

Package ‘int3ract’

September 1, 2026

Type Package

Title Johnson-Neyman Analysis of Two- and Three-Way Interactions

Version 2.0.0

Description Reports and plots the conditional effect of each variable involved in a multiplicative interaction across the range of its moderators, together with the region over which that effect is distinguishable from zero. Extends the classic framework of Johnson and Neyman (1936) and Johnson and Fay (1950) <[doi:10.1007/BF02288864](https://doi.org/10.1007/BF02288864)> to three-way interactions and to Bayesian models. The single entry point JN() dispatches on the fitted object, with methods for lm()/glm() models, 'lme4' models, 'RSiena' and 'multiSiena' results, and matrices of posterior draws; support for further model classes is added by writing one jn_input() method. Results are classed objects with print(), summary() and plot() methods, and the figures carry data-density panels showing how much empirical support each part of the moderator range has. A detailed introduction can be found in Krause (2026) <[doi:10.48550/arXiv.2604.22051](https://doi.org/10.48550/arXiv.2604.22051)>.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.4.0)

Imports ggplot2, ggpattern, patchwork

Suggests knitr, lme4, MCMCpack, rmarkdown, RSiena, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

URL <https://github.com/RWKrause/int3ract>

BugReports <https://github.com/RWKrause/int3ract/issues>

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Robert W. Krause [aut, cre, cph]

Maintainer Robert W. Krause <robert.w.krause@mailbox.org>

Repository CRAN

Date/Publication 2026-09-01 13:10:12 UTC

Contents

as.data.frame.JN	2
JN	3
jn_input	7
jn_plots	11
jn_regions	12
jn_save	13
jn_style	14
plot.JN	15
print.JN	17
print.JN_list	17
summary.JN	18
Index	20

as.data.frame.JN	<i>Extract the conditional effects of a Johnson-Neyman analysis</i>
------------------	---

Description

as.data.frame() returns the full grid: one row per focal variable and moderator value, with the conditional estimate, its uncertainty, and whether it is significant. coef() and vcov() return the model parameters the analysis was built from.

Usage

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

## S3 method for class 'JN'
coef(object, ...)

## S3 method for class 'JN'
vcov(object, ...)
```

Arguments

- x a JN object.
- row.names, optional ignored, for compatibility with the generic.
- ... ignored.
- object a JN object.

Value

A data frame, respectively a named numeric vector and a matrix.

Examples

```
set.seed(1)
dat <- data.frame(x = rnorm(100), z = rnorm(100))
dat$y <- dat$x + 0.5 * dat$x * dat$z + rnorm(100)
jn <- JN(lm(y ~ x * z, data = dat), theta_1 = "x", theta_2 = "z",
        range_size = 5)
head(as.data.frame(jn))
coef(jn)
```

JN

Johnson-Neyman analysis of a two- or three-way interaction

Description

Computes the conditional effect of each variable involved in a multiplicative interaction across the range of its moderators, together with the regions over which that effect is statistically distinguishable from zero. Two-way interactions give the classic Johnson-Neyman analysis; three-way interactions give its extension over a two-dimensional moderator grid, referred to here as JN3.

Usage

```
JN(object, ...)

## Default S3 method:
JN(
  object,
  theta_1,
  theta_2,
  theta_3 = NULL,
  theta_1_vals = NULL,
  theta_2_vals = NULL,
  theta_3_vals = NULL,
  support = NULL,
  alpha = 0.05,
  control_fdr = FALSE,
  range_size = NULL,
  thresholds = NULL,
  ...
)

## S3 method for class 'jn_input'
JN(
  object,
  theta_1_vals = NULL,
  theta_2_vals = NULL,
  theta_3_vals = NULL,
```

```
    alpha = 0.05,
    control_fdr = FALSE,
    range_size = NULL,
    thresholds = NULL,
    ...
)

