Package 'speccurvieR'

July 18, 2026

Type Package

Title Easy, Fast, and Pretty Specification Curve Analysis

Version 1.0.0

Description Making specification curve analysis easy, fast, and pretty. It improves upon existing offerings with additional features and 'tidyverse' integration. Users can easily visualize and evaluate how their models behave under different specifications with a high degree of customization. For a description and applications of specification curve analysis see Simonsohn, Simmons, and Nelson (2020) <doi:10.1038/s41562-020-0912-z>.

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports ggplot2, patchwork, magrittr, tidyr, dplyr, stringr, combinat, fixest, pbapply, parallel, lmtest, sandwich, generics

Suggests testthat (>= 3.0.0), knitr, rmarkdown, gt, kableExtra, flextable, modelsummary

Config/testthat/edition 3

RoxygenNote 7.3.3

License MIT + file LICENSE

URL https://github.com/zaynesember/speccurvieR

BugReports https://github.com/zaynesember/speccurvieR/issues

VignetteBuilder knitr

NeedsCompilation no

Author Zayne Sember [aut, cre, cph]

Maintainer Zayne Sember <zayne@mit.edu>

Repository CRAN

Date/Publication 2026-07-18 16:50:02 UTC

Contents

`bottles` *CalCOFI Bottle Data*

Description

A subset of data from the California Cooperative Oceanic Fisheries Investigations. Each observation describes a sample of ocean water collected.

Usage

```
bottles
```

Format

‘bottles‘ A data frame with 500 rows and 62 columns:

Cst_Cnt Cast count

Btl_Cnt Bottle Count

Sta_ID Line and Station

Depth_ID Depth ID

Depthm Bottle depth in meters

T_degC Water temperature in degrees Celsius

Salnty Salinity (Practical Salinity Scale 1978)

O2ml_L Milliliters of oxygen per liter of seawater

STheta Potential Density (Sigma Theta), Kg/M^3

O2Sat Oxygen percent saturation

Oxy_µmol/Kg Oxygen micromoles per kilogram seawater

BtlNum Niskin bottle sample was collected from

RecInd Record Indicator

T_prec Temperature Precision

T_qual Quality Code

S_prec Salinity Precision

S_qual Quality Code

P_qual Quality Code

O_qual Quality Code

SThtaq Quality Code

O2Satq Quality Code

ChlorA Micrograms Chlorophyll-a per liter seawater

Chlqua Quality Code

Phaeop Micrograms Phaeopigment per liter seawater

Phaqua Quality Code

PO4uM Micromoles Phosphate per liter of seawater

PO4q Quality Code

SiO3uM Micromoles Silicate per liter of seawater

SiO3qu Quality Code

NO2uM Micromoles Nitrite per liter of seawater

NO2q Quality Code

NO3uM Micromoles Nitrate per liter of seawater

NO3q Quality Code

NH3uM Micromoles Ammonia per liter of seawater

NH3q Quality Code

C14As1 14C Assimilation of Replicate 1

C14A1p Precision of 14C Assimilation of Replicate 1

C14A1q Quality Code

C14As2 14C Assimilation of Replicate 2

C14A2p Precision of 14C Assimilation of Replicate 2

C14A2q Quality Code

DarkAs 14C Assimilation of Dark/Control Bottle

DarkAp Precision of 14C Assimilationof Dark/Control Bottle

Darkaq Quality Code

MeanAs Mean 14C Assimilation of Replicates 1 and 2

MeanAp Precision of Mean 14C Assimilation of Replicates 1 and 2

MeanAq Quality Code

IncTim Elapsed incubation time of primary productivity experiment

LightP Light intensities of the incubation tubes

R_Depth Reported Depth (from pressure) in meters

R_Temp Reported (Potential) Temperature in degrees Celsius

R_Sal Reported Salinity (from Specific Volume Anomoly, M^3/Kg)

R_DYNHT Reported Dynamic Height in units of dynamic meters

R_Nuts Reported Ammonium concentration

R_Oxy_µmol/Kg Reported Oxygen micromoles/kilogram

DIC1 Dissolved Inorganic Carbon micromoles per kilogram solution

DIC2 Dissolved Inorganic Carbon on a replicate sample

TA1 Total Alkalinity micromoles per kilogram solution

TA2 Total Alkalinity on a replicate sample

pH1 pH (the degree of acidity/alkalinity of a solution)

pH2 pH on a replicate sample

DIC Quality Comment Quality Comment

Source

<https://calcofi.org/data/oceanographic-data/bottle-database/>

plot_aic *Plots the AIC across model specifications.*

Description

plot_aic() plots the Akaike information criterion across model specifications. Only available for nonlinear regression models.

Usage

```
plot_aic(sca_data, title = "", show_index = TRUE, plot_vars = TRUE)
```

Arguments

sca_data	A data frame returned by 'sca()' containing model estimates from the specification curve analysis.
title	A string to use as the plot title. Defaults to an empty string, '""'.
show_index	A boolean indicating whether to label the model index on the the x-axis. Defaults to 'TRUE'.
plot_vars	A boolean indicating whether to include a panel on the plot showing which variables are present in each model. Defaults to 'TRUE'.

Value

If 'plot_vars = TRUE' a 'patchwork' object combining the plot and the variable panel; if 'plot_vars = FALSE' a ggplot object.

Examples

```
plot_aic(sca_data = sca(y = "Salnty", x = "T_degC",
                        controls = c("ChlorA", "O2Sat"),
                        data = bottles, progress_bar = TRUE, parallel = FALSE),
                      title = "AIC");
plot_aic(sca_data = sca(y = "Salnty", x = "T_degC",
                        controls = c("ChlorA*O2Sat"),
                        data = bottles, progress_bar = FALSE,
                        parallel = FALSE),
                        show_index = FALSE, plot_vars = FALSE);

plot_aic(sca_data = sca(y = "Salnty", x = "T_degC",
                          controls = c("ChlorA*NO3uM", "O2Sat*NO3uM"),
                          data = bottles,
                          progress_bar = TRUE, parallel = TRUE, workers = 2));
```

`plot_coef_fit` *Plots the independent variable's coefficient against model fit.*

Description

plot_coef_fit() plots the independent variable's coefficient against a measure of model fit across specifications, revealing whether better-fitting models tend to produce systematically different estimates (i.e. whether your best-fitting specifications are outliers).

Usage

```
plot_coef_fit(sca_data, metric = NULL, title = "")
```

Arguments

`sca_data`	A data frame returned by ‘sca()‘.
`metric`	A string naming the fit measure to plot against, one of ‘"RMSE"‘, ‘"adjR"‘, ‘"AIC"‘, or ‘"deviance"‘. Defaults to ‘NULL‘, in which case the first measure available in ‘sca_data‘ is used.
`title`	A string to use as the plot title. Defaults to ‘""‘.

Value

A ggplot object.

Examples

```
plot_coef_fit(sca(y = "Salnty", x = "T_degC",
                controls = c("ChlorA", "O2Sat", "NO2uM"),
                data = bottles, progress_bar = FALSE));
```

`plot_control_distributions`
Plots control variable distributions.

Description

plot_control_distributions() plots the distribution of coefficients for each control variable included in the model specifications.

Usage

```
plot_control_distributions(
  sca_data,
  title = "",
  type = "density",
  zero_line = TRUE
)
```

Arguments

sca_data	A data frame returned by ‘sca()‘ containing model estimates from the specification curve analysis.
title	A string to use as the plot title. Defaults to an empty string, ‘""‘.
type	A string indicating what type of distribution plot to produce. When ‘type = "density"‘ density plots are produced. When ‘type = "hist"‘ or ‘type = "histogram"‘ histograms are produced. Defaults to ‘"density"‘.
zero_line	A boolean indicating whether to draw a dashed reference line at zero, making it easy to see whether a control's effect crosses zero. Defaults to ‘TRUE‘.

