Package 'contrastable'

October 1, 2024

Type Package

Title Consistent Contrast Coding for Factors

Version 1.0.2

Description Quickly set and summarize contrasts for factors prior to regression analyses. Intended comparisons, baseline conditions, and intercepts can be explicitly set and documented without the user needing to directly manipulate matrices. Reviews and introductions for contrast coding are available in Brehm and Alday (2022)<doi:10.1016/j.jml.2022.104334> and Schad et al. (2020)<doi:10.1016/j.jml.2019.104038>.

License MIT + file LICENSE

Encoding UTF-8

Imports cli, crayon, glue, MASS, purrr, rlang (>= 0.1.2), stats, tidyselect

RoxygenNote 7.3.2

Suggests dplyr, knitr, testthat (>= 3.0.0), roxygen2, rmarkdown, covr, hypr

Config/testthat/edition 3

URL https://github.com/tsostarics/contrastable

BugReports https://github.com/tsostarics/contrastable/issues

VignetteBuilder knitr

NeedsCompilation no

Author Thomas Sostarics [aut, cre, cph] (<https://orcid.org/0000-0002-1178-7967>)

Maintainer Thomas Sostarics <tsostarics@gmail.com>

Repository CRAN

Date/Publication 2024-10-01 14:50:03 UTC

Contents

`.add_namespace` *Lookup namespace of contrast scheme function*

Description

Given the name of a contrast scheme (ie the function name that creates the contrast matrix), lookup which namespace it belongs to and add it to the string. Used to report which packages are used in the glimpse table, e.g., stats, contrastable, bayesTestR

Usage

```
.add_namespace(scheme_names)
```

Arguments

scheme_names	Character vector

Value

character vector of updated function names with namespaces

.get_from_params *Retrieve value from param list*

Description

Helper to evaluate param entries.

Usage

```
.get_from_params(what, list_params, formulas)
```

Arguments

what	Which parameter to retrieve
list_params	List of params, see .make_parameters()
formulas	Formulas used to set contrasts

Value

Requested value for each parameter as a string

.warn_if_nondefault *Warn user if nondefault contrasts are set*

Description

R automatically assigns specific contrast schemes to ordered and unordered factors as specified in options('contrasts') but users are of course free to set these on the factor themselves. But, if they do this outside of a call to glimpse_contrasts it's hard and time consuming to check what they set against the different possible common schemes. So, rather than checking all possible combinations, this will only check against the defaults R already uses and alert the user if something else is set.

Usage

```
.warn_if_nondefault(
  contrast_list,
  factor_names,
  factor_sizes,
  which_ordered,
  schemes_to_use
)
```

Arguments

contrast_list	List of contrasts like that generated by enlist_contrasts
factor_names	Names of the factors, also the names of the contrast list
factor_sizes	Number of levels for each factor
which_ordered	Which of the factors are ordered
schemes_to_use	Character vector of schemes, if any don't match the default for a factor, it will be replaced wtih ??? in th eoutput

Value

Warns if non default contrasts are set, returns schemes_to_use with modifications if necessary

---

as.unordered *Convert to unordered factor*

Description

Unordered analogue of base R's as.ordered. Will convert x to an unordered factor; unlike as.factor(), this will convert ordered factors to unordered factors.

Usage

```
as.unordered(x)
```

Arguments

x	Object to convert to unordered factor

Value

x as an unordered factor

See Also

as.factor()

Examples

```
# Convert an ordered factor to unordered
as.unordered(gl(5,1,ordered = TRUE))

# If level order is pre-specified differently from default alphabetical order
# then the ordering will be retained
as.unordered(ordered(c("a", "b", "c"), levels = c("c", "a", "b")))

# Otherwise the vector will be converted to an unordered factor with levels
# in the default alphabetical order
as.unordered(c("c", "a", "b"))

# Note that coercing integer values will sort the values to use as the levels
as.unordered(4:1)
```

---

backward_difference_code

Backward difference code

Description

Compares the mean of level k to level k-1. Differs in direction from forward_difference_code, so be careful to pick the right function. See also contr.sdif.

Usage

```
backward_difference_code(n)
```

Arguments

n	Integer umber of factor levels to compute contrasts for.

Details

Example interpretation for a 4 level factor:

- Intercept = Grand mean (mean of the means of each level)
- grp1 = mean(grp2) - mean(grp1)
- grp2 = mean(grp3) - mean(grp2)
- grp3 = mean(grp4) - mean(grp3)

Value

A contrast matrix with dimensions n rows and (n-1) columns.

Examples

```
mydf <- data.frame(
  grp = gl(4,5),
  resp = c(seq(1, 5), seq(5, 9), seq(10, 14), seq(15, 19))
)

mydf <- set_contrasts(mydf, grp ~ backward_difference_code)

lm(resp ~ grp, data = mydf)
```

---

cumulative_split_code *Cumulative split contrasts*

Description

Contrast coding scheme that repeatedly dichotomizes the factor levels.

Usage

```
cumulative_split_code(n)
```

Arguments

n	Integer umber of factor levels to compute contrasts for.

