-estimation

Slots

.approx an approx_control object

.root a root_control object

.deriv a deriv_control object

---

geex_summary-class *geex summary object*

Description

geex summary object

Slots

estFUN a estimating-function

outer_args the list arguments passed to the m_estimate call

inner_args the list arguments passed to the m_estimate call

data the data.frame passed to the m_estimate call

weights the weights passed to the m_estimate call

nobs the number of observational units used to compute the M-estimator

units the name of the variable identifying the observational units

corrections a list of correction performed on sandwich_components

estimates a numeric vector of parameter estimates

vcov the empirical sandwich variance matrix `get_corrections` *Gets the corrections from a geex object*

Description

Gets the corrections from a geex object

Usage

```
get_corrections(object, ...)

## S4 method for signature 'geex'
get_corrections(object)

## S4 method for signature 'geex_summary'
get_corrections(object)
```

Arguments

`object`	a `geex` object
`...`	arguments passed to other methods

Examples

```
myee <- function(data){
 function(theta){
   c(data$Y1 - theta[1],
   (data$Y1 - theta[1])^2 - theta[2])
  }
}

results <- m_estimate(
   estFUN = myee,
   data = geexex,
   root_control = setup_root_control(start = c(1,1)),
   corrections  = list(
     bias_correction_.1 = correction(fay_bias_correction, b = .1),
     bias_correction_.3 = correction(fay_bias_correction, b = .3))
   )

get_corrections(results)
```

grab — *Grab something from an object*

Description

Grab something from an object

Usage

```
grab(from, what, ...)
```

Arguments

`from`	an object
`what`	what to grab one of 'response', 'design_matrix', 'response_formula', 'fixed_formula', 'eeFUN'
`...`	additional arguments passed to `grab_**` function

See Also

`grab_response`, `grab_design_matrix`, `grab_response_formula`, `grab_fixed_formula`, `grab_design_levels`

grab_bread — *Grabs the .A (bread matrix) slot*

Description

Grabs the .A (bread matrix) slot

Usage

```
grab_bread(object)

## S4 method for signature 'sandwich_components'
grab_bread(object)
```

Arguments

`object`	a `sandwich_components` object

Examples

```
myee <- function(data){
 function(theta){
   c(data$Y1 - theta[1],
   (data$Y1 - theta[1])^2 - theta[2])
  }
}

results <- m_estimate(
   estFUN = myee,
   data = geexex,
   root_control = setup_root_control(start = c(1,1)))

grab_bread(results@sandwich_components)
```

---

grab_bread_list *Gets the .A_i (list of bread matrices) slot*

Description

Gets the .A_i (list of bread matrices) slot

Usage

```
grab_bread_list(object)

## S4 method for signature 'sandwich_components'
grab_bread_list(object)
```

Arguments

object	a sandwich_components object

Examples

```
myee <- function(data){
 function(theta){
   c(data$Y1 - theta[1],
   (data$Y1 - theta[1])^2 - theta[2])
  }
}

results <- m_estimate(
   estFUN = myee,
   data = geexex,
   root_control = setup_root_control(start = c(1,1)))

head(grab_bread_list(results@sandwich_components))
```

grab_design_levels *Grab a list of the levels of factor variables in a model.*

Description

Useful when splitting data later, used with grab_design_matrix or especially when calling grab_psiFUN from within an eeFun.

Usage

```
grab_design_levels(model)
```

Arguments

model	a model object such as lm, glm, merMod

Value

A named list of character vectors that provides the fentire set of levels that each factor predictor in model will take on. This is hopefully identical to what the xlev argument to link[stats]{model.frame} desires. When model has no factors as predictors, then an empty list is returned.

Examples

```
## Not run:
  geex::grab_design_matrix(
    data = data,
    rhs_formula = geex::grab_fixed_formula(model),
    xlev = geex::grab_design_levels(model)
  )
  ## Below is helpful within an eeFun.
  geex::grab_psiFUN(
    data = data,## Especially when this is a subset of the data
    rhs_formula = geex::grab_fixed_formula(model),
    xlev = geex::grab_design_levels(model)
  )

## End(Not run)
```

grab_design_matrix *Grab a matrix of fixed effects from a model object*

Description

Grab a matrix of fixed effects from a model object

Usage

```
grab_design_matrix(data, rhs_formula, ...)
```

Arguments

`data`	the data from which to extract the matrix
`rhs_formula`	the right hand side of a model formula
`...`	Can be used to pass `xlev` to `model.frame`