Value

A ggplot object.

Examples

```
plot_control_distributions(sca_data = sca(y="Salnty", x="T_degC",
                                  controls = c("ChlorA", "O2Sat"),
                                  data = bottles,
                                  progress_bar = TRUE, parallel = FALSE),
                           title = "Control Variable Distributions")
plot_control_distributions(sca_data = sca(y = "Salnty", x="T_degC",
                                  controls = c("ChlorA*O2Sat"),
                                  data = bottles,
                                  progress_bar = FALSE, parallel = FALSE),
                           type = "hist")

plot_control_distributions(sca_data = sca(y = "Salnty", x = "T_degC",
                                  controls = c("ChlorA*NO3uM",
                                               "O2Sat*NO3uM"),
                                  data = bottles, progress_bar = TRUE,
                                  parallel = TRUE, workers = 2),
                           type = "density")
```

---

plot_curve *Plots a specification curve.*

Description

plot_curve() takes the data frame output of sca() and produces a ggplot of the independent variable's coefficient (as indicated in the call to sca()) across model specifications. By default a panel is added showing which control variables are present in each model. The combined plot is returned as a ‘patchwork‘ object, so it can be further customised with ggplot2 and patchwork operators (e.g. ‘& theme_sca(base_size = 14)‘).

Usage

```
plot_curve(
  sca_data,
  title = "",
  show_index = TRUE,
  plot_vars = TRUE,
  ylab = "Coefficient",
  plot_se = "bar",
  median_line = FALSE,
  point_size = NULL
)
```

Arguments

sca_data	A data frame returned by ‘sca()‘ containing model estimates from the specification curve analysis.
title	A string to use as the plot title. Defaults to an empty string, ‘""‘.
show_index	A boolean indicating whether to label the model index on the the x-axis. Defaults to ‘TRUE‘.
plot_vars	A boolean indicating whether to include a panel on the plot showing which variables are present in each model. Defaults to ‘TRUE‘.
ylab	A string to be used as the y-axis label. Defaults to ‘"Coefficient"‘.
plot_se	A string indicating whether to display standard errors as bars or plots. For bars ‘plot_se = "bar"‘, for ribbons ‘plot_se = "ribbon"‘. If any other value is supplied then no standard errors are included. Defaults to ‘"bar"‘.
median_line	A boolean indicating whether to add a dotted line at the median coefficient across specifications. Defaults to ‘FALSE‘.
point_size	A number giving the size of the plotted points. Defaults to ‘NULL‘, in which case a size is chosen automatically from the number of specifications.

Value

If ‘plot_vars = TRUE‘ a ‘patchwork‘ object combining the curve and the variable panel; if ‘plot_vars = FALSE‘ a ggplot object. Both can be further customised with ggplot2 (and patchwork) operators.

Examples

```
plot_curve(sca_data = sca(y="Salnty", x="T_degC", c("ChlorA", "O2Sat"),
                          data=bottles, progress_bar=TRUE, parallel=FALSE),
                       title = "Salinity and Temperature Models",
                       show_index = TRUE, plot_vars = TRUE,
                       ylab = "Coefficient value", plot_se = "ribbon");
plot_curve(sca_data = sca(y="Salnty", x="T_degC",
                          c("ChlorA*O2Sat", "ChlorA", "O2Sat"),
                          data=bottles, progress_bar=FALSE, parallel=FALSE),
                       show_index = TRUE, plot_vars = TRUE,
                       plot_se = "ribbon");
```

```
plot_curve(sca_data = sca(y="Salnty", x="T_degC",
                          c("ChlorA*NO3uM", "O2Sat", "ChlorA", "NO3uM"),
                          data=bottles,
                          progress_bar = TRUE, parallel = TRUE, workers=2),
           plot_se="");
```

---

plot_deviance *Plots the deviance of residuals across model specifications.*

Description

plot_deviance() plots the deviance of residuals across model specifications. Only available for linear regression models.

Usage

```
plot_deviance(sca_data, title = "", show_index = TRUE, plot_vars = TRUE)
```

Arguments

sca_data	A data frame returned by 'sca()' containing model estimates from the specification curve analysis.
title	A string to use as the plot title. Defaults to an empty string, '""'.
show_index	A boolean indicating whether to label the model index on the the x-axis. Defaults to 'TRUE'.
plot_vars	A boolean indicating whether to include a panel on the plot showing which variables are present in each model. Defaults to 'TRUE'.

Value

If 'plot_vars = TRUE' a 'patchwork' object combining the plot and the variable panel; if 'plot_vars = FALSE' a ggplot object.

Examples

```
plot_deviance(sca_data = sca(y = "Salnty", x = "T_degC",
                             controls = c("ChlorA", "O2Sat"),
                             data = bottles, progress_bar = TRUE,
                             parallel = FALSE),
                       title = "Model Deviance");
plot_deviance(sca_data = sca(y = "Salnty", x = "T_degC",
                             controls = c("ChlorA*O2Sat"),
                             data = bottles, progress_bar = FALSE,
                             parallel = FALSE),
                       show_index = FALSE, plot_vars = FALSE);
```

```
plot_deviance(sca_data = sca(y = "Salnty", x="T_degC",
                         controls = c("ChlorA*NO3uM", "O2Sat*NO3uM"),
                         data = bottles, progress_bar = TRUE, parallel = TRUE,
                         workers = 2));
```

`plot_influence` *Plots how each control influences the independent variable's coefficient.*

Description

plot_influence() shows, for every control variable, the distribution of the independent variable's coefficient across the specifications that include versus exclude that control. It makes clear which modelling choices move the estimate, and by how much.

Usage

```
plot_influence(sca_data, title = "")
```

Arguments

`sca_data`	A data frame returned by 'sca()'.
`title`	A string to use as the plot title. Defaults to '""'.

Value

A ggplot object with one facet per control comparing the coefficient when that control is excluded versus included.

Examples

```
plot_influence(sca(y = "Salnty", x = "T_degC",
                   controls = c("ChlorA", "O2Sat", "NO2uM"),
                   data = bottles, progress_bar = FALSE));
```

`plot_multi_se` *Plots a specification curve under multiple standard error types.*

Description

plot_multi_se() estimates every specification (as 'sca()' does) and, for each, computes the independent variable's standard error under several types via 'se_compare()'. It then plots the specification curve faceted by standard error type: the coefficient estimates are identical across facets, but the confidence intervals – and hence which specifications are "significant" – change with the choice of standard error, showing how sensitive your conclusions are to that choice.

Usage

```
plot_multi_se(
  y,
  x,
  controls,
  data,
  types = c("iid", "HC3"),
  cluster = NULL,
  fixed_effects = NULL,
  level = 0.95,
  title = ""
)
```

Arguments

y	A string containing the column name of the dependent variable in data. Alternatively, a two-sided formula specifying the whole model, e.g. 'y ~ x + control1 + control2 | fixedEffect'. When a formula is supplied the first right-hand-side term is taken as the independent variable 'x', the remaining terms as 'controls', and anything after '|' as 'fixed_effects'; the 'x', 'controls', and 'fixed_effects' arguments are then taken from the formula. 'data' may be passed positionally in this case, e.g. 'sca(y ~ x + z, data)'.
x	A string containing the column name of the independent variable in data.
controls	A vector of strings containing the column names of the control variables in data.
data	A dataframe containing y, x, controls, and (optionally) the variables to be used for fixed effects or clustering.
types	A vector of standard error types to compare, passed to 'se_compare()': '"iid"', the '"HC*"' types, or (with 'cluster') clustered types. Defaults to 'c("iid", "HC3")'. Bootstrapped standard errors are not supported here; use 'se_compare()' directly for those.
cluster	Optional clustering variable(s) passed to 'se_compare()'.
fixed_effects	A string containing the column name of the variable in data desired for fixed effects. Defaults to NULL in which case no fixed effects are included.
level	The confidence level used for the intervals. Defaults to '0.95'.
title	A string to use as the plot title. Defaults to '""'.

