Package: flipscores (via r-universe)

November 12, 2026

Version 1.3.2

Date 2026-04-17

Title Robust Score Testing in GLMs, by Sign-Flip Contributions

Author Livio Finos [cre, aut] (<https://orcid.org/0000-0003-3181-8078>), Jelle J. Goeman [ctb], Jesse Hemerik [ctb], Riccardo De Santis [ctb], Andrea Panarotto [ctb], Angela Andreella [ctb], Anna Vesely [cbt], Filippo Gambarota [ctb]

Maintainer Livio Finos <livio.finos@unipd.it>

Description Provides robust tests for testing in GLMs, by sign-flipping score contributions. The tests are robust against overdispersion, heteroscedasticity and, in some cases, ignored nuisance variables. See Hemerik, Goeman and Finos (2020) <doi:10.1111/rssb.12369>, De Santis, Goeman, Hemerik, Davenport, Finos (2025) <doi:10.1080/01621459.2025.2491775> and De Santis, Goeman, Davenport, Hemerik, Finos (2025) <doi:10.1214/25-EJS2409>.

Depends R (>= 4.1.0)

Imports stats, MASS, methods

Suggests rmarkdown, knitr, testthat (>= 3.0.0)

VignetteBuilder knitr

Encoding UTF-8

License GPL-2

Config/testthat/edition 3

URL https://livioivil.github.io/flipscores/

Config/roxygen2/version 8.1.0

Repository https://livioivil.r-universe.dev

Date/Publication 2026-11-11 23:00:54 UTC

RemoteUrl https://github.com/livioivil/flipscores

RemoteRef HEAD

RemoteSha 5a6faade314f34892613a3cd4000f2d8433b9748

Contents

anova.flipscores . . . 2
as.data.frame.jfs . . . 3
as.jfs . . . 4
as.jfs.fs_contrasts . . . 4
c.jfs . . . 5
combine-methods . . . 5
compute_scores . . . 7
confint.flipscores . . . 8
flipscores . . . 10
flipscores-method . . . 13
flipscores_contrasts . . . 13
flipscores_lm . . . 14
gcor . . . 17
jfs-methods . . . 20
make_flips . . . 20
model.matrix.fs_contrasts . . . 21
print.jfs . . . 22
summary.jfs . . . 22
update.flipscores . . . 23
update.fs_contrasts . . . 23

Index **25**

`anova.flipscores` *anova.flipscores*

Description

This is the `anova` method for `flipscores` object. Remark: it performs type III deviance decomposition as in `car::Anova`.

Usage

```
## S3 method for class 'flipscores'
anova(object, model1 = NULL, score_type = NULL, n_flips = 5000, id = NULL, ...)
```

Arguments

`object`	`glm` (or `flipscores`) object with the model under the null hypothesis (i.e. the covariates, the nuisance parameters).
`model1`	a `glm` (or `flipscores`) or a `matrix` (or `vector`). If it is a `glm` object, it has the model under the alternative hypothesis. The variables in `model1` are the same variables in `object` plus one or more variables to be tested. Alternatively, if `model1` is a `matrix`, it contains the tested variables column-wise.

score_type	The type of score that is computed. It can be "orthogonalized", "effective" or "basic". Default is "orthogonalized". "effective" and "orthogonalized" take into account the nuisance estimation. The default is NULL, in this case the value is taken from object.
n_flips	The number of random flips of the score contributions. When n_flips is equal or larger than the maximum number of possible flips (i.e. n^2), all possible flips are performed. Default is 5000.
id	a vector identifying the clustered observations. If NULL (default) observations are assumed to be independent. NOTE: if object is a flipscores and model$flip_param_call$id is not NULL, this is considered in the inference.
...	other parameters allowed in stats::anova.

Examples

```
set.seed(1)
dt=data.frame(X=scale(rnorm(50)),
   Z=factor(rep(LETTERS[1:3],length.out=50)))
dt$Y=rpois(n=nrow(dt),lambda=exp(dt$X*(dt$Z=="C")))
mod0=flipscores(Y~Z+X,data=dt,family="poisson")
summary(mod0)
anova(mod0)

mod1=flipscores(Y~Z*X,data=dt,family="poisson")
summary(mod1)
anova(mod0,model1 = mod1)
```

---

as.data.frame.jfs *Coerce jfs to data frame*

Description

Coerce jfs to data frame

Usage

```
## S3 method for class 'jfs'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A jfs object.
row.names	Ignored.
optional	Ignored.
...	Additional arguments (currently unused).

Value

A data frame with all summary tables stacked, with a `Model` column prepended.

`as.jfs` *Coerce to jfs objects*

Description

Generic function to coerce objects to class 'jfs'.

Coerces an `fs_contrasts` object to class `jfs`.

Usage

```
as.jfs(x, ...)

