

Package ‘unsum’

September 4, 2026

Title Reconstruct Raw Data from Summary Statistics

Version 0.3.0

Description Reconstructs all possible raw data that could have led to reported summary statistics. Provides a wrapper for the 'Rust' implementation of the 'CLOSURE' algorithm.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.2.0)

Imports cli, ggplot2 (>= 3.4.0), ggtext, grid, nanoparquet, rlang, roundwork, S7, scales, stats, tibble, utils

Collate 'doc-helpers.R' 'standalone-last-export.R' 'constants.R' 'utils.R' 'read-write-basic.R' 'performance.R' 'horns.R' 'extendr-wrappers.R' 'count.R' 'plot-horns-histogram-basic.R' 'plot-bar-basic.R' 'generate-basic.R' 'fn-formals.R' 'fn-factory.R' 'generate.R' 'compare-sprite-closure.R' 'demo-plot.R' 'horns-corrected.R' 'horns-plot-for-export.R' 'plot-bar.R' 'plot-ecdf-basic.R' 'predicates-basic.R' 'read-write.R' 's7-result.R' 'standalone-new-wrapper.R' 'zzz.R'

Config/rextendr/version 0.4.2.9000

SystemRequirements Cargo (Rust's package manager), rustc

URL <https://github.com/lhdjung/unsum>, <https://lhdjung.github.io/unsum/>

BugReports <https://github.com/lhdjung/unsum/issues>

Suggests devtools, ggdist, glue, knitr, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/website rmarkdown

NeedsCompilation yes

Author Lukas Jung [aut, cre],
Nathanael Larigaldie [ctb]
Maintainer Lukas Jung <jung-lukas@gmx.net>
Repository CRAN
Date/Publication 2026-09-04 18:30:02 UTC

Contents

closure_count_all	2
closure_gauge_complexity	4
closure_generate	5
closure_plot_bar	9
closure_plot_ecdf	11
closure_write	13
demo_plot_bar	15
horns	16
horns-frequency	19
sprite_generate	21
sprite_plot_bar	25
sprite_write	27
Index	29

closure_count_all	<i>Count CLOSURE samples in advance</i>
-------------------	---

Description

Determine how many samples `closure_generate()` would find for a given set of summary statistics.

- `closure_count_all()` counts all CLOSURE samples that correspond to the input summary statistics, but without actually generating the samples. This is much faster than `closure_generate()` if there are many results.
- `closure_count_initial()` only counts the first round of samples, from which all other ones would be generated. Based on scale range only.

This can help predict how much time `closure_generate()` would take, and avoid prohibitively long runs.

Usage

```
closure_count_all(
  mean,
  sd,
  n,
  scale_min,
  scale_max,
  rounding = "up_or_down",
  threshold = 5
)

closure_count_initial(scale_min, scale_max)
```

Arguments

mean	String (length 1). Reported mean.
sd	String (length 1). Reported sample standard deviation.
n	Numeric (length 1). Reported sample size.
scale_min, scale_max	Integers (length 1 each). Minimum and maximum of the scales to which the reported statistics refer.
rounding	String (length 1). Rounding method assumed to have created mean and sd. See <i>Rounding options</i> , but also the <i>Rounding limitations</i> section below. Default is "up_or_down" which, e.g., unrounds 0.12 to 0.115 as a lower bound and 0.125 as an upper bound.
threshold	Numeric (length 1). Number from which to round up or down, if rounding is any of "up_or_down", "up", and "down". Default is 5.

Value

Integer (length 1).

Examples

```
closure_count_all(
  mean = "3.5",
  sd = "1.7",
  n = 70,
  scale_min = 1,
  scale_max = 5
)

closure_count_initial(scale_min = 1, scale_max = 5)
```

closure_gauge_complexity

Heuristic to predict CLOSURE runtime

Description

Before you run `closure_generate()`, you may want to get a sense of the time it will take to run. Use `closure_gauge_complexity()` to compute a heuristics-based complexity score. For reference, here is how it determines the messages in `closure_generate()`:

Value	Message
if < 1	(no message)
else if < 2	"Just a second..."
else if < 3	"This could take a minute..."
else	"NOTE: Long runtime ahead!"

Usage

```
closure_gauge_complexity(mean, sd, n, scale_min, scale_max)
```

Arguments

mean	String (length 1). Reported mean.
sd	String (length 1). Reported sample standard deviation.
n	Numeric (length 1). Reported sample size.
scale_min, scale_max	Numeric (length 1 each). Minimal and maximal possible values. For example, with a 1-7 Likert scale, use <code>scale_min = 1</code> and <code>scale_max = 7</code> . Prefer the empirical min and max if available: they constrain the possible values further.

Details

The result of this function is hard to interpret. All it can do is to convey an idea about the likely runtime of CLOSURE. This is because the input parameters interact in highly dynamic ways, which makes prediction difficult.

In addition, even progress bars or updates at regular intervals (e.g., "10% complete") prove to be extremely challenging: the Rust code computes CLOSURE results in parallel, which makes it hard to get an overview of the total progress across all cores; and especially to display such information on the R level.

Value

Numeric (length 1).

Examples

```
# Low SD, N, and scale range:
closure_gauge_complexity(
  mean = 2.55,
  sd = 0.85,
  n = 84,
  scale_min = 1,
  scale_max = 5
)

# Somewhat higher:
closure_gauge_complexity(
  mean = 4.26,
  sd = 1.58,
  n = 100,
  scale_min = 1,
  scale_max = 7
)

# Very high:
closure_gauge_complexity(
  mean = 3.81,
  sd = 3.09,
  n = 156,
  scale_min = 1,
  scale_max = 7
)
```

closure_generate	<i>Generate CLOSURE samples</i>
------------------	---------------------------------

Description

Call `closure_generate()` to run the CLOSURE algorithm on a given set of summary statistics.

This can take seconds, minutes, or longer, depending on the input. Wide variance and large `n` often lead to many samples, i.e., long runtimes. These effects interact dynamically. For example, with large `n`, even very small increases in `sd` can greatly increase runtime and number of values found. Consider specifying `path` in these cases; see "Writing to disk" below.