Value

A ggplot object: the specification curve faceted by standard error type, points coloured by significance under each type.

Examples

```
plot_multi_se(y = "Salnty", x = "T_degC", controls = c("ChlorA", "O2Sat"),
            data = bottles, types = c("iid", "HC1", "HC3"));
```

plot_r2_adj *Plots the adj. R-squared across model specifications.*

Description

plot_r2_adj() plots the adjusted R-squared across model specifications. Only available for linear regression models. Note when fixed effects are are specified the within adjusted R-squared is used (i.e. ‘fixest::r2()‘ with ‘type="war2"‘).

Usage

```
plot_r2_adj(sca_data, title = "", show_index = TRUE, plot_vars = TRUE)
```

Arguments

sca_data	A data frame returned by ‘sca()‘ containing model estimates from the specification curve analysis.
title	A string to use as the plot title. Defaults to an empty string, ‘""‘.
show_index	A boolean indicating whether to label the model index on the the x-axis. Defaults to ‘TRUE‘.
plot_vars	A boolean indicating whether to include a panel on the plot showing which variables are present in each model. Defaults to ‘TRUE‘.

Value

If ‘plot_vars = TRUE‘ a ‘patchwork‘ object combining the plot and the variable panel; if ‘plot_vars = FALSE‘ a ggplot object.

Examples

```
plot_r2_adj(sca_data = sca(y = "Salnty", x = "T_degC",
                           controls = c("ChlorA", "O2Sat"),
                           data = bottles, progress_bar = TRUE,
                           parallel = FALSE),
                       title = "Adjusted R^2");
plot_r2_adj(sca_data = sca(y="Salnty", x="T_degC",
                           controls = c("ChlorA*O2Sat"),
                           data = bottles, progress_bar = FALSE,
                           parallel = FALSE),
                       show_index = FALSE, plot_vars = FALSE);

plot_r2_adj(sca_data = sca(y = "Salnty", x = "T_degC",
                           controls = c("ChlorA*NO3uM", "O2Sat*NO3uM"),
                           data = bottles,
                           progress_bar = TRUE, parallel = TRUE, workers = 2));
```

plot_rmse *Plots RMSE across model specifications.*

Description

plot_rmse() plots the root mean square error across model specifications. Only available for linear regression models.

Usage

```
plot_rmse(sca_data, title = "", show_index = TRUE, plot_vars = TRUE)
```

Arguments

sca_data	A data frame returned by 'sca()' containing model estimates from the specification curve analysis.
title	A string to use as the plot title. Defaults to an empty string, '""'.
show_index	A boolean indicating whether to label the model index on the the x-axis. Defaults to 'TRUE'.
plot_vars	A boolean indicating whether to include a panel on the plot showing which variables are present in each model. Defaults to 'TRUE'.

Value

If 'plot_vars = TRUE' a 'patchwork' object combining the plot and the variable panel; if 'plot_vars = FALSE' a ggplot object.

Examples

```
plot_rmse(sca_data = sca(y="Salnty", x="T_degC", c("ChlorA", "O2Sat"),
                         data=bottles, progress_bar=TRUE, parallel=FALSE),
                       title = "RMSE");
plot_rmse(sca_data = sca(y="Salnty", x="T_degC", c("ChlorA*O2Sat"),
                         data=bottles, progress_bar=FALSE, parallel=FALSE),
                       show_index = FALSE, plot_vars = FALSE);

plot_rmse(sca_data = sca(y="Salnty", x="T_degC",
                         c("ChlorA*NO3uM", "O2Sat*NO3uM"), data=bottles,
                         progress_bar = TRUE, parallel=TRUE, workers=2));
```

plot_samplesizes *Plots the number of observations across model specifications.*

Description

plot_samplesizes() plots ‘n_obs‘, the number of observations each specification was fit on, against the specification index. It makes visible whether specifications were fit on different samples – with default listwise deletion they often are – which is a reason to consider ‘sca(common_sample = TRUE)‘.

Usage

```
plot_samplesizes(sca_data, title = "")
```

Arguments

sca_data	A data frame returned by ‘sca()‘ containing model estimates from the specification curve analysis.
title	A string to use as the plot title. Defaults to an empty string, ‘""‘.

Value

A ggplot object.

See Also

[sca()] and its ‘common_sample‘ argument.

Examples

```
plot_samplesizes(sca(y = "Salnty", x = "T_degC",
                     controls = c("ChlorA", "O2Sat", "NO2uM"),
                     data = bottles, progress_bar = FALSE, parallel = FALSE))
```

plot_sca_test *Plot the null distribution of a joint-inference test*

Description

‘plot_sca_test()‘ visualises the output of [sca_test()]. For each test statistic it draws the permutation null distribution with the observed value marked and the permutation p-value annotated, making it easy to see how extreme the observed specification curve is relative to the sharp null.

Usage

```
plot_sca_test(test_result, type = "histogram", title = "")
```

Arguments

test_result	An object of class ‘"sca_test"‘ returned by [sca_test()].
type	A string, ‘"histogram"‘ (default) or ‘"density"‘, selecting how the null distribution is drawn.
title	A string used as the plot title. Defaults to ‘""‘.

Value

A ggplot object with one facet per test statistic.

Examples

```
result <- sca_test(y = "Salnty", x = "T_degC", controls = c("ChlorA", "O2Sat"),
                   data = bottles, n_permutations = 100, progress_bar = FALSE)
plot_sca_test(result)
```

---

plot_sca_test_specs *Plot the specification curve against its per-specification null band*

Description

‘plot_sca_test_specs()‘ draws the observed specification curve (each specification’s focal coefficient, ranked) together with a shaded band giving that specification’s own null distribution under the permutation test. This is the specification-curve view of the joint-inference test of Simonsohn, Simmons, and Nelson (2020): specifications whose observed estimate falls outside their null band are highlighted, so it is easy to see which parts of the curve are more extreme than chance.

When the result carries per-specification family-wise-error-rate-adjusted p-values (i.e. it was computed with ‘keep_curves = TRUE‘ and a confound-preserving null, so [sca_minp()] has run – automatically or explicitly), the points are coloured in three tiers – within the chance band, beyond it but not significant after correction, and significant after multiple-comparison correction (enlarged) – so the specifications that survive correction stand out. Otherwise the usual two-tier within/outside-band colouring is used.

It requires an [sca_test()] result computed with ‘keep_curves = TRUE‘.

Usage

```
plot_sca_test_specs(test_result, level = 0.95, title = "")
```

Arguments

test_result	An object of class ‘"sca_test"‘ returned by [sca_test()] with ‘keep_curves = TRUE‘.
level	The width of the null band, as a probability. Defaults to ‘0.95‘ (the 2.5th to 97.5th percentile of each specification’s null coefficients).
title	A string used as the plot title. Defaults to ‘""‘.

Value

A ggplot object.