## Default S3 method:
as.jfs(object, ...)
```

Arguments

`x`	an object of class `fs_contrasts`
`...`	further arguments (currently unused)

Value

an object of class `jfs`

`as.jfs.fs_contrasts` *Coerce to jfs objects from contrasts*

Description

Coerce contrast objects to class 'jfs'.

Usage

```
## S3 method for class 'fs_contrasts'
as.jfs(x, ...)
```

Arguments

`x`	An object (typically from contrast methods).
`...`	Additional arguments passed to methods.

c.jfs *Combine jfs objects*

Description

Combine jfs objects

Usage

```
## S3 method for class 'jfs'
c(...)
```

Arguments

...	Objects to combine. Supported objects are `jfs`, `flipscores`, `glm` and `fs_contrasts`.

Value

A `jfs` object.

combine-methods *Nonparametric combination of* `jfs` *objects*

Description

Methods for combining `jfs` objects.

`combine_tests` combines the tests derived from multiverse models.

`combine_contrasts` combines the tests derived from the contrasts of a factor variable to get a global test for the factor (i.e. categorical predictor). It has strong analogies with ANOVA test.

Usage

```
combine_tests(mods, comb_funct = "maxT", by = NULL, by_list=NULL, tail = 0)

combine_contrasts(mods, comb_funct = "Mahalanobis", tail = 0)
```

Arguments

`mods`	a `jfs` object.
`comb_funct`	combining function to be used. Several functions are implemented: "mean", "median", "Fisher", "Liptak", (equal to) "Stoufer", "Tippet", (equal to) "minp", "maxT", "Mahalanobis". Alternatively it can be a custom function that has a Tspace matrix as input. For `combine_tests` the default is `comb_funct="maxT"`, while for `combine_contrasts` the default is `comb_funct="Mahalanobis"`.
`by`	if `NULL` (default), it combines all test statistics. If a characters, it refers to the column's name of `summary_table` (and printed by something like `summary(mods)`). The elements with the same value will be combined. If `by` is a vector, the values are defined by row-wise concatenation of the values of the columns in `by`. The argument is inactive if `by_list` is not `NULL`.
`by_list`	NULL (default) or a list of vectors. For each vector of the list it combines test statistics with position given by the element of the vector. If the vectors in the list are characters, these refer to names(mods$Tspace).
`tail`	direction of the alternative hypothesis. It can be "two.sided" (or 0, the default), "less" (or -1) or "greater" (or +1).

Value

The function returns a `jfs`-object.

Examples

```
#First example
library(flipscores)
set.seed(123)

#Simulate data
n=20
D=data.frame(X=rnorm(n),Z1=rnorm(n),Z2=rnorm(n))
D$Y=D$Z1+D$X+rnorm(n)

# Run four glms abd combine it in a list
mod1=glm(Y~X+Z1+Z2,data=D)
mod2=glm(Y~X+poly(Z1,2)+Z2,data=D)
mod3=glm(Y~X+poly(Z1,2)+poly(Z2,2),data=D)
mod4=glm(Y~X+Z1+poly(Z2,2),data=D)
mods=list(mod1=mod1,mod2=mod2,mod3=mod3,mod4=mod4)

# Let us analyze the tests related to coefficient "X" and combine them
res=flipscores(mods,n_flips = 5000, seed = 1, tested_coeffs = "X")
summary(combine_tests(res))
# Second (continued) example
# flipscores jointly on all models and all coefficients
mod1=glm(Y~X+Z1+Z2,data=D)
mod2=glm(Y~X+poly(Z1,2)+Z2,data=D)
```

```
mod3=glm(Y~X+poly(Z1,2)+poly(Z2,2),data=D)
mod4=glm(Y~X+Z1+poly(Z2,2),data=D)
mods=list(mod1=mod1,mod2=mod2,mod3=mod3,mod4=mod4)
res=flipscores(mods,n_flips = 2000)
summary(combine_tests(res))
summary(combine_tests(res, by="model"))
summary(combine_tests(res, by="coefficient"))
res2=combine_contrasts(res)
summary(res2)
#custom combinations:
coeffs=c("(Intercept)","X","Z1","Z2")
coeffs_ids=lapply(coeffs,grep,res2$summary_table$coeff)
names(coeffs_ids)=coeffs
summary(combine_tests(res2,by_list =   coeffs_ids))
```

`compute_scores` *compute_scores*

Description

Computes Score contributions for variables in `model1` (or derived from) under the null model `model0`. Same usage as `anova.glm`. The parameter `id` is used too, if present in `model0` (with priority) or in `model1`.

Usage