If the inputs are inconsistent, there is no solution. The function will then return empty results and throw a warning.

Usage

```
closure_generate(
  mean,
  sd,
  n,
  scale_min,
```

```

    scale_max,
    items = 1,
    path = NULL,
    stop_after = NULL,
    include = c("stats_and_horns", "stats_only", "all"),
    rounding = "up_or_down",
    threshold = 5,
    ask_to_proceed = TRUE
  )

```

Arguments

mean	String (length 1). Reported mean.
sd	String (length 1). Reported sample standard deviation.
n	Numeric (length 1). Reported sample size.
scale_min, scale_max	Numeric (length 1 each). Minimal and maximal possible values. For example, with a 1-7 Likert scale, use <code>scale_min = 1</code> and <code>scale_max = 7</code> . Prefer the empirical min and max if available: they constrain the possible values further.
items	Numeric (length 1). Number of items composing the original measurement scale. Default is 1, which assumes a single-item scale.
path	String (length 1). Optionally, choose the directory where a new folder with CLOSURE results should be created. Use <code>path = "."</code> for your current working directory. See "Writing to disk" below.
stop_after	Numeric (length 1). Optionally, make CLOSURE stop searching after it found this many samples. Useful if the goal is just to check whether the inputs are consistent: e.g., with <code>stop_after = 1</code> , the function returns either one sample or none. The latter would indicate an inconsistency.
include	String (length 1). If results are written to disk, which parts of them should be included in the R output? • With <code>"stats_and_horns"</code>, the default, all parts except for the samples are included. • <code>"stats_only"</code> excludes the <code>"results"</code> tibble, i.e., samples and horns. • <code>"all"</code> reads the full results, including the samples and horns values.
rounding	String (length 1). Rounding method assumed to have created mean and sd. See <i>Rounding options</i> , but also the <i>Rounding limitations</i> section below. Default is <code>"up_or_down"</code> which, e.g., unrounds 0.12 to 0.115 as a lower bound and 0.125 as an upper bound.
threshold	Numeric (length 1). Number from which to round up or down, if rounding is any of <code>"up_or_down"</code> , <code>"up"</code> , and <code>"down"</code> . Default is 5.
ask_to_proceed	Logical (length 1). If the runtime is predicted to be very long in an interactive setting, should the function prompt you to proceed or abort? Default is TRUE.

Value

`closure_generate()` returns a named list of tibbles (data frames):

- `inputs`: Arguments to this function.
- `metrics_main`:
 - `samples_all`: double. Number of all samples. Equal to the number of rows in results.
 - `values_all`: double. Number of all individual values found. Equal to $n * \text{samples_all}$.
- `metrics_horns`:
 - `mean`: double. Average horns value of all samples. The horns index is a measure of dispersion for bounded scales; see [horns\(\)](#).
 - `uniform`: double. The value that mean would have if all samples were uniformly distributed; see [horns_uniform\(\)](#).
 - `sd`, `cv`, `mad`, `min`, `median`, `max`, `range`: double. Standard deviation, coefficient of variation, median absolute deviation, minimum, median, maximum, and range of the horns index values across all samples. Note that `mad` is not scaled using a constant, as `stats::mad()` is by default.
- `frequency`:
 - `samples`: string. Frequencies apply to one of three subsets of samples: "all" for all samples, "horns_min" for those samples with the lowest horns index among all samples, and "horns_max" for those samples with the highest horns index.
 - `value`: integer. Scale values derived from `scale_min` and `scale_max`.
 - `f_count`: double. Count of scale values in the group's medoid sample (the actual sample closest to the group centroid by EMD).
 - `f_relative`: double. Values' share of total values found.
- `results`:
 - `id`: integer. Runs from 1 to `samples_all`.
 - `sample` (not present by default if `path` was specified): list of integer vectors. Each of these vectors has length `n`. It is a sample (or distribution) of individual scale values found by CLOSURE.
 - `horns`: double. Horns index of each sample.
- `directory` (only present if `path` was specified):
 - `path`: string. Location of the folder in which the results were saved.

Writing to disk

Specify `path` if the expected runtime is very long. (In case you have trouble choosing a path, use `path = "."` for your current working directory.) This makes sure the results are preserved by incrementally writing them to disk. Otherwise, you might encounter an out-of-memory error because `closure_generate()` accumulates more data than your computer can hold in memory.

More about memory

Some output columns that contain counts, such as `f_count`, are doubles instead of integers. This is because doubles are able to contain much larger numbers. When counting CLOSURE results, it is possible to exceed the limit of 32-bit integers in R, which is roughly two billion. For instance, this will happen with `values_all` in the examples (`data_high`) if `n` is changed to 120 and `scale_max` to 7.

Rounding limitations

The rounding and threshold arguments are not fully implemented. For example, CLOSURE currently treats all rounding bounds as inclusive, even if the rounding value would imply otherwise. Many specifications of the two arguments will not make any difference, and those that do will most likely lead to empty results.

Printing

When printing results, you can use `print()` explicitly with the `show` argument to control which elements are shown. Set `show` to one of:

- "some" (the default): show some elements, hide others. The hidden ones have brief descriptions.
- "all": show all elements.
- "none": show no elements, only their descriptions.

For example: `print(your_results, show = "all")`

Examples

```
# High spread often leads to many samples --
# here, 2492.
data_high <- closure_generate(
  mean = "3.5",
  sd = "1.7",
  n = 70,
  scale_min = 1,
  scale_max = 5
)

data_high

# Get a clear picture of the distribution
# by following up with `closure_plot_bar()`:
closure_plot_bar(data_high)

# Low spread, only 4 samples, and not all
# scale values are possible.
data_low <- closure_generate(
  mean = "2.9",
  sd = "0.5",
  n = 25,
  scale_min = 1,
  scale_max = 5
)

data_low

# This can also be shown by `closure_plot_bar()`:
closure_plot_bar(data_low)
```

closure_plot_bar

Visualize CLOSURE data in a barplot

Description

Call `closure_plot_bar()` to get a barplot of CLOSURE results.