References

Simonsohn, U., Simmons, J. P., & Nelson, L. D. (2020). Specification curve analysis. *Nature Human Behaviour*, 4, 1208-1214. doi:10.1038/s415620200912z

Examples

```
result <- sca_test(y = "Salnty", x = "T_degC", controls = c("ChlorA", "O2Sat"),
                   data = bottles, n_permutations = 100, keep_curves = TRUE,
                   progress_bar = FALSE)
plot_sca_test_specs(result)
```

---

plot_se *Plots standard error estimates across types.*

Description

plot_se() takes the data frame output of ‘se_compare()‘ and plots, for each coefficient, the estimate together with a confidence interval derived from every available type of standard error. This makes it easy to see how inference about a coefficient changes with the choice of standard error.

Usage

```
plot_se(se_data, level = 0.95, intercept = FALSE, title = "")
```

Arguments

se_data	A data frame returned by ‘se_compare()‘.
level	The confidence level used for the intervals. Defaults to ‘0.95‘.
intercept	A boolean indicating whether to include the ‘"(Intercept)"‘ coefficient. Defaults to ‘FALSE‘.
title	A string to use as the plot title. Defaults to an empty string, ‘""‘.

Value

A ggplot object with one facet per coefficient; within each facet the estimate is plotted against each standard error type with a confidence interval, and the colour indicates whether that interval excludes zero.

Examples

```
plot_se(se_compare(formula = "Salnty ~ T_degC + ChlorA", data = bottles,
                   types = c("iid", "HC0", "HC3")));
plot_se(se_compare(formula = "Salnty ~ T_degC + ChlorA", data = bottles,
                   types = "HC1", cluster = c("Sta_ID", "Depth_ID")),
        level = 0.9);
```

`plot_variance` *Plot the variance decomposition of a specification curve*

Description

‘plot_variance()‘ visualises [sca_variance()] as a bar chart of the share of the focal coefficient’s variance attributable to each control choice (plus a ‘"Residual"‘ bar), making it easy to see which modelling choices drive the spread of estimates across the curve.

Usage

```
plot_variance(
  sca_data,
  estimate = "coef",
  method = c("lmg", "type2"),
  residual = TRUE,
  title = ""
)
```

Arguments

`sca_data`	A data frame returned by [sca()].
`estimate`	A string naming the numeric column to decompose. Defaults to ‘"coef"‘ (the focal coefficient).
`method`	One of ‘"lmg"‘ (default; LMG/Shapley decomposition of R-squared) or ‘"type2"‘ (normalised Type II partial sums of squares). ‘"shapley"‘ is an alias for ‘"lmg"‘.
`residual`	A boolean indicating whether to append a ‘"Residual"‘ row for the unexplained (and choice-interaction) variance. Defaults to ‘TRUE‘. When ‘FALSE‘, the control percentages are rescaled to sum to 100.
`title`	A string used as the plot title. Defaults to ‘""‘.

Value

A ggplot object.

See Also

[sca_variance()] for the underlying table.

Examples

```
s <- sca(y = "Salnty", x = "T_degC",
         controls = c("ChlorA", "O2Sat", "NO2uM"),
         data = bottles, progress_bar = FALSE, parallel = FALSE)
plot_variance(s)
```

---

plot_vars *Plots the variables in each model.*

Description

plot_vars() plots the variables included in each model specification in order of model index. Returns a ggplot object that can then be combined with the output of other functions like plot_rmse() if further customization of each plot is desired.

Usage

```
plot_vars(sca_data, title = "", color_controls = FALSE)
```

Arguments

sca_data	A data frame returned by 'sca()' containing model estimates from the specification curve analysis.
title	A string to use as the plot title. Defaults to an empty string, '""'.
color_controls	A boolean indicating whether to give each variable a color to improve readability. Defaults to 'FALSE'.

Value

A ggplot object.

Examples

```
plot_vars(sca_data = sca(y = "Salnty", x = "T_degC",
                         controls = c("ChlorA", "O2Sat"),
                         data = bottles, progress_bar = TRUE,
                         parallel = FALSE),
                       title = "Model Variable Specifications");
plot_vars(sca_data = sca(y = "Salnty", x = "T_degC",
                         controls = c("ChlorA*O2Sat"),
                         data = bottles, progress_bar = FALSE,
                         parallel = FALSE),
                       color_controls = TRUE);

plot_vars(sca_data = sca(y = "Salnty", x = "T_degC",
                         controls = c("ChlorA*NO3uM", "O2Sat*NO3uM"),
                         data = bottles,
                         progress_bar = TRUE, parallel = TRUE, workers = 2));
```

print.sca_test *Print a specification curve joint-inference test*

Description

Print a specification curve joint-inference test

Usage

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

Arguments

x	An object of class ‘"sca_test"‘ returned by [sca_test()].
...	Ignored.

Value

‘x‘, invisibly.

sca *Perform specification curve analysis*

Description

sca() is the workhorse function of the package–this estimates models with every possible combination of the controls supplied and returns a data frame where each row contains the pertinent information and parameters for a given model by default. This data frame can then be input to plot_curve() or any other plotting function in the package. Alternatively, if ‘return_formulae = TRUE‘, it returns a list of formula objects with every possible combination of controls.

Usage

```
sca(
  y,
  x,
  controls,
  data,
  weights = NULL,
  family = "linear",
  link = NULL,
  fixed_effects = NULL,
  common_sample = FALSE,
  return_formulae = FALSE,
```

```
  progress_bar = TRUE,
  parallel = FALSE,
  workers = 2,
  ...
)
```

Arguments

y	A string containing the column name of the dependent variable in data. Alternatively, a two-sided formula specifying the whole model, e.g. 'y ~ x + control1 + control2 | fixedEffect'. When a formula is supplied the first right-hand-side term is taken as the independent variable 'x', the remaining terms as 'controls', and anything after '|' as 'fixed_effects'; the 'x', 'controls', and 'fixed_effects' arguments are then taken from the formula. 'data' may be passed positionally in this case, e.g. 'sca(y ~ x + z, data)'.
x	A string containing the column name of the independent variable in data.
controls	A vector of strings containing the column names of the control variables in data.
data	A dataframe containing y, x, controls, and (optionally) the variables to be used for fixed effects or clustering.
weights	Optional string with the column name in 'data' that contains weights.
family	A string indicating the family of models to be used. Defaults to "linear" for OLS regression but supports all families supported by 'glm()'.
link	A string specifying the link function to be used for the model. Defaults to 'NULL' for OLS regression using 'lm()' or 'fixest::feols()' depending on whether fixed effects are supplied. Supports all link functions supported by the family parameter of 'glm()'.
fixed_effects	A string containing the column name of the variable in data desired for fixed effects. Defaults to NULL in which case no fixed effects are included.
common_sample	A boolean. When 'TRUE', every specification is fit on the same sample: the rows that are complete across all model variables (the dependent, independent, control, fixed- effects, and weights variables). When 'FALSE' (the default) each specification uses its own complete cases, so specifications with different controls may be fit on different samples; the returned 'n_obs' column reveals this. See also [plot_samplesizes()].
return_formulae	A boolean. When 'TRUE' a list of model formula objects is returned but the models are not estimated. Defaults to 'FALSE' in which case a dataframe of model results is returned.
progress_bar	A boolean indicating whether the user wants a progress bar for model estimation. Defaults to 'TRUE'.
parallel	A boolean indicating whether to parallelize model estimation. Parallelization only offers a speed advantage when a large (> 1000) number of models is being estimated. Defaults to 'FALSE'.
workers	An integer indicating the number of workers to use for parallelization. Defaults to 2.
...	Deprecated camelCase arguments ('fixedEffects', 'returnFormulae', 'progressBar'); use the snake_case equivalents instead.