Details

This scheme is similar to Helmert contrasts, but instead of comparing one level to the accumulated mean of all previous levels, each comparison with this scheme splits the levels into two groups: those below and including the current level, and those above the current level. Conceptually this is similar to continuation ratio logits used in ordinal models. For example, with a four level factor with levels A, B, C, and D, the comparisons would be:

- A vs. BCD
- AB vs. CD
- ABC vs. D

In other words, each comparison splits the levels into two groups. Each of these comparisons uses the cumulative mean of all the levels in each group. The intercept is the grand mean.

Value

A contrast matrix with dimensions n rows and (n-1) columns.

Examples

```
set.seed(111)
mydf <- data.frame(
  grp = rep(c("a", "b", "c", "d"), each = 2000),
  val = c(
    rnorm(200, 2, 1),
    rnorm(200, 5, 1),
    rnorm(200, 7.5, 1),
    rnorm(200, 15, 1)
  )
) |>
  set_contrasts(grp ~ cumulative_split_code |
    c("a-rest", "ab-rest", "abc-rest"))

lm(val ~ grp, data = mydf)
```

`decompose_contrasts` *Decompose contrasts into new columns*

Description

Given a dataframe with factor columns, this function will extract the contrasts from the factor column and place them inside new columns. This is useful for cases where you want to work with the numeric values of the contrasts. For a pedagogical example, you can explicitly show how factor variables are transformed into numeric values. For a practical example, you're typically allowed n-1 contrasts for n levels of a factor. If you don't want to use all of the contrasts, you can extract the ones you want and use them in your model. This is sometimes used with polynomial contrasts when you don't want to use higher order polynomials.

Usage

```
decompose_contrasts(
  model_data,
  extract,
  remove_intercept = TRUE,
  remove_original = FALSE
)
```

Arguments

`model_data`	Dataframe with factor columns
`extract`	A one-sided formula denoting the factors to extract. Note this should ideally be what you would pass to your model fitting function, sans any non-factors.
`remove_intercept`	Logical, whether to remove the column corresponding to the intercept. Default `TRUE` since it's always just a column of 1s

`remove_original`
: Logical, whether to remove the original columns in the data frame after decomposing into separate columns. Default `FALSE`.

Details

An additional usage for this function is to compute the contrasts for interaction terms in a model. In `lm(y ~ A * B)`, where A and B are factors, the expanded form is `lm(y ~ A + B + A:B)` with an equation of $y = \beta_A x_A + \beta_B x_B + \beta_{A:B} x_A x_B$. The thing to note is that the coefficient for the interaction(s) are multiplied by the product of x_A and x_B. Let's call this product x_C. For example, if one value of x_A is -1/3 and one value of x_B is 2/3, then the product x_C is -2/9. But, if there are 3 levels for x_A and 3 levels for x_B, then we get 4 columns for the fixed effects *and* 4 *more* columns for the interaction terms. It can be a lot of tedious work to precompute the products manually, so we can use this function with `extract_interaction = TRUE` to compute everything at once.

Value

`model_data` but with new columns corresponding to the numeric coding of the given factor's contrasts

See Also

`set_contrasts()`

Examples

```
# Decompose contrasts for carb and gear columns into new columns, using
# the contrast labels used when setting the contrasts
mtcars |>
  set_contrasts(
    carb ~ scaled_sum_code,
    gear ~ contr.sum | c("4-mean", "5-mean")
  ) |>
  decompose_contrasts(~ carb + gear) |>
  str()

# Decompose an interaction term between the two factors
mtcars |>
  set_contrasts(
    carb ~ scaled_sum_code,
    gear ~ contr.sum | c("4-mean", "5-mean")
  ) |>
  decompose_contrasts(~ carb * gear) |>
  str()
```

`enlist_contrasts` *List of contrast matrices*

Description

Returns a named list of contrast matrices to use with modeling functions directly. See `set_contrasts()` for a function to set contrasts directly to the dataframe. See details for syntax information

Usage

```
enlist_contrasts(model_data, ..., verbose = getOption("contrastable.verbose"))
```

Arguments

`model_data`	Data frame you intend on passing to your model
`...`	A series of 2 sided formulas with factor name on the left hand side and desired contrast scheme on the right hand side. The reference level can be set with `+`, the intercept can be overwritten with `*`, comparison labels can be set using `|`, and trends for polynomial coding can be removed using `-`.
`verbose`	Logical, defaults to FALSE, whether messages should be printed

Details

`enlist_contrasts()`, `set_contrasts()`, and `glimpse_contrasts()` use special syntax to set contrasts for multiple factors. The syntax consists of two-sided formulas with the desired factor column on the left hand side and the contrast specification on the right hand side. For example, `varname ~ scaled_sum_code`. Many contrasts support additional kinds of contrast manipulations using overloaded operators:

- `+ X`: Set the reference level to the level named X. Only supported for schemes that have a singular reference level such as `sum_code()`, `scaled_sum_code()`, `treatment_code()`, `stats::contr.treatment()`, `stats::contr.sum()`, `stats::contr.SAS()`. Ignored for schemes like `helmert_code()`.
- `* X`: Overwrite the intercept to the mean of the level named X
- `- A:B`: For polynomial coding schemes only, drop comparisons A through B.
- `| c(...)`: Change the comparison labels for the contrast matrix to the character vector `c(...)` of length `n-1`. These labels will appear in the output/summary of a statistical model. Note that for `brms::brm`, instances of `-` (a minus sign) are replaced with `M`.

You can also specify multiple variables on the left hand side of a formula using tidyselect helpers. See examples for more information.