## S3 method for class 'merMod'
JN(
  object,
  theta_1,
  theta_2,
  theta_3 = NULL,
  theta_1_vals = NULL,
  theta_2_vals = NULL,
  theta_3_vals = NULL,
  support = NULL,
  alpha = 0.05,
  control_fdr = FALSE,
  range_size = NULL,
  thresholds = NULL,
  fixed_only = TRUE,
  group_var = NULL,
  ...
)

## S3 method for class 'lmerModLmerTest'
JN(
  object,
  theta_1,
  theta_2,
  theta_3 = NULL,
  theta_1_vals = NULL,
  theta_2_vals = NULL,
  theta_3_vals = NULL,
  support = NULL,
  alpha = 0.05,
  control_fdr = FALSE,
  range_size = NULL,
  thresholds = NULL,
  fixed_only = TRUE,
  group_var = NULL,
  ...
)

## S3 method for class 'sienaBayesFit'
JN(
  object,
```

```

    theta_1,
    theta_2,
    theta_3 = NULL,
    theta_1_vals = NULL,
    theta_2_vals = NULL,
    theta_3_vals = NULL,
    support = NULL,
    alpha = 0.05,
    control_fdr = FALSE,
    range_size = NULL,
    thresholds = NULL,
    hyper_only = TRUE,
    ...
)

## S3 method for class 'multiSiena'
JN(
  object,
  theta_1,
  theta_2,
  theta_3 = NULL,
  theta_1_vals = NULL,
  theta_2_vals = NULL,
  theta_3_vals = NULL,
  support = NULL,
  alpha = 0.05,
  control_fdr = FALSE,
  range_size = NULL,
  thresholds = NULL,
  hyper_only = TRUE,
  ...
)

```

Arguments

<code>object</code>	a fitted model, a matrix of draws, or a <code>jn_input</code> object built with <code>jn_wald</code> or <code>jn_posterior</code> .
<code>...</code>	passed to the <code>jn_input</code> method for object.
<code>theta_1, theta_2</code>	name (or, for <code>sienaFit</code> and <code>sienaBayesFit</code> , integer position) of the first and second variable involved in the interaction.
<code>theta_3</code>	the third variable, or <code>NULL</code> (default) for a two-way interaction.
<code>theta_1_vals, theta_2_vals, theta_3_vals</code>	the values each variable takes when it acts as a moderator. A vector of length two is read as a range and filled in with <code>range_size</code> points; any other length is used verbatim. Defaults to the observed range where the model carries its data, and is required otherwise.

support	optional list of length 2 or 3 with the observed values of each variable, used for the data-density panels of plot.JN . Extracted automatically from <code>lm</code> , <code>glm</code> and lme4 models.
alpha	significance level for the Wald path, and the basis of the default thresholds for the posterior path. Default 0.05.
control_fdr	control the false discovery rate across the moderator grid with the Benjamini-Hochberg procedure? Default FALSE.
range_size	number of moderator values used when a range is expanded. Defaults to 1000 for two-way and 50 for three-way Wald analyses, and to 13 and 50 respectively for posterior analyses, where each value becomes its own conditional posterior.
thresholds	two Bayesian p values bounding the region treated as inconclusive; the conditional effect counts as significant outside them. Defaults to <code>c(alpha/2, 1 - alpha/2)</code> .
fixed_only	analyse only the fixed effects? When FALSE, a separate analysis is returned for every level of the grouping factor, using group-specific coefficients (fixed effects plus conditional modes).
group_var	name of the grouping factor to use when <code>fixed_only = FALSE</code> . Defaults to the first one.
hyper_only	analyse only the population-level parameter? When FALSE, a separate analysis is returned for each group as well, in the same way that <code>fixed_only = FALSE</code> works for lme4 models. Group-level analyses are only produced when at least one of the parameters involved actually varies between groups; when none does, there is nothing for them to show and the population-level analysis is returned alone.

Details

`JN()` dispatches on object. Fitted models carrying point estimates and a covariance matrix (`lm`, `glm`, `lmerMod`, `sienaFit`, ...) are analysed with Wald z tests; objects carrying draws (a posterior matrix, `mcmc`, `sienaBayesFit`, ...) are analysed as conditional posterior distributions with Bayesian p values. Both paths return an object of class `JN` with the same set of methods.

Support for a model class that is not listed above is added by writing a single `jn_input` method for it; `JN()` itself needs no change. See [jn_input](#).

Value

An object of class `JN`, with methods [print](#), [summary](#), [plot](#) and `as.data.frame`. Analyses producing one result per group return a `JN_list`, which carries the same methods.

References

- Johnson PO, Neyman J (1936). "Tests of Certain Linear Hypotheses and Their Application to Some Educational Problems." *Statistical Research Memoirs*, 1, 57-93.
- Bauer DJ, Curran PJ (2005). "Probing Interactions in Fixed and Multilevel Regression: Inferential and Graphical Techniques." *Multivariate Behavioral Research*, 40(3), 373-400. doi:10.1207/s15327906mbr4003_5

Examples