```
compute_scores(model0, model1, score_type = "standardized", ..., id = NULL)
```

Arguments

`model0`	a `glm` object with the model under the null hypothesis (i.e. the covariates, the nuisance parameters).
`model1`	a `glm` or a `matrix` (or `vector`). If it is a `glm` object, it has the model under the alternative hypothesis. The variables in `model1` are the same variables in `model0` plus one or more variables to be tested. Alternatively, if `model1` is a `matrix`, it contains the tested variables column-wise.
`score_type`	The type of score that is computed. It is "orthogonalized", "effective" or "basic". "effective" and "orthogonalized" take into account the nuisance estimation.
`...`	other arguments.
`id`	Cluster identifiers, a column name, or a one-sided formula. The resolved vector is stored in `attr(scores, "id")`, in score-row order.

Author(s)

Jesse Hemerik, Riccardo De Santis, Jelle Goeman and Livio Finos

Examples

```
set.seed(1)
Z=rnorm(20)
X=Z+rnorm(20)
Y=rpois(n=20,lambda=exp(Z+X))
mod0=glm(Y~Z,family="poisson")
X=data.frame(X=X)
scr0=compute_scores(model0 = mod0, model1 = X)
head(scr0)
```

---

confint.flipscores *Confidence intervals for flipscores*

Description

This function allows to apply the `confint` method to `flipscores` objects.

Usage

```
## S3 method for class 'flipscores'
confint(
  object,
  parm = NULL,
  level = 0.95,
  type = c("equitailed", "symmetric"),
  score_type = c("standardized", "effective"),
  alternative = c("two.sided", "greater", "less"),
  amp_const = 20,
  flips = NULL,
  n_flips = NULL,
  ...
)
```

Arguments

`object`	a `flipscores`) model object.
`parm`	a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, they are recovered from the input model (`to_be_tested`).
`level`	the confidence level required.
`type`	the type of confidence interval to build, either "equitailed" or "symmetric". Default is "equitailed".
`score_type`	The type of score that is computed. It can be "standardized" or "effective". Default is inherited by the input model.
`alternative`	The direction of the test to invert, and corresponding interval with respect to the estimate. It can be "two.sided" (default), "greater", or "less".

amp_const	Argument related to the maximum initial span of the search interval. See details.
flips	the fixed flips to use in the interval search. If NULL (default), it tries to inherit it from the input model.
n_flips	If flips are not given or retrieved by the model, the number of random flips of the score contributions. Default inherits them from the input model.
...	additional arguments for methods.

Details

All families allowed in glm, as well as Negative Binomial, are supported, and inherited here by the input model object. See the main function flipscores for further information.

flipscores is based on randomly sign-flipped score contributions. Performing the test (and generating the confidence interval) different times might give slightly different results. Providing a flip matrix generated with make_flips is suggested to uniform the results.

to_be_tested works differently for factors and character variables: the corresponding dummy variables are to be tested separately. So, to test a numeric variable X, one could write to_be_tested = "X", but to test a factor Z with levels A, B and C one should write to_be_tested = c("ZB", "ZC") as they appear in the model summary.

The bisection search is initialized in a predetermined interval. The maximum width of this interval is given by an amplitude (depending on the size of the estimated parameter and its standard error) multiplied by amp_const. The default for amp_const is 20. If the confidence bound is not found in the initial interval, Inf or -Inf is returned.

#' @seealso flipscores, make_flips

Value

A matrix (or vector) with columns giving lower and upper confidence limits for each parameter. These will be labelled as (1-level)/2 and 1 - (1-level)/2 in

Examples

```
set.seed(1)
dt <- data.frame(X=scale(rnorm(50)),
   Z=factor(rep(LETTERS[1:3],length.out=50)))
dt$Y=rpois(n=nrow(dt),lambda=exp(dt$X*(dt$Z=="C")))
mod0 <- flipscores(Y~Z+X,data=dt,family="poisson")
summary(mod0)
confint(mod0, parm="X")

xx <- rnorm(20)
zz <- rnorm(20, 0.2 * xx)
yy <- rnorm(20, 1 + 2 * xx - zz)
flps <- make_flips(20, 1000)
mod1 <- flipscores(yy ~ xx + zz)
summary(mod1)
confint(mod1, flips = flps)
```

`flipscores` *Flip Scores Test*

Description

Extended version of `flipscores()` supporting joint testing when `formula` is a list of glm objects, a list of formulas, or a formula with a matrix response.

Usage