Typically model functions like lm will have a contrasts argument where you can set the contrasts at model run time, rather than having to manually change the contrasts on the underlying factor columns in your data. This function will return such a named list of contrast matrices to pass to these functions. Note that this function should not be used within a modeling function call, e.g., `lm(y~x, data = model_data, contrasts = enlist_contrasts(model_data, x~sum_code))`. Often, this will call `enlist_contrasts` twice, rather than just once.

For some model fitting functions, like `brms::brm`, there is no contrasts argument. For such cases, use `set_contrasts()` to set contrasts directly to the factors in a dataframe.

One good way to use `enlist_contrasts()` is in conjunction with `MASS::fractions()` to create a list of matrices that can be printed to explicitly show the entire contrast matrices you're using for your models. This can be especially helpful for supplementary materials in an academic paper.

Sometimes when using orthogonal polynomial contrasts from `stats::contr.poly()` people will drop higher level polynomials for parsimony. Note however that these do capture some amount of variation, so even though they're orthogonal contrasts the lower level polynomials will have their estimates changed. Moreover, you cannot reduce a contrast matrix to a matrix smaller than size n*n-1 in the dataframe you pass to a model fitting function itself, as R will try to fill in the gaps with something else. If you want to drop contrasts you'll need to use something like `enlist_contrasts(df, x ~ contr.poly - 3:5)` and pass this to the `contrasts` argument in the model fitting function.

Value

List of named contrast matrices. Internally, if called within set_contrasts, will return a named list with `contrasts` equal to the list of named contrast matrices and `data` equal to the passed `model_data` with any factor coercions applied (so that `set_contrasts()` doesn't need to do it a second time).

See Also

`set_contrasts()` `glimpse_contrasts()`

Examples

```
my_df <- mtcars
my_df$gear <- factor(my_df$gear)
my_df$carb <- factor(my_df$carb)

# Use formulas where left hand side is the factor column name
# and the right hand side is the contrast scheme you want to use
enlist_contrasts(
  my_df,
  gear ~ scaled_sum_code,
  carb ~ helmert_code,
  verbose = FALSE
)

# Add reference levels with +
enlist_contrasts(
  my_df,
  gear ~ scaled_sum_code + 5,
  carb ~ contr.sum + 6,
  verbose = FALSE
)
# Manually specifying matrix also works
enlist_contrasts(
  my_df,
  gear ~ matrix(c(1, -1, 0, 0, -1, 1), nrow = 3),
  carb ~ forward_difference_code,
```

```
  verbose = FALSE
)

# User matrices can be assigned to a variable first, but this may make the
# comparison labels confusing. You should rename them manually to something
# that makes sense. This will invoke use_contrast_matrix, so reference levels
# specified with + will be ignored.
my_gear_contrasts <- matrix(c(1, -1, 0, 0, -1, 1), nrow = 3)
colnames(my_gear_contrasts) <- c("CMP1", "CMP2")
enlist_contrasts(
  my_df,
  gear ~ my_gear_contrasts,
  carb ~ forward_difference_code,
  verbose = FALSE
)


# Will inform you if there are factors you didn't set
enlist_contrasts(my_df, gear ~ scaled_sum_code)

# Use MASS::fractions to pretty print matrices for academic papers:
lapply(enlist_contrasts(my_df, gear ~ scaled_sum_code, carb ~ helmert_code),
       MASS::fractions)

# Use a list of formulas to use the same contrasts with different datasets
my_contrasts <- list(gear ~ scaled_sum_code, carb ~ helmert_code)
enlist_contrasts(my_df,  my_contrasts)
enlist_contrasts(mtcars, my_contrasts)

# Use tidyselect helpers to set multiple variables at once
# These are all equivalent
contr_list1 <- enlist_contrasts(mtcars,
                cyl ~ sum_code, gear ~ sum_code,
                verbose = FALSE)

contr_list2 <- enlist_contrasts(mtcars,
                cyl + gear ~ sum_code,
                verbose = FALSE)

contr_list3 <- enlist_contrasts(mtcars,
                c(cyl, gear) ~ sum_code,
                verbose = FALSE)

contr_list4 <- enlist_contrasts(mtcars,
                all_of(c('cyl', 'gear')) ~ sum_code,
                verbose = FALSE)


these_vars <- c("cyl", "gear")
contr_list5 <- enlist_contrasts(mtcars,
                                all_of(these_vars) ~ sum_code,
                                verbose = FALSE)
```

```
all.equal(contr_list1, contr_list2)
all.equal(contr_list2, contr_list3)
all.equal(contr_list3, contr_list4)
all.equal(contr_list4, contr_list5)

# You can also use [tidyselect::where()] with class checking helpers:
contr_list6 <- enlist_contrasts(mtcars,
                                where(is.numeric) ~ sum_code,
                                verbose = FALSE)

# Each variable name must only be set ONCE, e.g. these will fail:
try(enlist_contrasts(mtcars,
                     cyl ~ sum_code,
                     cyl ~ scaled_sum_code,
                     verbose = FALSE))
try(enlist_contrasts(mtcars,
                     cyl ~ sum_code,
                     all_of(these_vars) ~ scaled_sum_code,
                     verbose = FALSE))
try(enlist_contrasts(mtcars,
                     cyl ~ sum_code,
                     where(is.numeric) ~ scaled_sum_code,
                     verbose = FALSE))
```

`forward_difference_code` *Forward difference code*

Description

Compares the mean of level k to level k+1. Differs in direction from backward_difference_code, so be careful to pick the right function. See also contr.sdif.