For each scale value, the bars show how often this value appears in the mean samples with the minimum or maximum horns index (*h*). This displays the typical sample with the least or most amount of variance from among all CLOSURE samples.

As CLOSURE finds all possible samples, this plot shows the min and max *possible* amount of variance given the input statistics.

Usage

```
closure_plot_bar(
  data,
  min_max = c("both", "min", "max"),
  format = c("percent", "absolute_percent", "absolute", "relative"),
  samples = c("mean", "all"),
  overlay = c("all_avg", "none", "interval", "pointinterval", "dots"),
  facet_labels = c("Minimal variance", "Maximal variance"),
  facet_labels_parens = "h",
  bar_alpha = 0.75,
  bar_color = "#5D3FD3",
  show_text = TRUE,
  text_color = bar_color,
  text_size = 12,
  text_offset = 0.05,
  mark_thousand = ",",
  mark_decimal = "."
)
```

Arguments

<code>data</code>	List returned by <code>closure_generate()</code> or <code>closure_read()</code> .
<code>min_max</code>	String (length 1). Which plot panel(s) to show? Options are "both" (the default), "min", and "max".
<code>format</code>	String (length 1). What should the bars show? The default is "percent". Similarly, "absolute_percent" shows the count of each scale value and its percentage of all values. Other options are "absolute" and "relative" frequencies.
<code>samples</code>	String (length 1). How to aggregate the samples? Either take the average sample ("mean", the default) or the sum of all samples ("all"). This only matters if absolute frequencies are shown.
<code>overlay</code>	String (length 1). Visualization mode for the frequency distribution across all samples, overlaid behind the main bars. Only applies when <code>samples = "mean"</code> . Options:

	• "all_avg" (default): gray background bars showing the mean frequency across all samples. • "none": no overlay. • "interval": nested quantile intervals via <code>ggdist::stat_interval()</code> (requires the <code>ggdist</code> package and <code>include = "all"</code> in <code>closure_generate()</code>). • "pointinterval": median point with quantile interval lines via <code>ggdist::stat_pointinterval()</code> (same requirements as "interval"). • "dots": quantile dot plot via <code>ggdist::stat_dots()</code> (same requirements as "interval").
facet_labels	String (length 2). Labels of the two individual panels. Set it to NULL to remove the labels. Default is <code>c("Minimal variance", "Maximal variance")</code> .
facet_labels_parens	String (length 1). Italicized part of the facet labels inside the parentheses. Set it to NULL to remove the parentheses altogether. See details. Default is "h".
bar_alpha	Numeric (length 1). Opacity of the bars. Default is 0.75.
bar_color	String (length 1). Color of the bars. Default is "#5D3FD3", a purple color.
show_text	Logical (length 1). Should the bars be labeled with the corresponding frequencies? Default is TRUE.
text_color	String (length 1). Color of the frequency labels. By default, the same as <code>bar_color</code> .
text_size	Numeric (length 1). Base font size in pt. Default is 12.
text_offset	Numeric (length 1). Distance between the text labels and the bars. Default is 0.05.
mark_thousand, mark_decimal	Strings (length 1 each). Delimiters between groups of digits in text labels. Defaults are ", " for <code>mark_thousand</code> (e.g., "20,000") and "." for <code>mark_decimal</code> (e.g., "0.15").

Details

By default, both faceted plots have a label that includes their horns index (*h*); see `horns()`. You can remove the parenthesized part using `facet_labels_parens = NULL` or the entire label using `facet_labels = NULL`.

Value

A `ggplot` object.

See Also

`closure_plot_ecdf()`, an alternative visualization.

Examples

```
# Create CLOSURE data first:
data <- closure_generate(
  mean = "2.9",
  sd = "0.5",
```

```

    n = 70,
    scale_min = 1,
    scale_max = 5
  )

  # Even with minimal and maximal variance,
  # the results are almost the same:
  closure_plot_bar(data)

  # Only show the minimum-variance panel:
  closure_plot_bar(data, min_max = "min")

```

closure_plot_ecdf

Visualize CLOSURE data in an ECDF plot

Description

Call `closure_plot_ecdf()` to visualize CLOSURE results using the data's empirical cumulative distribution function (ECDF). This can be useful to display any variation between CLOSURE samples.

See [closure_plot_bar\(\)](#) for more intuitive visuals.

Usage

```

closure_plot_ecdf(
  data,
  samples = c("mean_min_max", "mean", "all"),
  pad = c("extend", "match", "stop"),
  legend_title = NULL,
  line_color_single = "#5D3FD3",
  line_color_multiple = c("royalblue4", "deeppink", "darkcyan"),
  text_size = 12,
  reference_line_alpha = 0.6,
  mark_decimal = "."
)

```

Arguments

- | | |
|---------|---|
| data | List returned by closure_generate() or closure_read() . |
| samples | String (length 1). How to map the samples to ECDF lines? <ul style="list-style-type: none"> • "mean_min_max", the default, draws three lines: overall mean across all samples, mean of the samples with the minimum horns index, and mean of the samples with the maximum horns index. • "mean" draws a single line for the overall mean. • "all" draws a separate line for each sample, colored by its horns index value. <i>Note:</i> This is invalid if <code>data\$results</code> does not include the sample and horns columns. If many samples were found, it can be very slow or even crash your R session. |