Examples

```
# Create a "desigm" matrix for the first ten rows of iris data
fit <- lm(Sepal.Width ~ Petal.Width, data = iris)
grab_design_matrix(
  data = iris[1:10, ],
  grab_fixed_formula(fit))
```

`grab_ee_list` *Gets the .ee_i (observed estimating function) slot*

Description

Gets the .ee_i (observed estimating function) slot

Usage

```
grab_ee_list(object)
```

Arguments

`object`	a `sandwich_components` object

Examples

```
myee <- function(data){
 function(theta){
   c(data$Y1 - theta[1],
   (data$Y1 - theta[1])^2 - theta[2])
  }
}

results <- m_estimate(
   estFUN = myee,
   data = geexex,
   root_control = setup_root_control(start = c(1,1)))

grab_ee_list(results@sandwich_components)
```

grab_estFUN — *Grab estimating functions from a model object*

Description

Grab estimating functions from a model object

Usage

```
grab_estFUN(object)

## S4 method for signature 'estimating_function'
grab_estFUN(object)
```

Arguments

`object`	a `estimating_function` object

grab_fixed_formula — *Grab the RHS formula from a model object*

Description

Grab the RHS formula from a model object

Usage

```
grab_fixed_formula(model)
```

Arguments

`model`	a model object such as `lm`, `glm`, `merMod`

Examples

```
fit <- lm(Sepal.Width ~ Petal.Width, data = iris)
grab_fixed_formula(fit)
```

grab_GFUN *Gets the .psi_list slot in a m_estimation_basis*

Description

Gets the .psi_list slot in a m_estimation_basis

Usage

```
grab_GFUN(object)

## S4 method for signature 'm_estimation_basis'
grab_GFUN(object)

## S4 method for signature 'geex'
grab_GFUN(object)
```

Arguments

`object`	a `m_estimation_basis` object

grab_meat *Gets the .B (meat matrix) slot*

Description

Gets the .B (meat matrix) slot

Usage

```
grab_meat(object)

## S4 method for signature 'sandwich_components'
grab_meat(object)
```

Arguments

`object`	a `sandwich_components` object

Examples

```
myee <- function(data){
 function(theta){
   c(data$Y1 - theta[1],
   (data$Y1 - theta[1])^2 - theta[2])
  }
}

results <- m_estimate(
   estFUN = myee,
   data = geexex,
   root_control = setup_root_control(start = c(1,1)))

grab_meat_list(results@sandwich_components)
```

`grab_meat_list` *Gets the .B_i (list of bread matrices) slot*

Description

Gets the .B_i (list of bread matrices) slot

Usage

```
grab_meat_list(object)

## S4 method for signature 'sandwich_components'
grab_meat_list(object)

## S4 method for signature 'sandwich_components'
grab_ee_list(object)
```

Arguments

`object`	a `sandwich_components` object

Examples

```
myee <- function(data){
 function(theta){
   c(data$Y1 - theta[1],
   (data$Y1 - theta[1])^2 - theta[2])
  }
}

results <- m_estimate(
   estFUN = myee,
   data = geexex,
   root_control = setup_root_control(start = c(1,1)))
```

```
head(grab_meat_list(results@sandwich_components))
```

grab_psiFUN *Grab estimating functions from a model object*

Description

Grab estimating functions from a model object

Usage

```
grab_psiFUN(object, ...)

## S3 method for class 'glm'
grab_psiFUN(object, data, ...)

## S3 method for class 'geeglm'
grab_psiFUN(object, data, ...)

## S3 method for class 'merMod'
grab_psiFUN(object, data, numderiv_opts = NULL, ...)
```

Arguments

`object`	the object from which to extrace `psiFUN`
`...`	additonal arguments passed to other methods
`data`	the data to use for the estimating function
`numderiv_opts`	a list of arguments passed to `numDeriv::grad`

Methods (by class)

- `glm`: Create estimating equation function from a `glm` object
- `geeglm`: Create estimating equation function from a `geeglm` object
- `merMod`: Create estimating equation function from a `merMod` object