Usage

```
forward_difference_code(n)
```

Arguments

n	Integer umber of factor levels to compute contrasts for.

Details

Example interpretation for a 4 level factor:

- Intercept = Grand mean (mean of the means of each level)
- grp1 = mean(grp1) - mean(grp2)
- grp2 = mean(grp2) - mean(grp3)
- grp3 = mean(grp3) - mean(grp4)

Value

A contrast matrix with dimensions n rows and (n-1) columns.

Examples

```
mydf <- data.frame(
  grp = gl(4,5),
  resp = c(seq(1, 5), seq(5, 9), seq(10, 14), seq(15, 19))
)

mydf <- set_contrasts(mydf, grp ~ forward_difference_code)

lm(resp ~ grp, data = mydf)
```

---

glimpse_contrasts *Glimpse contrasts in dataframe*

Description

Uses the same syntax as `enlist_contrasts()` and `set_contrasts()`. Returns a summary table of the contrasts you've set. If you set `return.list=TRUE` then you can access a list of contrasts in the second element of the resulting list. The glimpse dataframe is the first element. `FALSE` will return just the glimpse data frame.

Usage

```
glimpse_contrasts(
  model_data,
  ...,
  return_list = FALSE,
  show_all_factors = TRUE,
  add_namespace = FALSE,
  show_one_level_factors = FALSE,
  minimal = TRUE,
  verbose = getOption("contrastable.verbose")
)
```

Arguments

`model_data`	Data to be passed to a model fitting function
`...`	Series of formulas
`return_list`	Logical, defaults to FALSE, whether the output of enlist_contrasts should be returned
`show_all_factors`	Logical, defaults to TRUE, whether the factors not explicitly set with formulas should be included

add_namespace	Logical, defaults to FALSE, whether to append the namespace of the contrast scheme to the scheme name
show_one_level_factors	Logical, should factors with only one level be included in the output? Default is FALSE to omit
minimal	Logical, default TRUE, whether to omit the orthogonal, centered, dropped_trends, and explicitly_set columns from the output table
verbose	Logical, defaults to TRUE, whether messages should be printed

Details

Generally, `glimpse_contrasts` will give warnings about mismatches between the specified contrasts and what's actually set on the factors in a dataframe. The warnings will typically tell you how to resolve these mismatches. See the `contrasts` and `warnings` vignettes for more information.

Value

A dataframe if return.list is FALSE, a list with a dataframe and list of named contrasts if TRUE.

See Also

`enlist_contrasts()` `set_contrasts()`

Examples

```
my_contrasts <- list(cyl ~ sum_code, carb ~ helmert_code)
my_data <- set_contrasts(mtcars, my_contrasts, verbose = FALSE)
my_data$gear <- factor(my_data$gear) # Make gear a factor manually

# View information about contrasts; gear will use default for unordered
glimpse_contrasts(my_data, my_contrasts)
```

---

helmert_code *Helmert code*

Description

R's `stats::contr.helmert()` function is unscaled, meaning that you need to scale the coefficients of a model fit to get the actual comparisons of interest. This version will automatically scale the contrast matrix such that the coefficients are the expected scaled values.

Usage

```
helmert_code(n)
```

Arguments

n	Integer umber of factor levels to compute contrasts for.

Details

Helmert coding compares each level to the total mean of all levels that have come before it. Differs from backward difference coding, which compares only pairs of levels (not a level to a cumulative mean of levels)

Example interpretation for a 4 level factor:

- Intercept = Grand mean (mean of the means of each level)
- grp2 = mean(grp2) - mean(grp1)
- grp3 = mean(grp3) - mean(grp1, grp2)
- grp4 = mean(grp4) - mean(grp1, grp2, grp3)

Value

A contrast matrix with dimensions n rows and (n-1) columns.

Examples

```
mydf <- data.frame(
  grp = gl(4,5),
  resp = c(seq(1, 5), seq(5, 9), seq(10, 14), seq(15, 19))
)

mydf <- set_contrasts(mydf, grp ~ helmert_code)
lm(resp ~ grp, data = mydf)
```

---

interpret_intercept *Interpret intercept from contrasts*

Description

Given a contrast matrix, try and interpret the intercept. Will usually be either the grand mean, the mean of a reference level (e.g. contr.treatment), the unweighted mean of multiple levels. Anything else would indicate custom weights that the user provided, hence they should know how to interpret it.

Usage

```
interpret_intercept(contrast_matrix)
```

Arguments

`contrast_matrix`	Contrast matrix

Value

A string describing how to interpret the effect on the intercept this coding scheme has

Examples

```
interpret_intercept(contr.treatment(2)) # mean(1)
interpret_intercept(contr.SAS(2)) # mean(2)
interpret_intercept(contr.sum(2)) # grand mean

# Here there are 3 levels but the intercept is either an unweighted
# mean of 2 levels or a weighted mean of 2 levels
unweighted_intercept <-
  solve(t(matrix(c(.5, .5, 0, -1, 1, 0, -1, 0, 1), nrow = 3)))[, 2:3]
weighted_intercept <-
  solve(t(matrix(c(.8, .2, 0, -1, 1, 0, -1, 0, 1), nrow = 3)))[, 2:3]

interpret_intercept(unweighted_intercept) # mean(1,2)
interpret_intercept(weighted_intercept) # custom weights
```

---

is.unordered *Check for unordered factor*

Description

Helper to check if a factor is exclusively unordered. is.factor(x) is TRUE when x is unordered OR ordered.