```

set.seed(1402)
dat <- data.frame(x = rnorm(100), z = rnorm(100), w = rnorm(100))
dat$y <- dat$x + 0.5 * dat$z - 0.5 * dat$w +
  0.5 * dat$x * dat$z * dat$w + rnorm(100, sd = 4)

## two-way
fit2 <- lm(y ~ x * z, data = dat)
jn2 <- JN(fit2, theta_1 = "x", theta_2 = "z")
jn2
summary(jn2)
plot(jn2, which = "x")

## three-way
fit3 <- lm(y ~ x * z * w, data = dat)
jn3 <- JN(fit3, theta_1 = "x", theta_2 = "z", theta_3 = "w",
  range_size = 20)
summary(jn3)

## a matrix of posterior draws takes the same route
post <- cbind(x = rnorm(500, 0.5, 0.2), z = rnorm(500, -0.3, 0.2),
  `x:z` = rnorm(500, 0.4, 0.2))
JN(post, theta_1 = "x", theta_2 = "z",
  theta_1_vals = seq(-2, 2, 1), theta_2_vals = seq(-2, 2, 1))

```

jn_input

Carriers for Johnson-Neyman input

Description

`jn_input()` is the generic that turns a fitted model into the normalized representation used by [JN](#). It is the extension point of the package: to support a model class that **int3ract** does not know about, write a `jn_input()` method for it that returns either a `jn_wald` or a `jn_posterior` object built with the constructors documented here.

Usage

```

jn_input(object, ...)

jn_wald(coefficients, vcov, labels, ranges = NULL, support = NULL)

jn_posterior(draws, labels, ranges = NULL, support = NULL)

## Default S3 method:
jn_input(object, ...)

## S3 method for class 'lm'

```

```
jn_input(object, theta_1, theta_2, theta_3 = NULL, ...)
```

```
## S3 method for class 'merMod'
```

```
jn_input(object, theta_1, theta_2, theta_3 = NULL, ...)
```

```
## S3 method for class 'lmerModLmerTest'
```

```
jn_input(object, theta_1, theta_2, theta_3 = NULL, ...)
```

```
## S3 method for class 'sienaFit'
```

```
jn_input(  
  object,  
  theta_1,  
  theta_2,  
  theta_3 = NULL,  
  theta_int_12 = NULL,  
  theta_int_13 = NULL,  
  theta_int_23 = NULL,  
  theta_int_123 = NULL,  
  ranges = NULL,  
  support = NULL,  
  ...  
)
```

```
## S3 method for class 'matrix'
```

```
jn_input(  
  object,  
  theta_1,  
  theta_2,  
  theta_3 = NULL,  
  theta_int_12 = NULL,  
  theta_int_13 = NULL,  
  theta_int_23 = NULL,  
  theta_int_123 = NULL,  
  burn_in = 0,  
  thin = 1,  
  ranges = NULL,  
  support = NULL,  
  ...  
)
```