```
flipscores(
  formula,
  family = gaussian(),
  data = NULL,
  score_type = "standardized",
  n_flips = 5000,
  alternative = "two.sided",
  id = NULL,
  seed = NULL,
  to_be_tested = NULL,
  tested_coeffs = NULL,
  flips = NULL,
  precompute_flips = TRUE,
  ...
)
```

Arguments

`formula`	One of: 1. A `formula` object (standard use, returns `flipscores` object) 2. A list of fitted `glm` objects (returns `jfs` object) 3. A list of `formula` objects (returns `jfs` object) 4. A `formula` with matrix response e.g. `cbind(y1,y2) ~ x` (returns `jfs` object)
`family`	Error distribution for `glm`. Used in cases 1, 3, 4.
`data`	A data frame. Used in cases 1, 3, 4.
`score_type`	Type of score. One of `"standardized"` (default), `"orthogonalized"`, `"effective"`, `"basic"`.
`n_flips`	Number of random flips. Default `5000`.
`alternative`	Alternative hypothesis. Default `"two.sided"`.
`id`	Optional cluster identifiers, also used to align score rows across models. Use a vector with one value per score row, a column name, or a one-sided formula such as `~ids`. Column names and formulas are evaluated in `data`, or separately in each fitted model's original data for a list of

models. With `NULL`, rows are matched by row names. Repeated IDs are aligned by occurrence, with the maximum multiplicity across models, and all occurrences share the same random sign. A supplied vector is applied unchanged to every model and must match each model's number of score rows, after `subset` and missing-value removal. A length mismatch is an error; values are never recycled or truncated. For models with different fitted row counts, consider `id = ~ids` or `id = "ids"`, with an ID column present in every model's data.

`seed` Random seed for reproducibility.

`to_be_tested` Character vector or list of coefficient names to test. If a character vector, the same terms are intersected with each model's coefficients. If a list, each element applies to the corresponding model. If `NULL` all coefficients are tested.

`tested_coeffs` Alias for `to_be_tested` for compatibility.

`flips` Optional matrix of pre-computed flips. Columns may be named by observations or, when `id` is supplied, by cluster identifiers. All observations in the same cluster must receive the same sign. For joint tests with `id`, observation columns follow the common rows in `alignment$rows`; alternatively supply columns named by ID.

`precompute_flips`
Logical, default `TRUE`. If `FALSE`, GLM flips are generated on demand using the same observation or cluster mapping.

`...` Additional arguments passed to `glm()` or the internal flipscores engine.

Value

A `flipscores` object (case 1) or a `jfs` object (cases 2, 3, 4). Joint objects retain local score matrices in `objects`, with resolved identifiers in `attr(scores, "id")`, and an `alignment` list containing the common `rows` and local row-to-common-row `index` maps.

Examples

```
set.seed(1)
n <- 50
df <- data.frame(
  y1  = rnorm(n),
  y2  = rnorm(n),
  yb  = rbinom(n, 1, 0.5),
  y1p = rpois(n, lambda = 3),
  y2p = rpois(n, lambda = 5),
  x1  = rnorm(n),
  x2  = rnorm(n)
)

# Case 1: single formula
res=flipscores(y1 ~ x1 + x2, family = gaussian(), data = df)
summary(res)

# Case 2: list of glm objects
```

```
models <- list(
  glm(y1 ~ x1 + x2, data = df, family = gaussian()),
  glm(y2 ~ x1 + x2, data = df, family = gaussian()),
  glm(yb ~ x1 + x2, data = df, family = binomial())
)
res=flipscores(models)
summary(res)
res=flipscores(models, to_be_tested = "x1")
summary(res)

# Different dataset sizes: resolve IDs separately in each model's data
df$ids <- rep(seq_len(n / 2), each = 2)
df_short <- df[-c(2, 5, 9), ]
models_by_id <- list(
  full = glm(y1 ~ x1 + x2, data = df),
  shorter = glm(y2 ~ x1 + x2, data = df_short)
)
# A shared vector of length 50 cannot match the shorter model's 47 rows:
tryCatch(
  flipscores(models_by_id, id = df$ids, to_be_tested = "x1", n_flips = 99),
  error = function(e) message(conditionMessage(e))
)
# These forms support different row counts and cluster flips by the same IDs:
res_by_id <- flipscores(models_by_id, id = ~ids, to_be_tested = "x1",
                        n_flips = 99, seed = 1)
res_by_name <- flipscores(models_by_id, id = "ids", to_be_tested = "x1",
                          n_flips = 99, seed = 1)