Examples

```
## Not run:
library(geepack)
data('ohio')

glmfit  <- glm(resp ~ age, data = ohio,
               family = binomial(link = "logit"))
geefit  <- geeglm(resp ~ age, data = ohio, id = id,
                  family = binomial(link = "logit"))
```

```
glmmfit <- glmer(resp ~ age + (1|id), data = ohio,
                 family = binomial(link = "logit"))
example_ee <- function(data, model){
 f <- grab_psiFUN(model, data)
 function(theta){
  f(theta)
 }
}

m_estimate(
  estFUN = example_ee,
  data = ohio,
  compute_roots = FALSE,
  units = 'id',
  roots = coef(glmfit),
  outer_args = list(model = glmfit))
m_estimate(
  estFUN = example_ee,
  data = ohio,
  compute_roots = FALSE,
  units = 'id',
  roots = coef(geefit),
  outer_args = list(model = geefit))
m_estimate(
  estFUN = example_ee,
  data = ohio,
  compute_roots = FALSE,
  units = 'id',
  roots = unlist(getME(glmmfit, c('beta', 'theta'))),
  outer_args = list(model = glmmfit))

## End(Not run)
```

`grab_psiFUN_list` *Gets the .psi_list slot in a m_estimation_basis*

Description

Gets the .psi_list slot in a m_estimation_basis

Usage

```
grab_psiFUN_list(object)

## S4 method for signature 'm_estimation_basis'
grab_psiFUN_list(object)

## S4 method for signature 'geex'
grab_psiFUN_list(object)
```

Arguments

`object`	a `m_estimation_basis` object

grab_response — *Grab a vector of responses from a model object*

Description

Grab a vector of responses from a model object

Usage

```
grab_response(data, formula)
```

Arguments

`data`	data.frame from which to extract the vector of responses
`formula`	model formula

Examples

```
# Grab vector of responses for the first ten rows of iris data
fit <- lm(Sepal.Width ~ Petal.Width, data = iris)
grab_response(
  data = iris[1:10, ],
  formula(fit))
```

grab_response_formula — *Grab the LHS formula from a model object*

Description

Grab the LHS formula from a model object

Usage

```
grab_response_formula(model)
```

Arguments

`model`	a model object such as `lm`, `glm`, `merMod`

Examples

```
fit <- lm(Sepal.Width ~ Petal.Width, data = iris)
grab_response_formula(fit)
```

m_estimate — *Estimate parameters and their covariance from a set of estimating equations*

Description

M-estimation theory provides a framework for asympotic properties of estimators that are solutions to estimating equations. Many R packages implement specific applications of estimating equations. **geex** aims to be provide a more general framework that any modelling method can use to compute point and variance estimates for parameters that are solutions to estimating equations of the form:

$$\sum_i \psi(O_i, \hat{\theta}) = 0$$

Usage

```
m_estimate(
  estFUN,
  data,
  units = character(0),
  weights = numeric(0),
  outer_args = list(),
  inner_args = list(),
  roots = NULL,
  compute_roots = TRUE,
  compute_vcov = TRUE,
  corrections,
  deriv_control,
  root_control,
  approx_control
)
```

Arguments

`estFUN`	a function that takes in group-level data and returns a function that takes parameters as its first argument
`data`	a data.frame
`units`	an optional character string identifying the grouping variable in `data`
`weights`	an optional vector of weights. See details.
`outer_args`	a list of arguments passed to the outer (data) function of `estFUN`. (optional)
`inner_args`	a list of arguments passed to the inner (theta) function of `estFUN`. (optional)
`roots`	a vector of parameter estimates must be provided if `compute_roots = FALSE`
`compute_roots`	whether or not to find the roots of the estimating equations. Defaults to `TRUE`.
`compute_vcov`	whether or not to compute the variance-covariance matrix. Defaults to `TRUE`.

corrections — an optional list of small sample corrections where each list element is a correct_control object which contains two elements: correctFUN and correctFUN_options. The function correction constructs correct_control objects. See details for more information.

deriv_control — a deriv_control object

root_control — a root_control object

approx_control — a approx_control object

Details

The basic idea of **geex** is for the analyst to provide at least two items:

- data
- estFUN: (the ψ function), a function that takes unit-level data and returns a function in terms of parameters (θ)

With the estFUN, **geex** computes the roots of the estimating equations and/or the empirical sandwich variance estimator.

The root finding algorithm defaults to multiroot to estimate roots though the solver algorithm can be specified in the rootFUN argument. Starting values for multiroot are passed via the root_control argument. See vignette("v03_root_solvers",package = "geex") for information on customizing the root solver function.

To compute only the covariance matrix, set compute_roots = FALSE and pass estimates of θ via the roots argument.

M-estimation is often used for clustered data, and a variable by which to split the data.frame into independent units is specified by the units argument. This argument defaults to NULL, in which case the number of units equals the number of rows in the data.frame.