Usage

```
is.unordered(x)
```

Arguments

x	a vector of data

Value

TRUE if x is an unordered factor, FALSE if x is not a factor or is an ordered factor

Examples

```
is.unordered(gl(5,1)) # True
is.unordered(gl(5,1,ordered = TRUE)) # False
```

is_centered *Check for orthogonality*

Description

Given a contrast matrix or list of contrast matrices (eg from `enlist_contrasts()`), return a logical vector of whether each contrast is centered or not.

Usage

```
is_centered(contrast_matrices, USE.NAMES = FALSE)
```

Arguments

`contrast_matrices`	Contrast matrix or list of contrast matrices
`USE.NAMES`	Logical, whether vector should be named

Value

Logical vector, will retain names of a passed list

Examples

```
is_centered(treatment_code(5)) # FALSE
is_centered(scaled_sum_code(5)) # TRUE
```

is_orthogonal *Check for orthogonality*

Description

Given a contrast matrix or list of contrast matrices (eg from `enlist_contrasts()`), return a logical vector of whether each contrast is orthogonal or not.

Usage

```
is_orthogonal(contrast_matrices, USE.NAMES = FALSE)
```

Arguments

`contrast_matrices`	Contrast matrix or list of contrast matrices
`USE.NAMES`	Logical, whether vector should be named

Value

Logical vector, will retain names of a passed list

Examples

```
is_orthogonal(treatment_code(5)) # FALSE
is_orthogonal(helmert_code(5)) # TRUE
```

`orth_polynomial_code` *Orthogonal Polynomial code*

Description

Wrapper around `stats::contr.poly()`. You can also use `polynomial_code()` as an alias.

Usage

```
orth_polynomial_code(n)

polynomial_code(n)
```

Arguments

`n`	Integer umber of factor levels to compute contrasts for.

Details

For n levels of factors where k in 1:n, generate a matrix with n-1 comparisons where each comparison looks for a polynomial trend of degree k where each polynomial is independent of the others.

Value

A contrast matrix with dimensions n rows and (n-1) columns.

Examples

```
mydf <- data.frame(
  grp = rep(c("a", "b", "c", "d"), each = 2000),
  val = c(
    rnorm(200, 2, 1),
    rnorm(200, 5, 1),
    rnorm(200, 7.5, 1),
    rnorm(200, 15, 1)
  )
) |>
  set_contrasts(grp ~ polynomial_code)

stats::lm(val ~ grp, data = mydf)
```

raw_polynomial_code *Raw Polynomial code*

Description

Make raw polynomial contrast, rather than orthogonal ones. Normally you would use orthogonal polynomials, so make sure this is what you want. Using raw polynomials may increase the collinearity in your model, especially with higher numbers of levels.

Usage

```
raw_polynomial_code(n)
```

Arguments

n	Integer umber of factor levels to compute contrasts for.

Details

For n levels of factors where k in 1:n, generate a matrix with n-1 comparisons where each comparison looks for a polynomial trend of degree k, where each polynomial may be correlated with the others. Normally you would use orthogonal polynomials, see `stats::contr.poly()` and `orth_polynomial_code()`

Value

A contrast matrix with dimensions n rows and (n-1) columns.

Examples

```
mydf <- data.frame(
  grp = rep(c("a", "b", "c", "d"), each = 2000),
  val = c(
    rnorm(200, 2, 1),
    rnorm(200, 5, 1),
    rnorm(200, 7.5, 1),
    rnorm(200, 15, 1)
  )
) |>
  set_contrasts(grp ~ raw_polynomial_code)

stats::lm(val ~ grp, data = mydf)
```

`reverse_helmert_code` *Reverse Helmert code*

Description

Reverse helmert coding is the same concept as helmert coding, but the order of the groupings is reversed. See also helmert_code.

Usage

```
reverse_helmert_code(n)
```

Arguments

`n`	Integer umber of factor levels to compute contrasts for.

Details

Reverse helmert coding compares each level to the total mean of all levels that come after it. Differs from forward difference coding, which only compares pairs of levels (not a level to a cumulative mean of levels).

Example interpretation for a 4 level factor:

- Intercept = Grand mean (mean of the means of each level)
- grp1 = mean(grp4, grp3, grp2) - grp(1)
- grp2 = mean(grp4, grp3) - mean(grp2)
- grp3 = mean(grp3) - mean(grp4)

Value

A contrast matrix with dimensions n rows and (n-1) columns.

Examples

```
mydf <- data.frame(
  grp = gl(4,5),
  resp = c(seq(1, 5), seq(5, 9), seq(10, 14), seq(15, 19))
)

mydf <- set_contrasts(mydf, grp ~ reverse_helmert_code)
lm(resp ~ grp, data = mydf)
```

`scaled_sum_code` *Scaled sum coding*

Description

Contrast coding scheme with a centered intercept and comparisons from a baseline reference level.