# Case 3: list of formulas, same family
res=flipscores(
  list(y1 ~ x1 + x2, y2 ~ x1 + x2),
  family = gaussian(),
  data   = df
)
summary(res)

# Case 4: matrix response, gaussian
res=flipscores(cbind(y1, y2) ~ x1 + x2, family = gaussian(), data = df)
summary(res)

# Case 4: matrix response, poisson
res=flipscores(cbind(y1p, y2p) ~ x1 + x2, family = poisson(), data = df)
summary(res)

# more examples
n=20
DD=data.frame(X=rnorm(n),Z1=rnorm(n),Z2=rnorm(n))
DD$Y=DD$Z1+DD$X+rnorm(n)
# Run four glms abd combine it in a list
mod1=glm(Y~X+Z1+Z2,data=DD)
mod2=glm(Y~X+poly(Z1,2)+Z2,data=DD)
mod3=glm(Y~X+poly(Z1,2)+poly(Z2,2),data=DD)
mod4=glm(Y~X+Z1+poly(Z2,2),data=DD)
```

```
mods=list(mod1=mod1,mod2=mod2,mod3=mod3,mod4=mod4)
res=flipscores(mods, to_be_tested = "X")
summary(res)
```

`flipscores-method` *Methods for flipscores objects*

Description

Methods for `flipscores` objects. The following are methods to extract and manipulate relevant information from a `flipscores` object.

Usage

```
## S3 method for class 'flipscores'
print(x, ...)

## S3 method for class 'flipscores'
summary(object, ...)
```

Arguments

`x`	a flipscores object
`...`	additional arguments to be passed
`object`	a flipscores object

`flipscores_contrasts` *Flip-score tests for custom contrasts*

Description

Defines factor contrasts from a formula or accepts fully custom coefficient contrasts, then applies the flip-score test to each contrast.

Usage

```
flipscores_contrasts(
  model,
  specs = NULL,
  linfct = NULL,
  ref = NULL,
  score_type = "standardized",
  n_flips = 5000,
  alternative = "two.sided",
  id = NULL,
```

```
  seed = NULL,
  flips = NULL,
  ...
)
```

Arguments

`model`	a fitted `glm`, `lm`, or `flipscores` object.
`specs`	a contrast formula, coefficient contrast matrix, numeric vector, named list of numeric vectors, or `NULL` when `linfct` is supplied.
`linfct`	optional custom coefficient contrast matrix, numeric vector, or named list of numeric vectors. This follows the same idea as `multcomp::glht()`: rows define linear functions of the model coefficients.
`ref`	control level for Dunnett-like formula contrasts. It can be a level name, cell name, or numeric index. The default is the first level/cell, except for `trt.vs.ctrlk`, where the default is the last level/cell.
`score_type`	type of score, as in `flipscores()`.
`n_flips`	number of sign flips.
`alternative`	`"two.sided"`, `"greater"`, or `"less"`.
`id`	optional cluster id.
`seed`	optional random seed.
`flips`	optional precomputed flip matrix.
`...`	currently unused.

Value

an object of class `fs_contrasts`.

`flipscores_lm` *Robust score testing for linear models via sign-flipping*

Description

Performs robust score tests for one or more coefficients in a linear model (ordinary least squares) by sign-flipping score contributions. The test is valid under heteroscedasticity, overdispersion and, in some cases, ignored nuisance variables. It is an efficient alternative to `flipscores()` when the model is Gaussian and the link is identity, the `score_type` implemented is the `"standardized"`.

Usage