pad	String (length 1). How far should the ECDF line(s) stretch? • "extend", the default, draws the lines to both ends of the y-axis vertically and slightly beyond that horizontally, as <code>ggplot2::stat_ecdf()</code> does by default. • "match" draws them vertically as above, but not horizontally. <i>Note:</i> This is currently invalid in combination with <code>samples = "all"</code>. • "stop" does not draw the lines beyond the data points at all.
legend_title	String (length 1). Defaults for the legend title depend on <code>samples</code> : • With <code>samples = "mean_min_max"</code>, the legend title is absent by default because it can make the legend extend beyond the plot itself. If you do choose a title, consider <code>legend_title = "Subset of samples"</code>. • With <code>samples = "mean"</code>, there is no legend, and hence no title. • With <code>samples = "all"</code>, the title says "Horns index" unless you provide a different one. To remove the legend or change its position, use <code>legend.position</code> in <code>ggplot2::theme()</code>.
line_color_single	String (length 1). If <code>samples</code> is "mean", this is the color of the single ECDF line. Default is "#5D3FD3", a purple color.
line_color_multiple	String (length 3). If <code>samples</code> is "mean_min_max", these are the colors of the three ECDF lines. Default is "royalblue4" for the overall mean, "deeppink" for the minimum horns index, and "darkcyan" for the maximum horns index; in this order. If <code>samples</code> is "all", the colors for min and max horns index values are used for the low and high ends of the gradient.
text_size	Numeric (length 1). Base font size in pt. Default is 12.
reference_line_alpha	Numeric (length 1). Opacity of the diagonal reference line. Default is 0.6.
mark_decimal	String (length 1). Decimal delimiter in the labels. Default is "." (e.g., "0.15").

Details

This function was inspired by `rsprite2::plot_distributions()` with its option `plot_type = "ecdf"`. However, `plot_distributions()` invariably shows one line per (randomly drawn) possible dataset, and it does not support the horns index or other measures of dispersion. Some further differences exist.

Value

A ggplot object.

Examples

```
# Create CLOSURE data first:
data <- closure_generate(
  mean = "3.5",
```

```

sd = "2",
n = 52,
scale_min = 1,
scale_max = 5
)

# Visualize:
closure_plot_ecdf(data)

```

closure_write

*Write CLOSURE results to disk (and read them back in)***Description**

You can use `closure_write()` to save the results of `closure_generate()` on your computer. A message will show the exact location.

The data are saved in a new folder as five separate files, one for each tibble in `closure_generate()`'s output.

`closure_read()` is the opposite: it reads those files back into R, recreating the original CLOSURE list. This is useful for later analyses if you don't want to re-run a lengthy `closure_generate()` call. It also works with results that `closure_generate()` wrote to disk itself using `path = "your/path"`.

Usage

```

closure_write(data, path)

closure_read(
  path,
  include = c("stats_only", "stats_and_horns", "capped_error", "all"),
  samples_cap = NULL
)

```

Arguments

<code>data</code>	List returned by <code>closure_generate()</code> .
<code>path</code>	String (length 1). File path where <code>closure_write()</code> will create a new folder with the results. Set it to <code>"."</code> to choose the current working directory. For <code>closure_read()</code> , the path to an existing folder with results.
<code>include</code>	String (length 1). Which parts of the detailed results should be read in? • With <code>"stats_only"</code>, the default, no results are read. • <code>"stats_and_horns"</code> reads the horns index values, but not the samples. • <code>"capped_error"</code> checks whether the number of samples is higher than a given threshold (see <code>samples_cap</code>). If so, it throws an error; but if not, it reads both the samples and the horns values. • <code>"all"</code> reads both the samples and the horns values.
<code>samples_cap</code>	Numeric (length 1). When using <code>include = "capped_error"</code> , enter a whole number here to specify a cap. Default is <code>NULL</code> .

Details

`closure_write()` saves all tibbles as Parquet files. This is much faster and takes up far less disk space — roughly 1% of a CSV file with the same data. Speed and disk space can be relevant with large result sets.

Use `closure_read()` to import the CLOSURE list from the folder back into R. This is based on [nanoparquet::read_parquet\(\)](#).

Value

- `closure_write()` returns the path to the new folder it created.
- `closure_read()` returns a list of the same kind as [closure_generate\(\)](#).

Folder name

The new folder's name will contain all the inputs that determine the CLOSURE results. Dashes separate values and underscores replace decimal periods. For example:

```
CLOSURE-3_5-1_0-90-1-5-up_or_down-5
```

The order is the same as in `closure_generate()`:

```
closure_generate(
  mean = "3.5",
  sd = "1.0",
  n = 90,
  scale_min = 1,
  scale_max = 5,
  rounding = "up_or_down", # default
  threshold = 5            # default
)
```

Examples

```
data <- closure_generate(
  mean = "2.7",
  sd = "0.6",
  n = 45,
  scale_min = 1,
  scale_max = 5
)
```

```
# Writing to a temporary folder just for this example.
# You should write to a real folder instead.
# A simple way is path = "." for your current directory.
path_new_folder <- closure_write(data, path = tempdir())
```

```
# In a later session, conveniently read the files
# back into R. This returns the original list,
# identical except for floating-point error.
# (Of course, the `path_new_folder` variable will
# no longer be available -- instead, paste the path
# to your folder here.)
closure_read(path_new_folder)
```

demo_plot_bar

Visualize example distributions and their h values

Description

The `demo_plot_*()` functions are variants of `closure_plot_*()` that directly visualize a given frequency distribution. Their purpose is to illustrate general points about CLOSURE-type techniques and the horns index.

- `demo_plot_bar()` is like `closure_plot_bar()` except it is never faceted.
- `demo_plot_horns_histogram()` is like `closure_plot_horns_histogram()`.
- `demo_plot_ecdf()` is like `closure_plot_ecdf()`.

The top line shows the horns index of the given distribution (h) and the horns index of a hypothetical uniform distribution with the same number of scale points (h_u). See `horns()` and `horns_uniform()` for the corresponding functions.