```
## S3 method for class 'mcmc'
```

```
jn_input(object, ...)
```

```
## S3 method for class 'mcmc.list'
```

```
jn_input(object, ...)
```

```
## S3 method for class 'data.frame'
```

```
jn_input(object, ...)
```

```

## S3 method for class 'sienaBayesFit'
jn_input(
  object,
  theta_1,
  theta_2,
  theta_3 = NULL,
  theta_int_12 = NULL,
  theta_int_13 = NULL,
  theta_int_23 = NULL,
  theta_int_123 = NULL,
  burn_in = NULL,
  thin = 1,
  group = NULL,
  ranges = NULL,
  support = NULL,
  ...
)

## S3 method for class 'multiSiena'
jn_input(
  object,
  theta_1,
  theta_2,
  theta_3 = NULL,
  theta_int_12 = NULL,
  theta_int_13 = NULL,
  theta_int_23 = NULL,
  theta_int_123 = NULL,
  burn_in = NULL,
  thin = 1,
  group = NULL,
  ranges = NULL,
  support = NULL,
  ...
)

```

Arguments

object	a fitted model object.
...	passed to methods; ignored by the constructors.
coefficients	numeric vector of length 3 (two-way) or 7 (three-way), in the canonical order (theta_1, theta_2, theta_1:theta_2) or (theta_1, theta_2, theta_3, theta_1:theta_2, theta_1:theta_3, theta_2:theta_3, theta_1:theta_2:theta_3).
vcov	covariance matrix of coefficients, in the same order.
labels	character vector of length 2 or 3 with display names for the focal variables.
ranges	list of length 2 or 3; the range over which each variable is evaluated when it acts as a moderator. Elements may be NULL if they are to be derived from the data.

support	list of length 2 or 3 holding the observed values of each variable, or NULL. Used to draw the data-density panels of <code>plot.JN</code> , which show how much empirical support each part of the moderator range actually has. Supplying it is optional but strongly recommended: a region of significance covering moderator values that were barely observed is not evidence of much.
draws	matrix of draws with one row per iteration and one column per parameter, in the same canonical order.
theta_1, theta_2	name (or, for <code>sienaFit</code> and <code>sienaBayesFit</code> , integer position) of the first and second variable involved in the interaction.
theta_3	the third variable, or NULL (default) for a two-way interaction.
theta_int_12, theta_int_13, theta_int_23, theta_int_123	positions of the interaction parameters. Required for <code>sienaFit</code> and <code>sienaBayesFit</code> input, where effects are addressed by position rather than by name; optional for matrices of draws whose columns are named.
burn_in	number of initial draws to discard.
thin	thinning interval applied after burn-in.
group	for <code>sienaBayesFit</code> input, the index of the group whose draws to extract. NULL (the default) uses the hyper-parameter, or the fixed parameter when none of the effects varies between groups.

Details

`jn_wald()` carries point estimates and their covariance matrix, and leads to Wald z tests of the conditional effect. `jn_posterior()` carries a matrix of draws, and leads to conditional posterior distributions and Bayesian p values. Which of the two a method returns is what makes an analysis frequentist or Bayesian: the distinction is a property of the input, not of the function the user calls.

Value

An object of class `c("jn_wald", "jn_input")` or `c("jn_posterior", "jn_input")`.

See Also

[JN](#) for the analysis itself.

Examples

```
set.seed(1)
dat <- data.frame(x = rnorm(100), z = rnorm(100))
dat$y <- dat$x + 0.5 * dat$x * dat$z + rnorm(100)
fit <- lm(y ~ x * z, data = dat)