```
flipscores_lm(
  formula,
  data = NULL,
  n_flips = 5000,
  alternative = "two.sided",
  seed = NULL,
  tested_coeffs = NULL,
  flips = NULL,
  ...
)
```

Arguments

`formula`	A formula describing the linear model (e.g. `y ~ x1 + x2`). The response can be a numeric vector or a matrix.
`data`	An optional data frame containing the variables in the model. If `NULL`, variables are taken from the environment of the formula.
`n_flips`	Number of random sign flips. Defaults to `5000`. When `n_flips >= n^2` (with n the number of observations) all possible flips are performed, giving an exact permutation test.
`alternative`	Alternative hypothesis. Must be one of `"two.sided"` (default), `"greater"` or `"less"`.
`seed`	Optional integer seed for reproducibility of the random flips.
`tested_coeffs`	Character vector naming the coefficients to test. If `NULL` (default), all coefficients in the model are tested.
`flips`	Optional matrix of pre-computed sign flips. It must have `n_flips` rows and n columns, where each entry is $+1$ or -1. If supplied, `n_flips` is ignored.
`...`	Additional arguments passed to internal functions (currently not used).

Details

The function works by first projecting the response and the tested covariate onto the orthogonal complement of the nuisance design matrix. The resulting score contributions are then randomly sign-flipped (with fixed signs across observations) to generate a permutation distribution of the test statistic.

The observed score statistic for a coefficient is $X_r^\top Y_r / \sqrt{\sum X_r^2}$, where X_r and Y_r are the residualised covariate and residualised response, respectively. Under the null hypothesis the sign of each observation's contribution is arbitrary, so repeatedly flipping the signs yields an exact conditional test.

The method is robust to misspecification of the variance structure and does not require the errors to be normally distributed or homoscedastic.

Value

An object of class `c("fs_lm", ...)` with the following components:

`Tspace` A matrix of dimension `n_flips` by `k`, where `k = length(tested_coeffs)`. The first row contains the observed test statistics; the remaining rows contain the test statistics obtained after random sign flips.

`summary_table` A data frame with one row per tested coefficient, containing the observed score, the two-sided or one-sided p-value, and optionally other summary statistics (e.g. standardised score).

`info` A list with the original formula, the names of the design matrix columns, and the names of the response variables.

`call` The matched call.

References

Hemerik, J., Goeman, J. J., & Finos, L. (2020). Robust testing in generalized linear models by sign-flipping score contributions. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 82(3), 841-864. doi:10.1111/rssb.12369

De Santis, R., Goeman, J. J., Hemerik, J., Davenport, S., & Finos, L. (2025). Inference in Generalized Linear Models with Robustness to Misspecified Variances. *Journal of the American Statistical Association*, 1-10. doi:10.1080/01621459.2025.2491775

See Also

`flipscores` for the general GLM version; `summary.fs_lm` for summarising results; `anova.fs_lm` for joint tests across coefficients.

Examples

```
set.seed(123)
n <- 100
data <- data.frame(
  x1 = rnorm(n),
  x2 = rnorm(n),
  y  = 2 * rnorm(n, mean = 0, sd = 1)   # pure noise (null holds)
)
data$y2  = data$x1+  rnorm(n, mean = 0, sd = 1)   # pure noise (null holds)

# Test the coefficient of x1
res <- flipscores_lm(cbind(y,y2) ~ x1 + x2, data = data, n_flips = 999, alternative = "two.sided")
flipscores:::summary.fs_lm(res)

# Test only x2
res2 <- flipscores_lm(y ~ x1 + x2, data = data, tested_coeffs = "x2")
flipscores:::summary.fs_lm(res2)
```

`gcor` *Compute Generalized Partial Correlations for GLM terms*

Description

This function computes the generalized partial correlation coefficient r for each specified variable in a generalized linear model. For each variable, it refits the null model excluding that variable and computes the cosine similarity between the residualized predictor and standardized residuals.

Usage