Usage

```
scaled_sum_code(n)
```

Arguments

`n`	Integer umber of factor levels to compute contrasts for.

Details

The name for this contrast scheme varies widely in different fields and across experimental psychology papers. It has been called simple, sum, contrast, sum-to-zero, and deviation coding (among other names). This package uses scaled sum coding to explicitly differentiate it from sum coding, which has an implementation in base R with `contr.sum`.

For n levels of factors, generate a matrix with n-1 comparisons where:

- Reference level = -1/n
- Comparison level = (n-1)/n
- All others = -1/n

Example interpretation for a 4 level factor:

- Intercept = Grand mean (mean of the means of each level)
- grp2 = mean(grp2) - mean(grp1)
- grp3 = mean(grp3) - mean(grp1)
- grp4 = mean(grp4) - mean(grp1)

Note: grp coefficient estimates are the same as with contr.treatment, but the intercept is changed to the grand mean instead of the mean of grp1.

It's also important to note that this coding scheme is NOT the same as `contr.sum/2` when the number of levels is greater than 2. When n=2, estimates with `contr.sum` can be interpreted as "half the distance between levels" but when k>2, `contr.sum` is to be interpreted as "the distance between this level and the GRAND MEAN". You may be tempted to use `contr.sum(n)/2`, but this tests the hypothesis that 3/2 times the mean of a level is equal to half the sum of the means of the other levels, i.e., $1.5\mu_1 - .5\mu_2 - .5\mu_3 - .5\mu_4 = 0$, which is not likely to be what you're looking for.

Value

A contrast matrix with dimensions n rows and (n-1) columns.

Examples

```
# Compare these two, note that contr.sum(4)/2 is not the same
scaled_sum_code(4)
contr.sum(4)

# Here they happen to be equivalent (modulo reference level)
scaled_sum_code(2)
contr.sum(2) / 2

mydf <- data.frame(
  grp = gl(4,5),
  resp = c(seq(1, 5), seq(5, 9), seq(10, 14), seq(15, 19))
)

mydf <- set_contrasts(mydf, grp ~ scaled_sum_code)

lm(resp ~ grp, data = mydf)
```

---

set_contrasts *Set contrasts to dataframe*

Description

Uses the same syntax as `enlist_contrasts()`, but returns the dataframe with the new contrasts applied. Use this when your model function doesnt have a contrasts argument and you want to avoid writing `contrasts<-` multiple times. See `enlist_contrasts()` for details about the package-specific syntax.

Usage

```
set_contrasts(
  model_data,
  ...,
  verbose = getOption("contrastable.verbose"),
  print_contrasts = FALSE
)
```

Arguments

`model_data`	Data frame you intend on passing to your model
`...`	A series of 2 sided formulas with factor name on the left hand side and desired contrast scheme on the right hand side. The reference level can be set with `+`, the intercept can be overwritten with `*`, comparison labels can be set using `|`, and trends for polynomial coding can be removed using `-`.
`verbose`	Logical, defaults to FALSE, whether messages should be printed
`print_contrasts`	Logical, default FALSE, whether to print the contrasts set for each factor. Fractions are displayed using `MASS::fractions()`

Details

enlist_contrasts(), set_contrasts(), and glimpse_contrasts() use special syntax to set contrasts for multiple factors. The syntax consists of two-sided formulas with the desired factor column on the left hand side and the contrast specification on the right hand side. For example, varname ~ scaled_sum_code. Many contrasts support additional kinds of contrast manipulations using overloaded operators:

- + X: Set the reference level to the level named X. Only supported for schemes that have a singular reference level such as sum_code(), scaled_sum_code(), treatment_code(), stats::contr.treatment(), stats::contr.sum(), stats::contr.SAS(). Ignored for schemes like helmert_code().
- * X: Overwrite the intercept to the mean of the level named X
- - A:B: For polynomial coding schemes only, drop comparisons A through B.
- | c(...): Change the comparison labels for the contrast matrix to the character vector c(...) of length n-1. These labels will appear in the output/summary of a statistical model. Note that for brms::brm, instances of - (a minus sign) are replaced with M.

You can also specify multiple variables on the left hand side of a formula using tidyselect helpers. See examples for more information.

Typically model functions like lm will have a contrasts argument where you can set the contrasts at model run time, rather than having to manually change the contrasts on the underlying factor columns in your data. This function will return such a named list of contrast matrices to pass to these functions. Note that this function should not be used within a modeling function call, e.g., lm(y~x, data = model_data, contrasts = enlist_contrasts(model_data, x~sum_code)). Often, this will call enlist_contrasts twice, rather than just once.

For some model fitting functions, like brms::brm, there is no contrasts argument. For such cases, use set_contrasts() to set contrasts directly to the factors in a dataframe.

One good way to use enlist_contrasts() is in conjunction with MASS::fractions() to create a list of matrices that can be printed to explicitly show the entire contrast matrices you're using for your models. This can be especially helpful for supplementary materials in an academic paper.

Sometimes when using orthogonal polynomial contrasts from stats::contr.poly() people will drop higher level polynomials for parsimony. Note however that these do capture some amount of variation, so even though they're orthogonal contrasts the lower level polynomials will have their estimates changed. Moreover, you cannot reduce a contrast matrix to a matrix smaller than size n*n-1 in the dataframe you pass to a model fitting function itself, as R will try to fill in the gaps with something else. If you want to drop contrasts you'll need to use something like enlist_contrasts(df, x ~ contr.poly - 3:5) and pass this to the contrasts argument in the model fitting function.