Usage

```
demo_plot_bar(
  freqs,
  format = c("percent", "absolute_percent", "absolute", "relative"),
  bar_alpha = 0.75,
  bar_color = "#5D3FD3",
  show_text = TRUE,
  text_color = bar_color,
  text_size = 12,
  text_offset = 0.05,
  mark_thousand = ",",
  mark_decimal = "."
)
```

Arguments

<code>freqs</code>	Numeric. Vector of relative or absolute frequencies to visualize.
<code>format</code>	String (length 1). What should the bars show? The default is "percent". Similarly, "absolute_percent" shows the count of each scale value and its percentage of all values. Other options are "absolute" and "relative" frequencies.

bar_alpha	Numeric (length 1). Opacity of the bars. Default is 0.75.
bar_color	String (length 1). Color of the bars. Default is "#5D3FD3", a purple color.
show_text	Logical (length 1). Should the bars be labeled with the corresponding frequencies? Default is TRUE.
text_color	String (length 1). Color of the frequency labels. By default, the same as bar_color.
text_size	Numeric (length 1). Base font size in pt. Default is 12.
text_offset	Numeric (length 1). Distance between the text labels and the bars. Default is 0.05.
mark_thousand, mark_decimal	Strings (length 1 each). Delimiters between groups of digits in text labels. Defaults are "," for mark_thousand (e.g., "20,000") and "." for mark_decimal (e.g., "0.15").

Details

In keeping with the forensic metascience tradition of tortured backronyms, DEMO stands for "displaying examples of meticulous operation".

Value

A ggplot object.

Examples

```
# Zero variance: h = 0
demo_plot_bar(freqs = c(0, 40, 0, 0, 0))

# Perfect "horns of no confidence": h = 1
demo_plot_bar(freqs = c(20, 0, 0, 0, 20))

# Grouped around h = ~0.44, the uniform horns index
# for 7-point scales
demo_plot_bar(freqs = 21:27)
```

horns	<i>Horns index (h)</i>
-------	------------------------

Description

horns() measures the dispersion in ordinal data based on scale limits or min/max values. The result is the actual variance as a proportion of the maximum possible variance. It ranges from 0 to 1:

- 0 means no variance, i.e., all observations have the same value.
- 1 means that the observations are evenly split between the extremes, with none in between.

`horns_uniform()` computes the value that `horns()` would return for a uniform distribution within given scale limits. This can be useful as a point of reference for `horns()`.

These two functions correspond to the `horns` and `horns_uniform` columns in `closure_generate()`.

`horns_rescaled()` is a version of `horns()` that is normalized by scale length, such that 0.5 always indicates a uniform distribution, independent of the number of scale points. It is meant to enable comparison across scales of different lengths, but it is harder to interpret for an individual scale. This makes it unlikely to be useful in most cases. Even so, the range and the meaning of 0 and 1 are the same as for `horns()`.

Usage

```
horns(freqs, scale_min, scale_max)
```

```
horns_uniform(scale_min, scale_max)
```

```
horns_rescaled(freqs, scale_min, scale_max)
```

Arguments

`freqs` Numeric. Vector with the frequencies (relative or absolute) of binned observations; e.g., a vector with 5 elements for a 1-5 scale.

`scale_min, scale_max`

Numeric (length 1 each). Minimal and maximal possible values. For example, with a 1-7 Likert scale, use `scale_min = 1` and `scale_max = 7`. Prefer the empirical min and max if available: they constrain the possible values further.

Details

The horns index h is defined as:

$$h = \frac{\sum_{i=1}^k f_i (i - \bar{s})^2}{\frac{1}{4}(k-1)^2}$$

where k is the number of scale points (i.e., the length of `freqs` here), f_i is the relative frequency of the i th point on an integer scale from 1 to k , and $\bar{s}$ is the weighted mean frequency. The mean is derived as follows:

$$\bar{s} = \sum_{i=1}^k i f_i$$

Note that h only depends on the frequency distribution, not the actual values of the scale points. This is why both formulas invariably use a scale from 1 to k . The only reason why the `horns()` function still takes `scale_min` and `scale_max` arguments is safety: if `freqs` is misstated such that its length is different from the number of points on the scale implied by those two arguments, there will be an error.

Maximum possible variance:

The "maximum possible" variance here is based on scale range only (Popoviciu 1935):

$$\sigma_{\max}^2 = \frac{1}{4}(k-1)^2$$

It is deliberately agnostic to the mean and the sample size. In the context of CLOSURE, these statistics are only relevant for generating possible samples, which then enter the equation via the numerator; just like the standard deviation. By contrast, the number of scale points, k , is an intrinsic property of those samples. It is sensible to assess the variance of the samples by benchmarking it against the greatest variance that can occur in any sample with the same k .

Uniform distribution:

Although `horns_uniform()` is implemented as a wrapper around `horns()` that constructs a perfect uniform distribution internally, an equivalent closed-form solution can be given as

$$h_u = \frac{k+1}{3(k-1)}$$

"Horns of no confidence":

The term *horns index* was inspired by [Heathers \(2017\)](#) which defines the "horns of no confidence" as a reconstructed sample "where an incorrect, impossible or unlikely value set has all its constituents stacked into its highest or lowest bins to try meet a ludicrously high SD". In its purest form, this is a case where $h = 1$, so `horns()` would return 1. However, note that the implications for the plausibility of any given set of summary statistics depend on the substantive context of the data ([Heathers et al. 2018](#)).

Heathers wrote about SPRITE, but the logic applies equally to CLOSURE. Indeed, as CLOSURE is exhaustive but SPRITE is not, it is only CLOSURE that provides certainty about the set of possible samples and their horns indices.

Rust implementation:

The `metrics_horns` tibble that is part of the output of `closure_generate()` is not based on the R functions presented here. Instead, it relies on efficient Rust implementations of the above formulas. These Rust functions are part of `closure-core`, which mainly implements CLOSURE but does not currently export horns functions for users.

Value

Numeric (length 1).

References

Popoviciu, T. (1935). Sur les équations algébriques ayant toutes leurs racines réelles. *Mathematica* (Cluj), 9, 129-145.

Examples

```
# For simplicity, all examples use a 1-5 scale and a total N of 300.

# ---- With all values at the extremes
```

```

horns(freqs = c(300, 0, 0, 0, 0), scale_min = 1, scale_max = 5)

horns(c(150, 0, 0, 0, 150), 1, 5)

horns(c(100, 0, 0, 0, 200), 1, 5)

# ---- With some values in between

horns(c(60, 60, 60, 60, 60), 1, 5)

horns(c(200, 50, 30, 20, 0), 1, 5)

horns(c(150, 100, 50, 0, 0), 1, 5)

horns(c(100, 40, 20, 40, 100), 1, 5)

```

horns-frequency

Visualize horns index (h) frequencies

Description

`closure_plot_horns_histogram()` visualizes the distribution of horns index values found by CLOSURE.