Value

When ‘return_formulae‘ is ‘FALSE‘, a dataframe where each row contains the independent variable coefficient estimate, standard error, test statistic, p-value, model specification, measures of model fit, and ‘n_obs‘, the number of observations the specification was fit on.

Examples

```
sca(y = "Salnty", x = "T_degC", controls = c("ChlorA", "O2Sat"),
    data = bottles, progress_bar = TRUE, parallel = FALSE);
# Equivalent call using the formula interface:
sca(Salnty ~ T_degC + ChlorA + O2Sat, data = bottles, progress_bar = FALSE);
# Formula interface with an interaction control and fixed effects:
sca(Salnty ~ T_degC + ChlorA + ChlorA*O2Sat | Sta_ID, data = bottles,
    progress_bar = FALSE);

sca(y = "Salnty", x = "T_degC", controls = c("ChlorA*NO3uM", "O2Sat*NO3uM"),
    data = bottles, progress_bar = TRUE, parallel = TRUE, workers = 2);

sca(y = "Salnty", x = "T_degC", controls = c("ChlorA", "O2Sat*NO3uM"),
    data = bottles, progress_bar = TRUE, parallel = FALSE,
    return_formulae = TRUE);
```

---

sca_minp *Family-wise-error-rate-adjusted p-values for each specification*

Description

‘sca_minp()‘ adds, to an existing [sca_test()] result, a multiple-comparison- corrected p-value for *every* specification in the curve. It answers the question the per-specification null band only shows descriptively: *which* individual specifications are more extreme than chance once you account for having searched all of them?

It reuses the permutation null [sca_test()] already computed (so it never re-runs the permutations) and requires that result to have been produced with ‘keep_curves = TRUE‘ and a confound-preserving null (‘null_type = "freedman_lane"‘ or ‘"residual_bootstrap"‘). When those prerequisites are met, [sca_test()] already attaches the single-step adjustment automatically; call ‘sca_minp()‘ to recompute it with a different method or significance threshold.

Usage

```
sca_minp(result, method = c("single_step", "step_down"), alpha = NULL)
```

Arguments

result	An object of class ‘"sca_test"‘ returned by [sca_test()] with ‘keep_curves = TRUE‘ and a confound-preserving ‘null_type‘.
method	One of ‘"single_step"‘ (default) or ‘"step_down"‘; see Details.
alpha	The significance threshold for ‘significant_adj‘. Defaults to ‘NULL‘, which inherits the ‘alpha‘ used by [sca_test()].

Details

Reporting a specification as "significant" because its ordinary p-value is below 0.05 is misleading when the curve contains many correlated specifications: searching dozens of them all but guarantees some will look significant by chance. ‘sca_minp()‘ corrects for this with the min-P / max-statistic permutation method of Westfall and Young (1993). For each permutation it records the most extreme specification anywhere in the curve, building the null distribution of "the best result a search could turn up by chance", and compares each observed specification to that distribution.

Each specification is compared to its *own* permutation null rather than to zero. This matters under the confound-preserving nulls: an under-controlled specification’s null distribution is centred on its confounded value, not on zero, so a specification is flagged only when its estimate is more extreme than that specification could itself produce under the null. Consequently a flagged under-controlled specification indicates an association beyond the superset-conditional sharp null – it is **not** a causal effect of size (estimate minus null centre), because omitting a confounder re-routes part of the focal variable’s coefficient.

‘method = "single_step"‘ (the default, and what [sca_test()] attaches automatically) controls the family-wise error rate in the weak sense, under the global null of no effect in any specification, with no further assumptions. ‘method = "step_down"‘ applies Westfall and Young’s free step-down refinement, which is more powerful and controls the family-wise error rate in the strong sense under subset pivotality (exact for ‘"freedman_lane"‘ via the common control-superset conditioning; approximate for ‘"residual_bootstrap"‘ and under partial alternatives). Adjusted p-values have a resolution floor of ‘1 / (n_eff + 1)‘, where ‘n_eff‘ is the number of permutations that estimated at least one specification (equal to ‘n_used‘ unless some permutations were wholly non-estimable).

Value

The input ‘result‘, augmented with a ‘fwer‘ element under ‘null_curves‘: a list with ‘specs‘ (a data frame with one row per specification giving its term-set key ‘spec‘, ‘observed‘ coefficient, uncorrected own-null p-value ‘p_raw‘, FWER-adjusted p-value ‘p_adj‘, and logical ‘significant_adj‘) and ‘summary‘ (the method, alpha, whether weak or strong FWER is controlled, the number of distinct specifications tested, the number significant after correction, and the smallest adjusted p-value).

References

Westfall, P. H., & Young, S. S. (1993). *Resampling-based multiple testing: Examples and methods for p-value adjustment*. Wiley.

See Also

[sca_test()], which attaches the single-step adjustment automatically; [plot_sca_test_specs()] to see which specifications survive correction.

Examples

```
result <- sca_test(y = "Salnty", x = "T_degC",
                   controls = c("ChlorA", "O2Sat", "NO2uM"), data = bottles,
                   null_type = "freedman_lane", n_permutations = 199,
                   keep_curves = TRUE, seed = 1, progress_bar = FALSE)
# The single-step adjustment is already attached; recompute it step-down.
result <- sca_minp(result, method = "step_down")
```

```
as.data.frame(result, what = "specs")
```

`sca_report` *Write a results paragraph for a specification curve analysis*

Description

'sca_report()' returns a one-paragraph, manuscript-ready description of a specification curve. For an [sca_test()] object it states the median estimate, the share of significant specifications, the joint-inference statistics with their permutation p-values, and (when the full set of statistics is available) whether the null of no effect is rejected. For an [sca()] curve it gives a purely descriptive summary.

Usage

```
sca_report(x, digits = 3, ...)
```

Arguments

`x`	An object returned by [sca()] or [sca_test()].
`digits`	Number of significant digits for estimates. Defaults to '3'.
`...`	Ignored.

Value

A length-one character string.

See Also

[sca_table()] for a tabular summary.

Examples

```
s <- sca(y = "Salnty", x = "T_degC", controls = c("ChlorA", "O2Sat"),
         data = bottles, progress_bar = FALSE, parallel = FALSE)
sca_report(s)

result <- sca_test(y = "Salnty", x = "T_degC", controls = c("ChlorA", "O2Sat"),
                   data = bottles, n_permutations = 50, seed = 1,
                   progress_bar = FALSE)
sca_report(result)
```

sca_table *Render a specification curve result as a publication table*

Description

‘sca_table()‘ turns the output of [sca()] or [sca_test()] into a compact results block suitable for a manuscript: the number of specifications, the median estimate, the share of significant specifications, and (for [sca_test()]) the joint-inference statistics and their permutation p-values.

The default ‘format = "data.frame"‘ returns a small two-column (‘label‘ / ‘value‘) data frame with a ‘print‘ method and adds no dependencies. The other formats render that frame with an optional package: ‘"markdown"‘ / ‘"latex"‘ use ‘knitr‘, and ‘"gt"‘, ‘"kableExtra"‘, and ‘"flextable"‘ use the package of the same name (install it first). A [tidy()]/[glance()]-aware tool such as ‘modelsummary‘ can consume the object directly without ‘sca_table()‘.

Usage

```
sca_table(
  x,
  format = c("data.frame", "markdown", "latex", "gt", "kableExtra", "flextable"),
  digits = 3,
  ...
)
```

Arguments

x	An object returned by [sca()] or [sca_test()].
format	The output format: one of ‘"data.frame"‘ (default), ‘"markdown"‘, ‘"latex"‘, ‘"gt"‘, ‘"kableExtra"‘, or ‘"flextable"‘.
digits	Number of significant digits for estimates. Defaults to ‘3‘.
...	Ignored.