Value

The model_data dataframe, but with updated contrasts.

See Also

enlist_contrasts() glimpse_contrasts()

Examples

```
head(
   set_contrasts(mtcars, carb + cyl ~ helmert_code, print_contrasts = TRUE)
)
```

`sum_code` *Sum code*

Description

Wrapper around contr.sum, but ensures that the reference level is the first level alphabetically, not the last. Returns a contrast matrix where comparisons give differences between comparison levels and the grand mean.

Usage

```
sum_code(n)
```

Arguments

`n`	Integer umber of factor levels to compute contrasts for.

Details

For n levels of factors, generate a matrix with n-1 comparisons where:

- Reference level = -1
- Comparison level = 1
- All others = 0

Example interpretation for a 4 level factor:

- Intercept = Grand mean (mean of the means of each level)
- grp2 = grp2 - mean(grp4, grp3, grp2, grp1)
- grp3 = grp3 - mean(grp4, grp3, grp2, grp1)
- grp4 = grp4 - mean(grp4, grp3, grp2, grp1)

Note that when n = 2, the coefficient estimate is half of the difference between the two levels. But, this coincidence does not hold when the number of levels is greater than 2.

Value

A contrast matrix with dimensions n rows and (n-1) columns.

Examples

```
mydf <- data.frame(
  grp = gl(4,5),
  resp = c(seq(1, 5), seq(5, 9), seq(10, 14), seq(15, 19))
)

mydf <- set_contrasts(mydf, grp ~ sum_code)

lm(resp ~ grp, data = mydf)
```

`treatment_code` *Treatment code*

Description

Wrapper around `stats::contr.treatment()`. Returns a contrast matrix where comparisons give differences between each comparison level and a baseline reference level, while the intercept equals the first level of the factor. See `scaled_sum_code()` for a function that centers the intercept on the grand mean while retaining pairwise comparisons from a reference level.

Usage

```
treatment_code(n)
```

Arguments

`n`	Integer umber of factor levels to compute contrasts for.

Details

For n levels of factors, generate a matrix with n-1 comparisons where:

- Reference level = 0
- Comparison level = 1
- All others = 0

Note that this function sets the first level (alphabetically) as the reference level while `stats::contr.SAS()` sets the LAST level as the reference level. However, in functions like `set_contrasts()`, and `enlist_contrasts()`, the reference level is automatically set to be the first level alphabetically.

Value

A contrast matrix with dimensions n rows and (n-1) columns.

Examples

```
mydf <- data.frame(
  grp = gl(4,5),
  resp = c(seq(1, 5), seq(5, 9), seq(10, 14), seq(15, 19))
)

mydf <- set_contrasts(mydf, grp ~ treatment_code)

lm(resp ~ grp, data = mydf)
```

`use_contrasts` *Contrast code factors*

Description

Helper to do contrast coding. There are two options:

- Manually specify a matrix for code_by (implements use_contrast_matrix). Reference level is automatically set to the row that's always negative.
- Specify a style of contrast coding as a function. Label of the reference level should be specified in ...

Usage

```
use_contrasts(
  factor_col,
  code_by = NA,
  reference_level = NA,
  set_intercept = NA,
  drop_trends = NA,
  labels = NULL,
  as_is = FALSE,
  ...
)
```

Arguments

`factor_col`	The factor column to use, eg data$gender
`code_by`	Either a matrix or a function
`reference_level`	The level to use as the reference level, default NA
`set_intercept`	The intercept to use, default NA
`drop_trends`	Whether to drop trends, default NA
`labels`	Labels to use in the contrast matrix, must equal number of contrasts

as_is	Logical, default FALSE, whether to suppress auto switching of the reference level to the first level if not specified
...	Additional arguments to be passed to use_contrast_function, specifically, which level you want the reference level to be

Value

A contrast coding matrix with labels and proper reference level

Examples

```
# Create a contrast matrix given some factor vector with the specified
# reference level
use_contrasts(gl(5,2), sum_code, reference_level = 3)

# Set column labels; order for labels is the same as the column indices
use_contrasts(gl(3,2), scaled_sum_code, labels = c("2-1", "3-1"))

my_data <- mtcars
my_data$gear <- factor(mtcars$gear)

MASS::fractions(use_contrasts(my_data$gear, helmert_code))
```

---

use_contrasts.AsIs *AsIs method for use_contrasts*

Description

Evaluates code_by, then applies the appropriate use_contrasts method

Usage

```
## S3 method for class 'AsIs'
use_contrasts(
  factor_col,
  code_by = NA,
  reference_level = NA,
  set_intercept = NA,
  drop_trends = NA,
  labels = NULL,
  as_is = FALSE,
  ...
)
```

Arguments

factor_col	A factor vector, eg from df$factorVarName
code_by	A symbol to be evaluated
reference_level	The level to use as the reference level, default NA
set_intercept	The intercept to use, default NA
drop_trends	The trends to drop, default NA
labels	A vector of labels to apply to the matrix column names, default NULL (no new labels)
as_is	Logical, default FALSE, whether to leave the resulting matrix as-is
...	Additional arguments to be passed on

Value

A contrast coding matrix with labels and proper reference level

Examples

```
use_contrasts(gl(5,1), I(scaled_sum_code))
```

---

use_contrasts.default *Default method for use_contrasts*

Description

If a user doesn't specify a contrast matrix, use the defaults from options(). If the user tries to use something we don't know how to work with, throw a warning that we'll be using the defaults from options().