It marks the min, max, and uniform values; see [horns_uniform\(\)](#). The x-axis always ranges from 0 to 1. This reveals the big picture, putting any variability among horns values into perspective.

Usage

```

closure_plot_horns_histogram(
  data,
  bar_color = "#5D3FD3",
  alpha = 0.75,
  binwidth = 0.01,
  show_labels = c("all", "min_max", "min_max_uniform", "min_max_bounds", "uniform",
    "uniform_bounds", "bounds", "none"),
  line_color_min_max = "red",
  line_color_reference = "grey20",
  text_limits = c(0.12, 0.88),
  text_size = 12,
  mark_thousand = ",",
  mark_decimal = "."
)

```

Arguments

<code>data</code>	List returned by <code>closure_generate()</code> or <code>closure_read()</code> .
<code>bar_color</code>	String (length 1). Color of the bars. Default is "#5D3FD3", a purple color.
<code>alpha</code>	Numeric (length 1). Opacity of the bars. Default is 0.75.
<code>binwidth</code>	Numeric (length 1). Only in <code>closure_plot_horns_histogram()</code> . Width of the bins that divide up the x-axis. Default is 0.01.
<code>show_labels</code>	String (length 1). Which text labels to show within the plot? One of: "all" (the default): <i>min</i>, <i>max</i>, and <i>uniform</i> horns values from data, as well as the <i>bounds</i> at 0 and 1. "min_max", "min_max_uniform", "min_max_bounds", "uniform", "uniform_bounds", or "bounds": a subset of these labels. "none": no labels.
<code>line_color_min_max</code>	String (length 1). Color of the lines that mark the lower and upper ends of the distribution. Default is "red".
<code>line_color_reference</code>	String (length 1). Color of the lines at $h = 0$, $h = 1$, and the uniform point. Default is "grey20".
<code>text_limits</code>	Numeric (length 2). If the minimum horns index is lower than the first element here (default: 0.12), both text labels go to the right of the maximum. The same applies in reverse with the maximum and the second element (default: 0.88).
<code>text_size</code>	Numeric (length 1). Base font size in pt. Default is 12.
<code>mark_thousand</code> , <code>mark_decimal</code>	Strings (length 1 each). Delimiters between groups of digits in text labels. Defaults are ", " for <code>mark_thousand</code> (e.g., "20,000") and "." for <code>mark_decimal</code> (e.g., "0.15").

Value

A ggplot object.

Examples

```
data_near_zero <- closure_generate(
  mean = "2.1",
  sd = "0.4",
  n = 35,
  scale_min = 1,
  scale_max = 7
)

# All CLOSURE samples have low horns indices:
closure_plot_horns_histogram(data_near_zero)

data_near_midpoint <- closure_generate(
  mean = "4.3",
  sd = "2.2",
```

```

    n = 35,
    scale_min = 1,
    scale_max = 7
  )

  # The min-max lines are beyond the midpoint,
  # so all CLOSURE samples are bimodal:
  closure_plot_horns_histogram(data_near_midpoint)

data_near_one <- closure_generate(
  mean = "4.3",
  sd = "2.8",
  n = 35,
  scale_min = 1,
  scale_max = 7
)

# Extremely high variance throughout; close to the upper limit.
# We can omit labels at the bounds so they don't cover the data:
closure_plot_horns_histogram(
  data_near_one,
  show_labels = "min_max_uniform"
)

# Large differences between horns values occur (only?)
# if `mean` and `sd` have no decimal places:
data_wide_spread <- closure_generate(
  mean = "3",
  sd = "2",
  n = 35,
  scale_min = 1,
  scale_max = 7
)

closure_plot_horns_histogram(data_wide_spread)

```

 sprite_generate

Generate SPRITE samples

Description

Call `sprite_generate()` to find possible samples using SPRITE (sample parameter reconstruction via iterative techniques). SPRITE reconstructs possible sample distributions given summary statistics and multi-item scale information.

`stop_after` is required when `items > 1` to prevent overflow errors in the current implementation.

Usage

```
sprite_generate(
```

```

mean,
sd,
n,
scale_min,
scale_max,
items = 1,
path = NULL,
stop_after = NULL,
include = c("stats_and_horns", "stats_only", "all"),
rounding = "up_or_down",
threshold = 5,
ask_to_proceed = TRUE
)

```

Arguments

mean	String (length 1). Reported mean.
sd	String (length 1). Reported sample standard deviation.
n	Numeric (length 1). Reported sample size.
scale_min, scale_max	Numeric (length 1 each). Minimal and maximal possible values. For example, with a 1-7 Likert scale, use scale_min = 1 and scale_max = 7. Prefer the empirical min and max if available: they constrain the possible values further.
items	Numeric (length 1). Number of items/questions in your scale. Must be at least 2. This represents how many individual items were averaged to produce each participant's mean score.
path	String (length 1). Optionally, choose the directory where a new folder with CLOSURE results should be created. Use path = "." for your current working directory. See "Writing to disk" below.
stop_after	Numeric (length 1). Required for SPRITE when items > 1. Limits the number of samples returned to prevent overflow. Recommended value: 100-1000 depending on your needs.
include	String (length 1). If results are written to disk, which parts of them should be included in the R output? • With "stats_and_horns", the default, all parts except for the samples are included. • "stats_only" excludes the "results" tibble, i.e., samples and horns. • "all" reads the full results, including the samples and horns values.
rounding	String (length 1). Rounding method assumed to have created mean and sd. See <i>Rounding options</i> , but also the <i>Rounding limitations</i> section below. Default is "up_or_down" which, e.g., unrounds 0.12 to 0.115 as a lower bound and 0.125 as an upper bound.
threshold	Numeric (length 1). Number from which to round up or down, if rounding is any of "up_or_down", "up", and "down". Default is 5.
ask_to_proceed	Logical (length 1). If the runtime is predicted to be very long in an interactive setting, should the function prompt you to proceed or abort? Default is TRUE.