Value

For ‘format = "data.frame"‘, a data frame of class ‘"sca_table"‘; otherwise an object of the corresponding rendering package.

See Also

[sca_report()] for a prose summary; [tidy()]/[glance()] for tidy data frames.

Examples

```
s <- sca(y = "Salnty", x = "T_degC", controls = c("ChlorA", "O2Sat"),
         data = bottles, progress_bar = FALSE, parallel = FALSE)
sca_table(s)

result <- sca_test(y = "Salnty", x = "T_degC", controls = c("ChlorA", "O2Sat"),
```

```
                    data = bottles, n_permutations = 50, seed = 1,
                    progress_bar = FALSE)
sca_table(result)
```

---

sca_test *Joint-inference test for a specification curve*

Description

'sca_test()' performs the permutation-based joint-inference test of Simonsohn, Simmons, and Nelson (2020). It tests the sharp null hypothesis that the focal independent variable 'x' has no effect on 'y' in any specification. The focal variable is repeatedly permuted (shuffled) and the entire specification curve is re-estimated on each permuted data set, building a null distribution for curve-level summary statistics. The observed statistics are then compared to that null distribution to obtain permutation p-values.

The permutation is *blocked within the first fixed effect* when 'fixed_effects' are supplied, because exchangeability of 'x' only holds within fixed-effect groups; otherwise it is a global permutation.

Usage

```
sca_test(
  y,
  x,
  controls,
  data,
  weights = NULL,
  family = "linear",
  link = NULL,
  fixed_effects = NULL,
  n_permutations = 500,
  test_stats = c("median", "share_significant", "stouffer"),
  direction = "two.sided",
  alpha = 0.05,
  sca_data = NULL,
  keep_curves = FALSE,
  null_type = c("shuffle_x", "freedman_lane", "residual_bootstrap"),
  common_sample = FALSE,
  parallel = FALSE,
  workers = 2,
  seed = NULL,
  progress_bar = TRUE
)
```

Arguments

y	A string with the dependent variable column name, or a two-sided formula specifying the whole model (‘y ~ x + control1 + control2 | fixed_effect‘), exactly as accepted by [sca()]. When a formula is supplied ‘data‘ may be passed positionally.
x	A string with the focal independent variable column name.
controls	A vector of strings with the control variable column names.
data	A data frame containing the variables.
weights	Optional string naming a weights column in ‘data‘. Weights are held fixed to their row and never permuted; this assumes the weights are exogenous to ‘x‘.
family	A string giving the model family, as in [sca()]. Defaults to ‘"linear"‘.
link	A string giving the link function, as in [sca()]. Defaults to ‘NULL‘.
fixed_effects	A string (or vector) naming fixed-effect variables, as in [sca()]. When supplied, ‘x‘ is permuted within levels of the first fixed effect.
n_permutations	An integer number of permutations used to build the null distribution. Defaults to ‘500‘. The smallest attainable p-value is ‘1 / (n_permutations + 1)‘.
test_stats	A character vector of statistics to compute. Any of ‘"median"‘ (median focal coefficient across specifications), ‘"share_significant"‘ (share of specifications statistically significant in the reference direction), ‘"stouffer"‘ (Stouffer’s combined Z over the per-specification p-values), and the descriptive ‘"share_sign"‘ (share of specifications with the reference sign). Defaults to ‘c("median", "share_significant", "stouffer")‘.
direction	One of ‘"two.sided"‘ (default), ‘"positive"‘, or ‘"negative"‘, giving the a-priori predicted direction of the effect. The package never chooses the direction from the data: with ‘"two.sided"‘ the tests are direction-agnostic; use ‘"positive"‘/‘"negative"‘ only for a genuinely a-priori prediction.
alpha	The significance threshold used by ‘"share_significant"‘. Defaults to ‘0.05‘.
sca_data	Optional data frame previously returned by [sca()] for the *observed* model, used purely to skip recomputing the observed curve. The null distribution is always recomputed from ‘data‘, so ‘data‘ remains required. Its control indicator columns must be consistent with ‘controls‘. Ignored (with a warning) for ‘null_type = "freedman_lane"‘ or ‘"residual_bootstrap"‘, where the observed curve is recomputed on the forced common sample so it matches the null.
keep_curves	A boolean indicating whether to retain, for every specification, the focal coefficient from each permuted curve (aligned across permutations by specification, not row order). Required by [plot_sca_test_specs()]; increases the size of the returned object. Defaults to ‘FALSE‘.
null_type	How the null is generated. One of: **‘"shuffle_x"‘** (default) Simonsohn-Simmons-Nelson sharp-null permutation: shuffle the focal variable (blocked within the first fixed effect). Appropriate for experimental / as-if-randomly-assigned ‘x‘. It breaks ‘x‘’s correlation with the controls, so it is miscalibrated (anti-conservative) under collinearity – the observational case.

‘"freedman_lane"‘ Freedman-Lane (1983) partial permutation. A reduced model ‘y ~ controls‘ (the full control superset, ‘x‘ omitted, fixed effects retained) is fit once; its residuals are permuted (blocked within the first fixed effect) and added back to its fitted values to form a null response, on which the whole curve is refit. This preserves ‘x‘’s partial correlation with the controls. It tests the sharp null of no partial effect of ‘x‘ given the superset controls, so under-controlled specifications may have non-zero null centres by design.

‘"residual_bootstrap"‘ The SSN (2020) observational scheme: impose the null on the response (‘y - betahat * x‘, with ‘betahat‘ from the full control-superset specification), then resample rows with replacement and refit. A null-imposed case bootstrap (nearly equivalent to Flachaire 1999); robust to heteroskedasticity but its p-values carry extra Monte-Carlo variability.

The ‘"freedman_lane"‘ and ‘"residual_bootstrap"‘ nulls are defined for ‘family = "linear"‘ only (including fixed effects) and force ‘common_sample = TRUE‘.

`common_sample` A boolean passed through to [sca()]: fit every specification on the rows complete across all model variables. Defaults to ‘FALSE‘; forced to ‘TRUE‘ for the ‘"freedman_lane"‘ and ‘"residual_bootstrap"‘ nulls.

`parallel` A boolean indicating whether to parallelise the permutations. Defaults to ‘FALSE‘. The inner [sca()] call is always run serially to avoid nested parallelism.

`workers` An integer number of parallel workers. Defaults to ‘2‘.

`seed` Optional integer seed. When supplied, results are reproducible and identical for the serial and parallel paths (the permutations are generated up front).

`progress_bar` A boolean indicating whether to show a progress bar for the permutations. Defaults to ‘TRUE‘.

Value

An object of class ‘"sca_test"‘: a list with elements ‘observed‘ (a named vector of observed statistics), ‘null_distribution‘ (a data frame with one row per usable permutation and one column per statistic, plus ‘n_valid‘), ‘p_values‘ (a named vector of permutation p-values), ‘params‘ (a list of run metadata), and, when ‘keep_curves = TRUE‘, ‘null_curves‘ (a list with ‘spec‘, a data frame of each specification’s key and observed coefficient, and ‘null_coef‘, a specifications-by-permutations matrix of the permuted focal coefficients). When ‘keep_curves = TRUE‘ *and* the null is confound-preserving (‘null_type = "freedman_lane"‘ or ‘"residual_bootstrap"‘), ‘null_curves‘ also gains ‘fwer‘, the per-specification family-wise-error-rate-adjusted p-values attached automatically by [sca_minp()] (see there for its structure).