For information on the finite-sample corrections, refer to the finite sample correction API vignette: vignette("v05_finite_sample_corrections",package = "geex")

Value

a geex object

Writing an estFUN

Description: An estFUN is a function representing ψ. **geex** works by breaking ψ into two parts:

- the "outer" part of the estFUN which manipulates data and outer_args and returns an
- "inner" function of theta and inner_args. Internally, this "inner" function is called psiFUN.

In pseudo-code this looks like:

```
function(data, <<outer_args>>){
  O <- manipulate(data, <<outer_args>>)
  function(theta, <<inner_args>>){
    map(O, to = theta, and = <<inner_args>>)
  }
}
```

See the examples below or the package vignettes to see an `estFUN` in action.

Importantly, the `data` used in an `estFUN` is *unit* level data, which may be single rows in a data.frame or block of rows for clustered data.

Additional arguments: Additional arguments may be passed to both the inner and outer function of the `estFUN`. Elements in an `outer_args` list are passed to the outer function; any elements of the `inner_args` list are passed to the inner function. For an example, see the finite sample correction vignette [`vignette("v05_finite_sample_corrections",package = "geex")`].

Setting up root_control

To estimate roots of the estimating functions, **geex** uses the **rootSolve** `multiroot` function by default, which requires starting values. The `root_control` argument expects a `root_control` object, which the utility function `setup_root_control` aids in creating. For example, `setup_root_control(start = 4)` creates a `root_control` setting the starting value to 4. In general, the dimension of `start` must the same as `theta` in the inner `estFUN`.

Using weights

In some situations, use of weights can massively speed computations. Refer to `vignette("v04_weights",package = "geex")` for an example.

References

Stefanski, L. A., & Boos, D. D. (2002). The calculus of M-estimation. The American Statistician, 56(1), 29-38.

Examples

```
# Estimate the mean and variance of Y1 in the geexex dataset
ex_eeFUN <- function(data){
 function(theta){
   with(data,
    c(Y1 - theta[1],
     (Y1 - theta[1])^2 - theta[2] ))
}}

m_estimate(
 estFUN = ex_eeFUN,
 data  = geexex,
 root_control = setup_root_control(start = c(1,1)))

# compare to the mean() and variance() functions
mean(geexex$Y1)
n <- nrow(geexex)
var(geexex$Y1) * (n - 1)/n

# A simple linear model for regressing X1 and X2 on Y4
lm_eefun <- function(data){
 X <- cbind(1, data$X1, data$X2)
 Y <- data$Y4
```

```
 function(theta){
    t(X) %*% (Y - X %*% theta)
   }
 }

m_estimate(
 estFUN = lm_eefun,
 data  = geexex,
 root_control = setup_root_control(start = c(0, 0, 0)))

# Compare to lm() results
summary(lm(Y4 ~ X1 + X2, data = geexex))
```

`m_estimation_basis-class` *m_estimation_basis S4 class*

Description

m_estimation_basis S4 class

Slots

`.data` the analysis data.frame

`.units` an (optional) character string identifying the variable in `.data` which splits the data into indepedent units

`.weights` a numeric vector of weights used in weighting the estimating functions

`.psiFUN_list` a list of `psiFUN`s created by `create_psiFUN_list`

`.GFUN` a function created by `create_GFUN`

`.control` a `geex_control` object

`nobs,geex-method` *Extract the number observations*

Description

Extract the number observations

Usage

```
## S4 method for signature 'geex'
nobs(object)

## S4 method for signature 'geex_summary'
nobs(object)
```

Arguments

`object`	a geex object

Examples

```
## Not run:
library(geepack)
data('ohio')

glmfit  <- glm(resp ~ age, data = ohio,
              family = binomial(link = "logit"))
z  <- m_estimate(
  estFUN = example_ee,
  data = ohio,
  compute_roots = FALSE,
  units = 'id',
  roots = coef(glmfit),
  outer_args = list(model = glmfit))

nobs(z)

## End(Not run)
```

---

roots *Gets the parameter estimates matrix from a geex object*

Description

Gets the parameter estimates matrix from a geex object

Usage

```
roots(object, ...)

## S4 method for signature 'geex'
roots(object)

## S4 method for signature 'geex_summary'
roots(object)
```

Arguments

`object`	a geex object
`...`	arguments passed to other methods

Examples

```
ex_eeFUN <- function(data){
 function(theta){
   with(data,
    c(Y1 - theta[1],
     (Y1 - theta[1])^2 - theta[2] ))
}}

results <- m_estimate(
 estFUN = ex_eeFUN,
 data  = geexex,
 root_control = setup_root_control(start = c(1,1)))

roots(results)
```

---

root_control-class *root_control S4 class*

Description

root_control S4 class

Slots

`.FUN` a root finding function whose first argument must be named `f`.