## what JN() does internally
inp <- jn_input(fit, theta_1 = "x", theta_2 = "z")
inp

## the same thing built by hand, for a model class without a method
```

```

idx <- c("x", "z", "x:z")
jn_wald(coefficients = coef(fit)[idx],
        vcov         = vcov(fit)[idx, idx],
        labels       = c("x", "z"),
        ranges       = list(range(dat$x), range(dat$z)),
        support      = list(dat$x, dat$z))

```

jn_plots

*Every figure of a Johnson-Neyman analysis***Description**

Returns all of the figures at once, named by focal variable, without drawing any of them. [plot.JN](#) draws instead; use this when the figures are wanted as objects, to arrange, modify or save.

Usage

```
jn_plots(x, type = NULL, style = jn_style(), ...)
```

Arguments

x	a JN object.
type	for posterior analyses, which quantity to draw; see plot.JN . Ignored for three-way posterior analyses, where both are returned.
style	appearance settings from jn_style .
...	ignored.

Value

A named list of figures. For three-way posterior analyses the list has two elements, `post_mean` and `bayes_p`, each holding one figure per focal variable.

Examples

```

set.seed(1)
dat <- data.frame(x = rnorm(100), z = rnorm(100))
dat$y <- dat$x + 0.5 * dat$x * dat$z + rnorm(100)
figs <- jn_plots(JN(lm(y ~ x * z, data = dat),
                    theta_1 = "x", theta_2 = "z"))
names(figs)

```

jn_regions

*Regions of significance of a Johnson-Neyman analysis***Description**

Reports, for each focal variable, the stretch of the moderator over which the conditional effect is significant. This is what [summary.JN](#) prints.

Usage

```
jn_regions(object, ...)

## S3 method for class 'JN_2way'
jn_regions(object, ...)

## S3 method for class 'JN_3way'
jn_regions(object, at = NULL, ...)
```

Arguments

object	a JN object.
...	ignored.
at	for three-way analyses, the values of the first moderator at which the region along the second moderator is reported. Defaults to its quartiles, which keeps the printed output readable; the full grid is always available through as.data.frame .

Details

For two-way Wald analyses the boundaries are solved for exactly rather than read off the grid, so their precision does not depend on `range_size`. For posterior analyses, and for the three-way case, the boundaries are obtained by linear interpolation between adjacent grid points.

Value

A data frame with one row per region, holding the focal variable, the moderator, the interval, the sign of the effect inside it, and – when the observed moderator values are known – the share of observations that fall inside it.

See Also

[summary.JN](#)

jn_save

*Save the figures of a Johnson-Neyman analysis***Description**

Writes every figure to disk with `ggplot2::ggsave()`. Useful for grouped analyses, where inspecting dozens of figures interactively is impractical.

Usage

```

jn_save(
  x,
  folder = NULL,
  device = "png",
  width = 10,
  height = 7,
  dpi = 600,
  ...
)

```

Arguments

<code>x</code>	a JN or JN_list object.
<code>folder</code>	directory to write into; created if it does not exist. Defaults to a session-temporary directory.
<code>device</code>	file extension passed to <code>ggsave()</code> . Default "png".
<code>width, height, dpi</code>	passed to <code>ggsave()</code> .
<code>...</code>	passed to <code>jn_plots</code> .

Value

The paths written, invisibly.

Examples

```

set.seed(1)
dat <- data.frame(x = rnorm(100), z = rnorm(100))
dat$y <- dat$x + 0.5 * dat$x * dat$z + rnorm(100)
jn <- JN(lm(y ~ x * z, data = dat), theta_1 = "x", theta_2 = "z")
jn_save(jn, folder = tempfile("jn"))

```

jn_style

*Appearance of Johnson-Neyman figures***Description**

Collects the colour and pattern settings shared by every figure the package draws, so that they can be set once and reused.

Usage