References

Simonsohn, U., Simmons, J. P., & Nelson, L. D. (2020). Specification curve analysis. *Nature Human Behaviour*, 4, 1208-1214. doi:10.1038/s415620200912z

Freedman, D., & Lane, D. (1983). A nonstochastic interpretation of reported significance levels. *Journal of Business & Economic Statistics*, 1(4), 292-298.

Winkler, A. M., Ridgway, G. R., Webster, M. A., Smith, S. M., & Nichols, T. E. (2014). Permutation inference for the general linear model. *NeuroImage*, 92, 381-397.

See Also

[plot_sca_test()] to visualise the null distributions of the test statistics, and [plot_sca_test_specs()] for the per-specification null-band plot (requires ‘keep_curves = TRUE‘).

Examples

```
# Test whether temperature robustly predicts salinity across specifications.
result <- sca_test(y = "Salnty", x = "T_degC",
                   controls = c("ChlorA", "O2Sat"),
                   data = bottles, n_permutations = 100, progress_bar = FALSE)
result
```

`sca_tidiers` *Tidy and glance methods for speccurvieR objects*

Description

[generics::tidy()] and [generics::glance()] methods for the data frame returned by [sca()] (class ‘"sca"‘) and the object returned by [sca_test()]. These are the same generics that ‘broom::tidy()‘ / ‘broom::glance()‘ and ‘modelsummary‘ dispatch on, so the results can feed those tools.

Note that the tidied column names follow the broom convention (‘estimate‘ / ‘std.error‘ / ‘p.value‘), which differs from the raw column names of [sca()] output (‘coef‘ / ‘se‘ / ‘p‘). For an ‘sca‘ curve every row shares the same focal ‘term‘, so a tool that keys on ‘term‘ (such as ‘modelsummary‘) needs each row distinguished (e.g. by ‘spec_id‘); the per-statistic rows of ‘tidy.sca_test()‘ are already distinct. The ‘controls‘ column lists the model terms included (so an interaction control ‘a*b‘ appears as its expanded terms), which can differ from the user-facing control count in ‘glance()$n_controls‘.

Usage

```
## S3 method for class 'sca'
tidy(x, ...)

## S3 method for class 'sca'
glance(x, ...)

## S3 method for class 'sca_test'
tidy(x, ...)

## S3 method for class 'sca_test'
glance(x, ...)

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

```
  ...,
  what = c("summary", "null", "params", "specs")
)
```

Arguments

x	An object returned by [sca()] (class '"sca"') or by [sca_test()].
...	Ignored.
row.names, optional	Passed through from the [as.data.frame()] generic; ignored.
what	For 'as.data.frame.sca_test()', which component to return: '"summary"' (default; observed statistics and p-values), '"null"' (the raw null distribution), '"params"' (the run metadata), or '"specs"' (the per-specification family-wise-error-rate-adjusted p-values, available only when the result was computed with 'keep_curves = TRUE' and a confound-preserving null; see [sca_minp()]).

Value

A data frame. For 'tidy()', one row per specification ('sca') or per test statistic ('sca_test'); for 'glance()', a one-row summary.

See Also

[sca_table()] and [sca_report()] for formatted reporting.

Examples

```
s <- sca(y = "Salnty", x = "T_degC", controls = c("ChlorA", "O2Sat"),
         data = bottles, progress_bar = FALSE, parallel = FALSE)
tidy(s)
glance(s)
```

---

sca_variance *Decompose the variance of the specification curve by analytic choice*

Description

'sca_variance()' quantifies how much of the variation in the focal coefficient across a specification curve is attributable to each analytic choice – here, the inclusion or exclusion of each control variable. It is the descriptive complement to [sca_test()]: where the joint-inference test asks whether the curve as a whole is more extreme than chance, the variance decomposition asks which modelling choices *drive* the spread of estimates.

The focal coefficient (the 'coef' column of [sca()] output) is regressed on the 0/1 control-inclusion indicators, and the variance explained is partitioned across controls. By default this uses the LMG (Shapley-value) decomposition of R-squared, whose shares are order-invariant and sum exactly to the model R-squared; the remaining '1 - R^2' is reported as a '"Residual"' row capturing interactions among choices and unexplained variation.

Usage

```
sca_variance(
  sca_data,
  estimate = "coef",
  method = c("lmg", "type2"),
  residual = TRUE
)
```

Arguments

sca_data	A data frame returned by [sca()].
estimate	A string naming the numeric column to decompose. Defaults to ‘"coef"‘ (the focal coefficient).
method	One of ‘"lmg"‘ (default; LMG/Shapley decomposition of R-squared) or ‘"type2"‘ (normalised Type II partial sums of squares). ‘"shapley"‘ is an alias for ‘"lmg"‘.
residual	A boolean indicating whether to append a ‘"Residual"‘ row for the unexplained (and choice-interaction) variance. Defaults to ‘TRUE‘. When ‘FALSE‘, the control percentages are rescaled to sum to 100.

Details

Method. With ‘method = "lmg"‘ (default) each control’s share is the Shapley value of R-squared: its average marginal contribution to R-squared over all orderings of the controls. These shares are non-negative, order-invariant, and sum to the full-model R-squared, so the reported percentages (controls plus ‘"Residual"‘) sum to 100. ‘method = "type2"‘ instead reports each control’s "drop-one" partial sum of squares, normalised to sum to 100; because partial sums of squares do not add up under a non-orthogonal design, those percentages are unique contributions rather than an exact partition. (The regression is purely additive – each control, including an interaction control, enters as a single 0/1 indicator with no product terms – so Type II and Type III partial sums of squares coincide; the ‘"type2"‘ label is kept only for familiarity.) ‘"shapley"‘ is accepted as an alias for ‘"lmg"‘. The LMG cost grows as 2 to the power of the number of varying controls (like ‘sca()‘’s own combinatorial growth) and is capped at 20 controls; for larger sets use ‘"type2"‘, which is linear.

Choice unit. The decomposition is over *controls* (one binary in/out choice each), read from the indicator columns of [sca()] output. An interaction control such as ‘"a*b"‘ is a single choice (its column ‘"a:b"‘). Within one [sca()] call the model family, fixed effects, and focal variable are constant, so only control choices (plus ‘"Residual"‘) appear.

Relationship to specr. This is analogous to ‘specr::icc_specs()‘ / ‘specr::plot_variance()‘ and returns a comparable choice-by-percent table and bar chart. It deliberately differs in method: ‘icc_specs()‘ fits a random-effects model and reads intraclass-correlation-based variance components, which treats each analytic choice as one grouping factor with many levels. speccurvieR’s only enumerated choice is each control’s binary status, for which two-level random-effect variance components are unreliable, so an LMG R-squared decomposition is used instead and no ‘lme4‘ dependency is added. The output is intentionally *not* labelled "ICC".

Caveats. The shares are descriptive, not inferential: they summarise an already-estimated set of point estimates and carry no sampling-uncertainty interpretation. A share measures how much a control’s in/out status *moves* the focal coefficient, not the control’s predictive importance and

not the direction of any shift. For glm or fixed-effects curves the coefficient is on the model's native (e.g. log-odds) scale, so shares are not comparable across families. At least two specifications with a varying control are required; with exactly two controls the three specifications are saturated by the two main effects, so the residual is zero. If an interaction control is supplied *together with* its component main-effect controls, the controls are not cleanly separable and the shares should be interpreted with care.

Value

A data frame with one row per control (and a final ‘"Residual"‘ row when ‘residual = TRUE‘), sorted by descending share, with columns: ‘choice‘ (the control name, or ‘"Residual"‘), ‘variance‘ (the portion of the focal coefficient's variance attributed to that choice), and ‘percent‘ (its share as a percentage; the column sums to 100).

See Also

[plot_variance()] to visualise the decomposition; [sca_test()] for the joint-inference test.

Examples