Value

`sprite_generate()` returns a named list of tibbles (data frames):

- **inputs:** Arguments to this function.
- **metrics_main:**
 - **samples_all:** double. Number of all samples. Equal to the number of rows in `results`.
 - **values_all:** double. Number of all individual values found. Equal to $n * \text{samples_all}$.
- **metrics_horns:**
 - **mean:** double. Average horns value of all samples. The horns index is a measure of dispersion for bounded scales; see `horns()`.
 - **uniform:** double. The value that mean would have if all samples were uniformly distributed; see `horns_uniform()`.
 - **sd, cv, mad, min, median, max, range:** double. Standard deviation, coefficient of variation, median absolute deviation, minimum, median, maximum, and range of the horns index values across all samples. Note that `mad` is not scaled using a constant, as `stats::mad()` is by default.
- **frequency:**
 - **samples:** string. Frequencies apply to one of three subsets of samples: "all" for all samples, "horns_min" for those samples with the lowest horns index among all samples, and "horns_max" for those samples with the highest horns index.
 - **value:** integer. Scale values derived from `scale_min` and `scale_max`.
 - **f_count:** double. Count of scale values in the group's medoid sample (the actual sample closest to the group centroid by EMD).
 - **f_relative:** double. Values' share of total values found.
- **results:**
 - **id:** integer. Runs from 1 to `samples_all`.
 - **sample** (not present by default if `path` was specified): list of integer vectors. Each of these vectors has length `n`. It is a sample (or distribution) of individual scale values found by SPRITE.
 - **horns:** double. Horns index of each sample.
- **directory** (only present if `path` was specified):
 - **path:** string. Location of the folder in which the results were saved.

Writing to disk

Specify `path` if the expected runtime is very long. (In case you have trouble choosing a path, use `path = "."` for your current working directory.) This makes sure the results are preserved by incrementally writing them to disk. Otherwise, you might encounter an out-of-memory error because `sprite_generate()` accumulates more data than your computer can hold in memory.

More about memory

Some output columns that contain counts, such as `f_count`, are doubles instead of integers. This is because doubles are able to contain much larger numbers. When counting SPRITE results, it is possible to exceed the limit of 32-bit integers in R, which is roughly two billion.

Rounding limitations

The rounding and threshold arguments are not fully implemented. For example, SPRITE currently treats all rounding bounds as inclusive, even if the rounding value would imply otherwise. Many specifications of the two arguments will not make any difference, and those that do will most likely lead to empty results.

Printing

When printing results, you can use `print()` explicitly with the `show` argument to control which elements are shown. Set `show` to one of:

- "some" (the default): show some elements, hide others. The hidden ones have brief descriptions.
- "all": show all elements.
- "none": show no elements, only their descriptions.

For example: `print(your_results, show = "all")`

Examples

```
# Basic example with 2 items
## Not run:
data_simple <- sprite_generate(
  mean = "3.0",
  sd = "1.0",
  n = 120,
  scale_min = 1,
  scale_max = 5,
  stop_after = 150
)

# Larger example - use stop_after to avoid overflow
sprite_generate(
  mean = "3.5",
  sd = "1.7",
  n = 1000,
  items = 5,
  scale_min = 1,
  scale_max = 5,
  stop_after = 1000
)

## End(Not run)
```

 sprite_plot_bar

 Visualize SPRITE data in a barplot

Description

Call `sprite_plot_bar()` to get a barplot of SPRITE results.

For each scale value, the bars show how often this value appears in the mean samples with the minimum or maximum horns index (*h*). This displays the typical sample with the least or most amount of variance from among all SPRITE samples.

Usage

```
sprite_plot_bar(
  data,
  min_max = c("both", "min", "max"),
  format = c("percent", "absolute_percent", "absolute", "relative"),
  samples = c("mean", "all"),
  overlay = c("all_avg", "none", "interval", "pointinterval", "dots"),
  facet_labels = c("Minimal variance", "Maximal variance"),
  facet_labels_parens = "h",
  bar_alpha = 0.75,
  bar_color = "royalblue1",
  show_text = TRUE,
  text_color = bar_color,
  text_size = 12,
  text_offset = 0.05,
  mark_thousand = ",",
  mark_decimal = "."
)
```

Arguments

data	List returned by <code>sprite_generate()</code> .
min_max	String (length 1). Which plot panel(s) to show? Options are "both" (the default), "min", and "max".
format	String (length 1). What should the bars show? The default is "percent". Similarly, "absolute_percent" shows the count of each scale value and its percentage of all values. Other options are "absolute" and "relative" frequencies.
samples	String (length 1). How to aggregate the samples? Either take the average sample ("mean", the default) or the sum of all samples ("all"). This only matters if absolute frequencies are shown.
overlay	String (length 1). Visualization mode for the frequency distribution across all samples, overlaid behind the main bars. Only applies when <code>samples = "mean"</code> . Options:

	• "all_avg" (default): gray background bars showing the mean frequency across all samples. • "none": no overlay. • "interval": nested quantile intervals via <code>ggdist::stat_interval()</code> (requires the <code>ggdist</code> package and <code>include = "all"</code> in <code>closure_generate()</code>). • "pointinterval": median point with quantile interval lines via <code>ggdist::stat_pointinterval()</code> (same requirements as "interval"). • "dots": quantile dot plot via <code>ggdist::stat_dots()</code> (same requirements as "interval").
facet_labels	String (length 2). Labels of the two individual panels. Set it to NULL to remove the labels. Default is <code>c("Minimal variance", "Maximal variance")</code> .
facet_labels_parens	String (length 1). Italicized part of the facet labels inside the parentheses. Set it to NULL to remove the parentheses altogether. See details. Default is "h".
bar_alpha	Numeric (length 1). Opacity of the bars. Default is 0.75.
bar_color	String (length 1). Color of the bars. Default is "#5D3FD3", a purple color.
show_text	Logical (length 1). Should the bars be labeled with the corresponding frequencies? Default is TRUE.
text_color	String (length 1). Color of the frequency labels. By default, the same as <code>bar_color</code> .
text_size	Numeric (length 1). Base font size in pt. Default is 12.
text_offset	Numeric (length 1). Distance between the text labels and the bars. Default is 0.05.
mark_thousand, mark_decimal	Strings (length 1 each). Delimiters between groups of digits in text labels. Defaults are ", " for <code>mark_thousand</code> (e.g., "20,000") and "." for <code>mark_decimal</code> (e.g., "0.15").