```

jn_style(
  sig_color = "seagreen3",
  non_sig_color = "chocolate",
  line_color = "black",
  color_low = "#3B9AB2",
  color_mid = "#EBCC2A",
  color_high = "#F21A00",
  color_values = "grey40",
  color_grid = "black",
  grid_density = 0.03,
  grid_spacing = 0.05,
  grid_linewidth = 0,
  crosshatch_non_sig = TRUE,
  show_density = TRUE,
  density_fill = "grey70",
  density_color = "grey30",
  density_height = 0.28,
  density_bins = 30
)

```

Arguments

sig_color, non_sig_color
ribbon fill for the significant and non-significant parts of a two-way plot.

line_color
colour of the conditional-effect line.

color_low, color_mid, color_high
the diverging fill scale used by the heatmaps and by the two-way posterior densities. The defaults are taken from the Zissou1 palette.

color_values
colour of the within-cell value labels drawn on small heatmaps.

color_grid, grid_density, grid_spacing, grid_linewidth
colour, density, spacing and line weight of the crosshatch drawn over non-significant heatmap cells. The defaults are chosen to read as texture over the fill rather than to hide it; widen grid_spacing for a coarser hatch.

crosshatch_non_sig
crosshatch the non-significant cells (TRUE, the default) or the significant ones?

show_density draw the data-density panels? Defaults to TRUE, and is ignored when the analysis carries no observed values.

density_fill, density_color fill and outline of the density panels.

density_height size of the density panels relative to the main panel.

density_bins number of histogram bins in the density panels.

Value

A list of settings, of class jn_style.

Examples

```
set.seed(1)
dat <- data.frame(x = rnorm(100), z = rnorm(100))
dat$y <- dat$x + 0.5 * dat$x * dat$z + rnorm(100)
jn <- JN(lm(y ~ x * z, data = dat), theta_1 = "x", theta_2 = "z")

plot(jn, style = jn_style(sig_color = "steelblue", show_density = FALSE))
```

plot.JN

Plot a Johnson-Neyman analysis

Description

Draws the figure for one focal variable. Two-way Wald analyses give the classic plot of the conditional effect against the moderator, with the confidence band shaded by significance; three-way analyses give a heatmap over the two-dimensional moderator grid, crosshatched where the effect is not significant; two-way posterior analyses give either overlaid conditional posterior densities or, with type = "band", the posterior mean and credible band against the moderator.

Usage

```
## S3 method for class 'JN'
plot(x, which = NULL, type = NULL, style = jn_style(), ...)

## S3 method for class 'JN'
autoplot(object, which = 1L, type = NULL, style = jn_style(), ...)

## S3 method for class 'JN_list'
plot(x, which = NULL, ...)
```

Arguments

<code>x</code>	a JN object.
<code>which</code>	the focal variable(s) to plot, by name or position. Defaults to all of them: every variable involved in the interaction takes its turn as the focal one, so a two-way analysis has two figures and a three-way analysis three.
<code>type</code>	for two-way posterior analyses, "density" (the default) for overlaid conditional posteriors or "band" for the posterior mean with its credible band; for three-way posterior analyses, "estimate" (the default) for the posterior mean or "bayes_p" for the Bayesian p value.
<code>style</code>	appearance settings from jn_style .
<code>...</code>	ignored.
<code>object</code>	a JN object.

Details

When the observed values of the moderators are known – automatically for `lm`, `glm` and **lme4** models, and otherwise from the `support` argument of [JN](#) – a histogram of them is attached to the figure: below the panel for two-way plots, along the top and right edges for heatmaps. Regions of significance covering moderator values that were hardly observed are the standard failure mode of this technique, and these panels make them visible. For heatmaps the histograms are marginal, so a cell can look well supported on both axes even though few observations lie near that combination of moderator values; the joint distribution is not shown.

Value

Invisibly, the figure when a single `which` was given, and a named list of them otherwise. Each figure is a **ggplot2** object, or a **patchwork** composition of one where density panels are attached; both print, extend with `+` and save with `ggsave()` in the usual way.

`plot()` draws; [jn_plots](#) and `autoplot()` return figures without drawing them.

Examples