`.options` a list of options passed to `.FUN`.

`.object_name` a character string identifying the object containing the roots in the output of `.FUN`.

---

sandwich_components-class *sandwich_components S4 class*

Description

sandwich_components S4 class

Slots

`.A` the "bread" matrix

`.A_i` a list of "bread" matrices per unit

`.B` the "meat" matrix

`.B_i` a list of "meat" matrices per unit

`.ee_i` a list of observed estimating function values per unit

setup_approx_control *Setup an approx_control object*

Description

Setup an approx_control object

Usage

```
setup_approx_control(FUN, ...)
```

Arguments

FUN	a function
...	arguments passed to FUN

Value

a approx_control object

Examples

```
# For usage, see example 7 in
## Not run: vignette("01_additional_examples", package = "geex")
```

setup_control *Setup a basic_control object*

Description

Setup a basic_control object

Usage

```
setup_control(type, FUN, ...)
```

Arguments

type	one of c("deriv","approx","root")
FUN	a function
...	arguments passed to FUN

Value

a basic_control object

See Also

setup_root_control, setup_deriv_control, setup_approx_control

---

`setup_deriv_control` *Setup a deriv_control object*

Description

Setup a deriv_control object

Usage

```
setup_deriv_control(FUN, ...)
```

Arguments

FUN	a function
...	arguments passed to FUN

Value

a deriv_control object

Examples

```
setup_deriv_control() # default
setup_deriv_control(method = "simple") # will speed up computations
```

`setup_root_control` *Setup a root_control object*

Description

Setup a root_control object

Usage

```
setup_root_control(FUN, roots_name, ...)
```

Arguments

FUN	a function
roots_name	a character string identifying the object containing the
...	arguments passed to FUN

Value

a `root_control` object

Examples

```
# Setup the default
setup_root_control(start = c(3, 5, 6))

# Also setup the default
setup_root_control(FUN = rootSolve::multiroot,
                   start = c(3, 5, 6))

# Or use uniroot()
setup_root_control(FUN = stats::uniroot,
                   interval = c(0, 1))
```

---

show — *Show (print) the S4 geex classes*

Description

`m_estimation_basis`, or `geex` object

Usage

```
show(object)

## S4 method for signature 'sandwich_components'
show(object)

## S4 method for signature 'm_estimation_basis'
show(object)

## S4 method for signature 'geex'
show(object)

## S4 method for signature 'geex_summary'
show(object)
```

Arguments

object	the object to print

summary,geex-method *Object Summaries*

Description

Object Summaries

Usage

```
## S4 method for signature 'geex'
summary(object, keep_data = TRUE, keep_args = TRUE)
```

Arguments

object	a geex object
keep_data	keep the original data or not
keep_args	keep the outer_args and inner_args passed to estFUN or not

Examples

```
## Not run:
library(geepack)
data('ohio')
glmfit  <- glm(resp ~ age, data = ohio,
              family = binomial(link = "logit"))
example_ee <- function(data, model){
  f <- grab_psiFUN(model, data)
  function(theta){
    f(theta)
  }
}
z  <- m_estimate(
estFUN = example_ee,
data = ohio,
compute_roots = FALSE,
units = 'id',
roots = coef(glmfit),
outer_args = list(model = glmfit))

object.size(z)
object.size(summary(z))
object.size(summary(z, keep_data = FALSE))
object.size(summary(z, keep_data = FALSE, keep_args = FALSE))

## End(Not run)
```

vcov,geex-method *Gets the variance-covariance matrix from a geex object*

Description

Gets the variance-covariance matrix from a geex object

Usage

```
## S4 method for signature 'geex'
vcov(object)

## S4 method for signature 'geex_summary'
vcov(object)
```

Arguments

object	a geex object

Examples

```
ex_eeFUN <- function(data){
 function(theta){
   with(data,
    c(Y1 - theta[1],
     (Y1 - theta[1])^2 - theta[2] ))
}}

results <- m_estimate(
 estFUN = ex_eeFUN,
 data  = geexex,
 root_control = setup_root_control(start = c(1,1)))

vcov(results)
```

weights,geex-method *Extract Model weights*

Description

Extract Model weights

Usage

```
## S4 method for signature 'geex'
weights(object)

## S4 method for signature 'geex_summary'
weights(object)
```

Arguments

object	a geex object

Index