Value

A ggplot object.

Examples

```
# Run SPRITE first
data <- sprite_generate(
  mean = "3.0",
  sd = "1.0",
  n = 120,
  scale_min = 1,
  scale_max = 5,
  stop_after = 150
)

sprite_plot_bar(data)
```

sprite_write	Write <i>SPRITE</i> results to disk (and read them back in)
--------------	---

Description

You can use `sprite_write()` to save the results of `sprite_generate()` on your computer. A message will show the exact location.

The data are saved in a new folder as five separate files, one for each tibble in `sprite_generate()`'s output.

`sprite_read()` is the opposite: it reads those files back into R, recreating the original *SPRITE* list. This is useful for later analyses if you don't want to re-run a lengthy `sprite_generate()` call. It also works with results that `sprite_generate()` wrote to disk itself using `path = "your/path"`.

Usage

```
sprite_write(data, path)

sprite_read(
  path,
  include = c("stats_only", "stats_and_horns", "capped_error", "all"),
  samples_cap = NULL
)
```

Arguments

data	List returned by <code>sprite_generate()</code> .
path	String (length 1). File path where <code>sprite_write()</code> will create a new folder with the results. Set it to <code>"."</code> to choose the current working directory. For <code>sprite_read()</code> , the path to an existing folder with results.
include	String (length 1). Which parts of the detailed results should be read in? • With <code>"stats_only"</code>, the default, no results are read. • <code>"stats_and_horns"</code> reads the horns index values, but not the samples. • <code>"capped_error"</code> checks whether the number of samples is higher than a given threshold (see <code>samples_cap</code>). If so, it throws an error; but if not, it reads both the samples and the horns values. • <code>"all"</code> reads both the samples and the horns values.
samples_cap	Numeric (length 1). When using <code>include = "capped_error"</code> , enter a whole number here to specify a cap. Default is <code>NULL</code> .

Details

`sprite_write()` saves all tibbles as Parquet files. This is much faster and takes up far less disk space — roughly 1% of a CSV file with the same data. Speed and disk space can be relevant with large result sets.

Use `sprite_read()` to import the *SPRITE* list from the folder back into R. This is based on `nanoparquet::read_parquet()`.

Value

- `sprite_write()` returns the path to the new folder it created.
- `sprite_read()` returns a list of the same kind as `sprite_generate()`.

Folder name

The new folder's name will contain all the inputs that determine the SPRITE results. Dashes separate values and underscores replace decimal periods. For example:

```
SPRITE-3_5-1_0-90-1-5-up_or_down-5
```

The order is the same as in `sprite_generate()`:

```
sprite_generate(
  mean = "3.5",
  sd = "1.0",
  n = 90,
  scale_min = 1,
  scale_max = 5,
  rounding = "up_or_down", # default
  threshold = 5             # default
)
```

Examples

```
data <- sprite_generate(
  mean = "2.7",
  sd = "0.6",
  n = 45,
  scale_min = 1,
  scale_max = 5,
  stop_after = 1000
)

# Writing to a temporary folder just for this example.
# You should write to a real folder instead.
# A simple way is path = "." for your current directory.
path_new_folder <- sprite_write(data, path = tempdir())

# In a later session, conveniently read the files
# back into R. This returns the original list,
# identical except for floating-point error.
# (Of course, the `path_new_folder` variable will
# no longer be available -- instead, paste the path
# to your folder here.)
sprite_read(path_new_folder)
```

Index

`closure_count_all`, [2](#)
`closure_count_initial`
 (`closure_count_all`), [2](#)
`closure_gauge_complexity`, [4](#)
`closure_generate`, [5](#)
`closure_generate()`, [2](#), [4](#), [9–11](#), [13](#), [14](#), [17](#),
 [18](#), [20](#), [26](#)
`closure_plot_bar`, [9](#)
`closure_plot_bar()`, [11](#), [15](#)
`closure_plot_ecdf`, [11](#)
`closure_plot_ecdf()`, [10](#), [15](#)
`closure_plot_horns_histogram`
 (`horns-frequency`), [19](#)
`closure_plot_horns_histogram()`, [15](#)
`closure_read` (`closure_write`), [13](#)
`closure_read()`, [9](#), [11](#), [20](#)
`closure_write`, [13](#)

`demo_plot_bar`, [15](#)

`ggdist::stat_dots()`, [10](#), [26](#)
`ggdist::stat_interval()`, [10](#), [26](#)
`ggdist::stat_pointinterval()`, [10](#), [26](#)
`ggplot2::stat_ecdf()`, [12](#)
`ggplot2::theme()`, [12](#)

`horns`, [16](#)
`horns()`, [7](#), [10](#), [15](#), [23](#)
`horns-frequency`, [19](#)
`horns_rescaled` (`horns`), [16](#)
`horns_uniform` (`horns`), [16](#)
`horns_uniform()`, [7](#), [19](#), [23](#)

`nanoparquet::read_parquet()`, [14](#), [27](#)

`rsprite2::plot_distributions()`, [12](#)

`sprite_generate`, [21](#)
`sprite_generate()`, [25](#), [27](#), [28](#)
`sprite_plot_bar`, [25](#)
`sprite_read` (`sprite_write`), [27](#)

`sprite_write`, [27](#)
`stats::mad()`, [7](#), [23](#)