```
set.seed(1)
dat <- data.frame(x = rnorm(200), z = rnorm(200))
dat$y <- dat$x + 0.6 * dat$x * dat$z + rnorm(200)
jn <- JN(lm(y ~ x * z, data = dat), theta_1 = "x", theta_2 = "z")

plot(jn)                # both figures: x moderated by z, and z by x
plot(jn, which = "x")    # just the one
plot(jn, style = jn_style(show_density = FALSE))
```

print.JN	<i>Print a Johnson-Neyman analysis</i>
----------	--

Description

Print a Johnson-Neyman analysis

Usage

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

Arguments

x	a JN object.
...	ignored.

Value

x, invisibly.

print.JN_list	<i>Methods for grouped Johnson-Neyman analyses</i>
---------------	--

Description

A JN_list holds one analysis per group, as produced by [JN](#) with fixed_only = FALSE for **lme4** models or hyper_only = FALSE for **multiSiena** models. Its first element is the fixed-effect, respectively hyper-parameter, analysis.

Usage

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

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

## S3 method for class 'JN_list'
as.data.frame(x, ...)
```

Arguments

x, object	a JN_list object.
...	passed to the corresponding JN method.

Value

`print()` returns its argument invisibly; `summary()` a list of summaries; `as.data.frame()` the grids of every group stacked into one data frame.

<code>summary.JN</code>	<i>Summarize a Johnson-Neyman analysis</i>
-------------------------	--

Description

Reports the regions of significance: the stretches of the moderator over which the conditional effect of each focal variable is distinguishable from zero. Where the observed moderator values are known, each region is annotated with the share of observations falling inside it, so that a region resting on almost no data is visible as such.

Usage

```
## S3 method for class 'JN'
summary(object, at = NULL, ...)

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

Arguments

<code>object</code>	a JN object.
<code>at</code>	for three-way analyses, the values of the first moderator at which to report the region along the second. Defaults to the quartiles.
<code>...</code>	ignored. The range each variable was evaluated over is reported alongside the regions, since a boundary can only ever be found inside it and the regions are not interpretable without it.
<code>x</code>	a <code>summary.JN</code> object.

Details

Regions that run to the edge of the evaluated range are marked with an asterisk: their boundary is where the moderator stopped being evaluated, not a Johnson-Neyman boundary.

Value

An object of class `summary.JN`, holding the region table in its `regions` element and the evaluated range of each variable in its `ranges` element.

Examples

```
set.seed(1402)
dat <- data.frame(x = rnorm(200), z = rnorm(200))
dat$y <- dat$x + 0.6 * dat$x * dat$z + rnorm(200)
summary(JN(lm(y ~ x * z, data = dat), theta_1 = "x", theta_2 = "z"))
```

Index

`as.data.frame`, [6](#), [12](#)
`as.data.frame.JN`, [2](#)
`as.data.frame.JN_list` (`print.JN_list`),
[17](#)
`autoplot.JN` (`plot.JN`), [15](#)

`coef.JN` (`as.data.frame.JN`), [2](#)

`JN`, [3](#), [7](#), [10](#), [16](#), [17](#)
`jn_input`, [5](#), [6](#), [7](#)
`jn_plots`, [11](#), [13](#), [16](#)
`jn_posterior`, [5](#)
`jn_posterior` (`jn_input`), [7](#)
`jn_regions`, [12](#)
`jn_save`, [13](#)
`jn_style`, [11](#), [14](#), [16](#)
`jn_wald`, [5](#)
`jn_wald` (`jn_input`), [7](#)

`plot`, [6](#)
`plot.JN`, [6](#), [10](#), [11](#), [15](#)
`plot.JN_list` (`plot.JN`), [15](#)
`print`, [6](#)
`print.JN`, [17](#)
`print.JN_list`, [17](#)
`print.summary.JN` (`summary.JN`), [18](#)

`summary`, [6](#)
`summary.JN`, [12](#), [18](#)
`summary.JN_list` (`print.JN_list`), [17](#)

`vcov.JN` (`as.data.frame.JN`), [2](#)