```
s <- sca(y = "Salnty", x = "T_degC",
         controls = c("ChlorA", "O2Sat", "NO2uM"),
         data = bottles, progress_bar = FALSE, parallel = FALSE)
sca_variance(s)
sca_variance(s, method = "type2", residual = FALSE)
```

---

se_compare *Compare different kinds of standard errors*

Description

se_compare() takes in a regression formula (with or without fixed effects), data, and the types of standard errors desired, including clustered, heteroskedasticity-consistent, and bootstrapped. It then returns a data frame with coefficient and standard error estimates for easy comparison and plotting.

Usage

```
se_compare(
  formula,
  data,
  weights = NULL,
  family = "linear",
  link = NULL,
  types = "all",
  cluster = NULL,
  clustered_only = FALSE,
  fixed_effects_only = FALSE,
  boot_samples = NULL,
```

```
  boot_sample_size = NULL,
  ...
)
```

Arguments

formula	A regression formula, with or without fixed effects, given either as a string (‘"y ~ x | fe"‘) or as a formula object (‘y ~ x | fe‘).
data	A data frame containing the variables provided in ‘formula‘ and any clustering variables passed to ‘cluster‘.
weights	Optional string with the column name in ‘data‘ that contains weights.
family	A string indicating the family of models to be used. Defaults to "linear" for OLS regression but supports all families supported by ‘glm()‘. When a non-linear family is supplied the models are estimated with ‘glm()‘; fixed effects are not supported in that case and are ignored with a warning.
link	A string specifying the link function to be used for the model. Defaults to ‘NULL‘, using OLS regression via ‘lm()‘ (or ‘fixest::feols()‘ when fixed effects are supplied). For a non-linear ‘family‘ the canonical link is used when ‘link‘ is ‘NULL‘. Supports all link functions supported by the family parameter of ‘glm()‘.
types	A string or vector of strings specifying what types of standard errors are desired. Defaults to "all". The following types are supported for non-fixed effects models: With clustering: "HC0, "HC1", "HC2", "HC3". Without clustering: "iid" (i.e. normal standard errors), "HC0, "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5", "bootstrapped". The following types are supported for fixed effects models: With clustering: "CL_FE" (standard errors clustered by the first fixed effect), if clusters are supplied then the conventional clustered standard errors from ‘feols()‘ are estimated for each clustering specification. Two-way (and multiway) clustering is supported; see the ‘cluster‘ argument. Without clustering: "HC0, "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5", "bootstrapped".
cluster	Variables in ‘data‘ to cluster the standard errors on. Either a character vector, in which case each element is used for a separate **one-way** clustering, or a list of character vectors, in which case each element is clustered on **jointly** (one-way when the element names a single variable, two-way or higher when it names several). For example ‘cluster = list("a", "b", c("a", "b"))‘ produces one-way SEs clustered by ‘a‘, one-way by ‘b‘, and two-way clustered by ‘a‘ and ‘b‘. Multiway columns are labelled with the clustering dimensions joined by ‘_BY_‘ (e.g. ‘"HC1_a_BY_b"‘, or ‘"CL_a_BY_b_FE"‘ for a fixed-effects model); the dimensions are sorted, so the label is the same regardless of the order they are listed in. Unknown variables are dropped with a warning. Defaults to ‘NULL‘ (no clustering).
clustered_only	A boolean indicating whether only standard errors with clustering should be estimated, defaults to ‘FALSE‘.

`fixed_effects_only`
: A boolean indicating whether only standard errors for fixed effects models should be estimated, defaults to 'FALSE'.

`boot_samples`
: An integer or vector of integers indicating how many times the model should be estimated with a random subset of the data. If a vector then every combination of 'boot_samples' and 'boot_sample_size' are estimated.

`boot_sample_size`
: An integer or vector of integers indicating how many observations are in each random subset of the data. If a vector then every combination of 'boot_samples' and 'boot_sample_size' are estimated.

`...`
: Deprecated camelCase arguments ('clusteredOnly', 'fixedEffectsOnly', 'bootSamples', 'bootSampleSize'); use the snake_case equivalents instead.

Value

A data frame where row represents an independent variable in the model and each column a type of standard error. Coefficient estimates for each variable are also included (column '"estimate"' for non-fixed effects model and column '"estimate_FE"' for fixed effects models). Columns are automatically named to specify the standard error type.

Some examples:

"iid" = normal standard errors, i.e. assuming homoskedasticity

"CL_FE" = standard errors clustered by the first fixed effect

"bootstrap_k8n300_FE" = bootstrapped standard errors for a fixed effects model where 'boot_samples = 8' and 'boot_sample_size = 300'

"CL_Depth_ID_FE" = standard errors clustered by the variable "Depth_ID" for a model with fixed effects

"HC0_Sta_ID" = HC0 standard errors clustered by the variable "Sta_ID"

"HC0_Depth_ID_BY_Sta_ID" = HC0 standard errors two-way clustered by "Depth_ID" and "Sta_ID"

Note: for fixed effects models the "(Intercept)" row will be all 'NA' because the intercept is not reported by 'feols()' when fixed effects are present.

Examples

```
se_compare(formula = "Salnty ~ T_degC + ChlorA + O2Sat | Sta_ID",
           data = bottles, types = "all", cluster = c("Depth_ID", "Sta_ID"),
           fixed_effects_only = FALSE, boot_samples=c(4, 8, 10),
           boot_sample_size=c(300, 500))

se_compare(formula = "Salnty ~ T_degC + ChlorA + O2Sat", data = bottles,
           types = "bootstrapped", boot_samples = c(8, 10),
           boot_sample_size = c(300, 500))

se_compare(formula = "Salnty ~ T_degC + ChlorA", data = bottles,
           types = c("HC0", "HC1", "HC3"))

# Two-way (and multiway) clustering: pass a list, where each element names
# the dimensions to cluster on jointly. Here: one-way by Sta_ID, one-way by
```

```
# Depth_ID, and two-way by both.
se_compare(formula = "Salnty ~ T_degC + ChlorA", data = bottles,
           types = "HC1",
           cluster = list("Sta_ID", "Depth_ID", c("Sta_ID", "Depth_ID")))

# Logistic regression: compare standard error types for a binary outcome.
bottles$saline <- as.integer(bottles$Salnty >
                               stats::median(bottles$Salnty, na.rm = TRUE))
se_compare(formula = "saline ~ T_degC + ChlorA", data = bottles,
           family = "binomial", types = c("iid", "HC0", "HC3"))
```

---

theme_sca *A clean, consistent ggplot2 theme for speccurvieR plots.*

Description

'theme_sca()' is the shared theme applied by the package's plotting functions. It is exported so the same look can be reused or tweaked when customising the plots they return.

Usage

```
theme_sca(
  base_size = 11,
  base_family = getOption("speccurvieR.base_family", "")
)
```

Arguments

base_size	Base font size, passed to [ggplot2::theme_minimal()]. Defaults to '11'.
base_family	Base font family, passed to [ggplot2::theme_minimal()]. Defaults to the value of the 'speccurvieR.base_family' option, or '""' (the graphics device's default font) if that option is unset, so plots stay portable across machines. Set the option (e.g. 'options(speccurvieR.base_family = "Roboto")') or pass a family directly to use a specific font.

Value

A ggplot2 theme object.

Examples

```
library(ggplot2)
ggplot(bottles, aes(T_degC, Salnty)) + geom_point() + theme_sca();
```

Index