```
gcor(full_glm, terms = NULL, normalize = FALSE, intercept_too = FALSE)
```

Arguments

`full_glm`	A fitted GLM object of class 'glm'.
`terms`	Character vector of variable names (referred to the model.matrix, that is pay attention for factors) for which to compute generalized partial correlations. If 'NULL' (default), computes for all non-intercept terms in the model.
`normalize`	FALSE by default.
`intercept_too`	Logical indicating whether to include the intercept as a variable. Default is FALSE.

Details

The generalized partial correlation° measures the association between a predictor and response after adjusting for all other terms in the model. It is defined as the cosine similarity between the residualized predictor X_r and standardized residuals Y_r:

$$= \frac{X_r^\top Y_r}{\|X_r\|\|Y_r\|}$$

where:

- $X_r = (I - H)W^{1/2}X$ is the residualized predictor
- $Y_r = V^{-1/2}(Y - \hat{\mu})$ is the standardized residual vector
- H is the hat matrix for the nuisance covariates
- $W = DV^{-1}D$ is the weight matrix
- V is the variance matrix and D is the derivative matrix

The function uses 'flipscores:::get_par_expo_fam()' to compute V and D consistently with the flipscores package methodology.

The normalized generalized partial correlation is computed as:

$$r_n = \begin{cases} +r/r_+ & \text{if } r > 0 \\ -r/r_- & \text{if } r < 0 \end{cases}$$

where r_+ is the maximum possible correlation and r_- is the minimum.

When the (full) model has only intercept and only one predictor X, the generalized (non partial) correlation is computed and the normalization factor for X is exact. In the more general case with more predictors, for sample sizes $n \leq n_{\text{exact}}$, brute force search is used to find the exact extrema. For larger sample sizes, a greedy multi-start algorithm is employed:

- Multiple starting points are generated using thresholding and random sampling
- From each start, coordinates are greedily flipped to improve the correlation
- Early stopping is used when no improvements are found for several iterations
- The best solution across all starts is returned

This approach provides a good trade-off between computational efficiency and solution quality for large problems where brute force is infeasible.

Value

if `normalize` is `FALSE`, a data frame with five columns:

`variable`	The variable name
`r`	The generalized partial correlation coefficient

while if `normalize` is `TRUE`

`terms`	The variable name
`r`	The generalized partial correlation coefficient
`r_n`	The normalized generalized partial correlation coefficient
`null_model`	The null model used to compute the generalized (partial) correlation
`algorithm`	The algorithm used to compute the upper/lower bounds of the generalized partial correlation coefficient (to compute its normalized version)
`exact`	logical

Author(s)

Livio Finos and Paolo Girardi

Examples

```
set.seed(1)
dt=data.frame(X=rnorm(20),
   Z=factor(rep(LETTERS[1:3],length.out=20)))
dt$Y=rpois(n=20,lambda=exp(dt$Z=="C"))
mod=flipscores(Y~Z+X,data=dt,family="poisson",n_flips=1000)
summary(mod)
```

```
# Compute generalized partial correlations for all terms
(results <- gcor(mod))

# Compute for specific terms only
gcor(mod, terms = c("X", "ZC"))

gcor(mod, terms = c("X", "ZC"),normalize=TRUE)


gcor(mod, intercept_too=TRUE, normalize=TRUE)
set.seed(123)
dt=data.frame(X=rnorm(20),
   Z=factor(rep(LETTERS[1:3],length.out=20)))
dt$Y=rbinom(n=20,prob=plogis((dt$Z=="C")*2),size=1)
mod=flipscores(Y~Z+X,data=dt,family="binomial",n_flips=1000)
summary(mod)

(results <- gcor(mod,normalize=TRUE))
# Compute for specific terms only
gcor(mod, terms = c("X", "ZC"),normalize=TRUE)


############################
set.seed(1)
Z=rnorm(20)
X=Z+rnorm(20)-10
Y=rpois(n=20,lambda=exp(Z+.25*X))
model1=glm(Y~Z+X,family="poisson")
gcor(model1)
gcor(model1, normalize=TRUE)
#normalized_gcor_poisson(model1, "X", verbose = TRUE)[1:3]
model0=glm(Y~Z,family="poisson")
## Not run: sapply(c('v','kl','sse','lr','n'),function(type) rsq.partial(model1,model0,type=type)$partial

set.seed(1)
Z=rnorm(20)
X=Z+rnorm(20)
Y=rbinom(n=20,1,prob=plogis((-.5*Z+1*X)))
model1=glm(Y~Z+X,family="binomial")
gcor(model1)
gcor(model1, normalize=TRUE)
#normalized_gcor_binomial(model1, "X", verbose = TRUE)[1:3]
model0=glm(Y~Z,family="binomial")
## Not run: sapply(c('v','kl','sse','lr','n'),function(type) rsq.partial(model1,model0,type=type)$partial

set.seed(1)
Z=rnorm(20)
X=10+Z+rnorm(20)
Y=rnorm(n=20,mean=(Z+2*X))
model1=glm(Y~Z+X)
gcor(model1)
gcor(model1, normalize=TRUE)
```

```
#library(rsq)
model0x=glm(X~Z)
model0y=glm(Y~Z)
#t(.nrmz(resid(model0x)))%*% .nrmz(resid(model0y))
#library(ppcor)
#pcor(cbind(X,Z,Y))
## Not run: sapply(c('v','kl','sse','lr','n'),function(type) rsq.partial(model1,model0,type=type)$partial
```

`jfs-methods` *Methods for jfs objects*

Description

Collection of methods for objects of class `jfs`.

`p.adjust` method for class "`jfs`" and "`flipscores`". Add adjusted p-values into the `jfs`and `flipscores` object.

Usage