Usage

```
## Default S3 method:
use_contrasts(
  factor_col,
  code_by = NA,
  reference_level = NA,
  set_intercept = NA,
  drop_trends = NA,
  labels = NULL,
  as_is = FALSE,
  ...
)
```

Arguments

factor_col	A factor vector, eg from df$factorVarName
code_by	Some object that's not a matrix or function. If NA, no warning will be thrown, and the default contrasts will be used. A warning will be thrown if it's not NA.
reference_level	Not used
set_intercept	Not used
drop_trends	Not used
labels	A vector of labels to apply to the matrix column names, default
as_is	Logical, default FALSE, whether to leave the resulting matrix
...	Additional arguments, not used

Value

Contrast matrix, using the ordered or unordered default from options()

Examples

```
use_contrasts(gl(5,1), helmert_code) # a function
my_matrix <- helmert_code(5)
use_contrasts(gl(5,1), my_matrix) # a matrix
```

---

use_contrasts.function

Function method for use_contrasts

Description

If the user provides a function, use the function and supplied arguments to create a contrast matrix

Usage

```
## S3 method for class '`function`'
use_contrasts(
  factor_col,
  code_by = NA,
  reference_level = NA,
  set_intercept = NA,
  drop_trends = NA,
  labels = NULL,
  as_is = FALSE,
  ...
)
```

Arguments

factor_col	A factor vector, eg from df$factorVarName
code_by	A function to be called, should return a contrast matrix
reference_level	The name of the level to use as the reference level, default NA
set_intercept	The intercept to use, default NA
drop_trends	The trends to drop, default NA
labels	A vector of labels to apply to the matrix column names, default
as_is	Logical, default FALSE, whether to leave the resulting matrix
...	Additional arguments to be passed to code_by()

Value

A contrast coding matrix with labels and proper reference level

Examples

```
use_contrasts(gl(5,1), sum_code)
```

---

use_contrasts.hypr *Use a hypr object for contrasts*

Description

Use a hypr object for contrasts

Usage

```
## S3 method for class 'hypr'
use_contrasts(
  factor_col,
  code_by = NA,
  reference_level = NA,
  set_intercept = NA,
  drop_trends = NA,
  labels = NULL,
  as_is = FALSE,
  ...
)
```

Arguments

factor_col	A factor vector, eg from df$factorVarName
code_by	A hypr object created with hypr::hypr()
reference_level	Not used
set_intercept	Not used
drop_trends	Not used
labels	A vector of labels to apply to the matrix column names, default
as_is	Logical, default FALSE, whether to leave the resulting matrix
...	Additional arguments, not used

Value

Contrast matrix specified by the hypr object

Examples

```
hypr_obj <- hypr::hypr(a ~ b, c ~ b) # centered pairwise comparisons to b

use_contrasts(factor(c('a', 'b', 'c')), hypr_obj)
```

---

use_contrasts.matrix *Matrix method for use_contrasts*

Description

If a user provides a raw matrix, then use that matrix as the contrast matrix

Usage

```
## S3 method for class 'matrix'
use_contrasts(
  factor_col,
  code_by = NA,
  reference_level = NA,
  set_intercept = NA,
  drop_trends = NA,
  labels = NULL,
  as_is = FALSE,
  ...
)
```

Arguments

factor_col	A factor vector, eg from df$factorVarName
code_by	A matrix to be used as the contrast matrix, should have the same dimensions as the contrast matrix already applied to code_by
reference_level	Not used
set_intercept	Not used
drop_trends	Not used
labels	A vector of labels to apply to the matrix column names, default
as_is	Logical, default FALSE, whether to leave the resulting matrix
...	Additional arguments, not used

Value

A contrast coding matrix with labels and proper reference level

Examples

```
contrast_matrix <- sum_code(4)
use_contrasts(gl(4,1), contrast_matrix)
```

---

use_contrasts.name *Symbol method for use_contrasts*

Description

Evaluates code_by, then applies the appropriate use_contrasts method

Usage

```
## S3 method for class 'name'
use_contrasts(
  factor_col,
  code_by = NA,
  reference_level = NA,
  set_intercept = NA,
  drop_trends = NA,
  labels = NULL,
  as_is = FALSE,
  ...
)
```

Arguments

factor_col	A factor vector, eg from df$factorVarName
code_by	A symbol to be evaluated
reference_level	The level to use as the reference level, default NA
set_intercept	The intercept to use, default NA
drop_trends	The trends to drop, default NA
labels	A vector of labels to apply to the matrix column names, default NULL (no new labels)
as_is	Logical, default FALSE, whether to leave the resulting matrix as-is
...	Additional arguments to be passed on

Value

A contrast coding matrix with labels and proper reference level

Examples

```
aliased_scheme <- sum_code
contrast_scheme <- rlang::sym("aliased_scheme")

# Result will be as if sum_code was used directly
use_contrasts(gl(5,1), contrast_scheme)
```

Index