```
p.adjust(object, method = "maxT", tail = 0, ...)
```

Arguments

`object`	an object of class `jfs` and `flipscores`.
`method`	any method implemented in `flip.adjust` or a custom function. In the last case it must be a function that uses a matrix as input and returns a vector of adjusted p-values equal to the number of columns of the inputed matrix.
`tail`	argument: expresses the tail direction of the alternative hypothesis. It can be "two.sided" (or 0, the default), "less" (or -1) or "greater" (or +1).
`...`	additional arguments to be passed

`make_flips` *It creates a* `n_flips`*x*`n_obs` *matrix of random +1 and -1. The first row is made by ones (i.e. the observed test statistic is computed)*

Description

It creates a `n_flips`x`n_obs` matrix of random +1 and -1. The first row is made by ones (i.e. the observed test statistic is computed)

Usage

```
make_flips(n_obs, n_flips, obs_names = NULL, id = NULL)
```

Arguments

`n_obs`	number of observations
`n_flips`	number of flips
`obs_names`	optional observation names for the columns.
`id`	optional cluster identifiers, one per observation.

Examples

```
# example code
make_flips(n_obs=10,n_flips=20)
```

`model.matrix.fs_contrasts` *Model matrix for fs_contrasts objects*

Description

Extract the model matrix from the underlying fitted model stored in an ‘fs_contrasts‘ object.

Usage

```
## S3 method for class 'fs_contrasts'
model.matrix(object, ...)
```

Arguments

`object`	An object of class ‘fs_contrasts‘.
`...`	Additional arguments passed to ‘model.matrix‘ (e.g., ‘data‘).

Value

The model matrix (design matrix) of the original model.

`print.jfs` *Print method for jfs*

Description

Print method for jfs

Usage

```
## S3 method for class 'jfs'
print(x, n = 2, ...)
```

Arguments

`x`	A `jfs` object.
`n`	Number of rows to show at head and tail of summary table.
`...`	Additional arguments (currently unused).

`summary.jfs` *Summary method for jfs*

Description

Summary method for jfs

Usage

```
## S3 method for class 'jfs'
summary(object, digits = 4, ...)
```

Arguments

`object`	A `jfs` object.
`digits`	Number of digits to print. Default `4`.
`...`	Additional arguments (currently unused).

update.flipscores *Update method for flipscores*

Description

Ensures that `update()` on a `flipscores` object returns a proper `flipscores` object rather than a plain `glm`.

Usage

```
## S3 method for class 'flipscores'
update(object, ...)
```

Arguments

`object`	A `flipscores` object.
`...`	Additional arguments to update in the call, such as `formula`, `data`, `family`, `score_type`, etc. Supports the `.` shorthand in formula updates, e.g. `formula = . ~ . + offset(.OFFSET___)`.

Value

A `flipscores` object.

update.fs_contrasts *Convert an fs_contrasts object to a jfs object*

Description

This method creates a 'jfs' object from a fitted model and a set of contrasts. The resulting object can be used with 'flipscores' for joint permutation testing across multiple contrasts or models.

Usage

```
## S3 method for class 'fs_contrasts'
update(object, formula., ..., evaluate = TRUE)
```

Arguments

`object`	An object of class 'fs_contrasts'.
`formula.`	Changes to the formula – see [update.formula()] for details.
`...`	Additional arguments passed to 'update()' for the underlying model, or to the contrast-generating function.
`data`	New data frame.
`family`	New family for the GLM.
`linfct`	New contrast matrix (or formula to generate it).

Value

An object of class ‘jfs‘ (joint flipscores) that contains the original model, the contrast matrix, and other relevant information.

An updated object of class ‘fs_contrasts‘.

Index

anova.flipscores, 2
anova.fs_lm, *16*
as.data.frame.jfs, 3
as.jfs, 4
as.jfs.fs_contrasts, 4

c.jfs, 5
combine-methods, 5
combine_contrasts *(combine-methods)*, 5
combine_tests *(combine-methods)*, 5
compute_scores, 7
confint.flipscores, 8

flipscores, *9*, 10, *16*
flipscores-method, 13
flipscores_contrasts, 13
flipscores_lm, 14

gcor, 17

jfs-methods, 20

make_flips, *9*, 20
model.matrix.fs_contrasts, 21

p.adjust *(jfs-methods)*, 20
print.flipscores *(flipscores-method)*, 13
print.jfs, 22

summary.flipscores *(flipscores-method)*, 13
summary.fs_lm, *16*
summary.jfs, 22

update.flipscores, 23
update.fs_contrasts, 23