Package 'nimbleExtra'

August 8, 2026

Type Package

Title Interoperating with 'NIMBLE' for Generic Analysis of MCMC Samples

Version 0.1.15

Description 'NIMBLE' is a general purpose modeling framework that facilitates the analysis of Markov Chain Monte Carlo (MCMC) models. However, it constructs a complex and variable data structure, which is difficult to use in a general way. The 'nimbleExtra' package provides a set of functions to insulate users from the complexity and variability of 'NIMBLE' data structures. It also complements 'NIMBLE' and other MCMC analysis packages by providing missing functionality to encourage generic programming. Together these features ease the burden of working with MCMC data and encourage more systematic approaches to MCMC analysis.

License BSD_3_clause + file LICENSE | GPL (>= 2)

Encoding UTF-8

LazyData true

Imports coda, dplyr, ggplot2, grDevices, nimble, posterior, purrr, rlang, tidyselect, utils

Suggests ggmcmc, scales, testthat (>= 3.0.0), tidyr

RoxygenNote 7.3.3

Config/build/clean-inst-doc FALSE

Config/testthat/edition 3

Depends R (>= 4.1.0)

NeedsCompilation no

Author Brook Milligan [aut, cre, cph]

Maintainer Brook Milligan <brook@biology.nmsu.edu>

Repository CRAN

Date/Publication 2026-08-08 11:30:13 UTC

Contents

`nimbleExtra-package` *nimbleExtra: Interoperating with 'NIMBLE' for Generic Analysis of MCMC Samples*

Description

'NIMBLE' is a general purpose modeling framework that facilitates the analysis of Markov Chain Monte Carlo (MCMC) models. However, it constructs a complex and variable data structure, which is difficult to use in a general way. The 'nimbleExtra' package provides a set of functions to insulate users from the complexity and variability of 'NIMBLE' data structures. It also complements 'NIMBLE' and other MCMC analysis packages by providing missing functionality to encourage generic programming. Together these features ease the burden of working with MCMC data and encourage more systematic approaches to MCMC analysis.

Details

The purpose of the nimbleExtra package is to address several challenges encountered with developing and analyzing Markov Chain Monte Carlo (MCMC) models in R. Although developed in part to ease the burden of using NIMBLE, a widely used, general purpose modeling framework that facilitates the analysis of MCMC models, nimbleExtra includes functions that are applicable to a broad array of situations.

NIMBLE is broadly useful, but has three features that make interoperation with it difficult. First, the most natural parameters used to specify a set of MCMC are not the ones taken as arguments by NIMBLE functions. As a result, varying those natural parameters entails follow-on calculations in order to construct the required arguments. Second, and vastly more important, the output from NIMBLE models does not adhere to a uniform format; rather, the resulting data structure depends greatly on the arguments given. As a result, downstream analysis of the MCMC samples requires bespoke decoding of the data, which makes generic programming extremely difficult. Finally, NIMBLE requires a functional C++ toolchain to compile its models, yet the process of installing one often leads to errors and correct operation may not be easily validated by many users. nimbleExtra addresses all of these issues.

Additionally, nimbleExtra provides features of interest to users of other packages for analysis of MCMC data. For example, because visualization of data is so crucial, it provides a set of ggplot2 stats intended to simplify common visualizations of MCMC samples, while retaining the full flexibility offered by the Grammar of Graphics. Because reshaping MCMC data is so often needed, it provides functions to simplify the use of `tidyr::pivot_longer()`. Because summarizing MCMC data is essential, it provides functions for inspecting dataframes, calculating probabilities and quantiles from empirical distributions, extracting summary statistics, and selecting a subset of samples. All of these facilitate the analysis of MCMC samples and encourage generic programming approaches.

Author(s)

Maintainer: Brook Milligan <brook@biology.nmsu.edu> [copyright holder]

`add_predictions` *Add predictions to a dataframe of parameters*

Description

`select_samples()` selects a subset of rows from a dataframe. Depending on the `method` argument, the subset may be deterministic or not.

Usage

```
add_predictions(data, .f, .name = ".prediction", ...)
```

Arguments

data	A dataframe containing all the parameters needed for the prediction function. (dataframe)
.f	The predictor function. The predictor function shall accept as arguments all the fields in the dataframe data; however, some or all may be absorbed by .... The names of the arguments to the predictor function shall match the names of the fields in the dataframe data. (vectorized function)
.name	The name of the new variable to contruct. Defaults to ".prediction". (character)
...	Additional arguments forwarded to purrr::pmap().

Value

A vector of predictions calculated from the fields in the dataframe.

Examples

```
f <- function(x, mu, sigma, ...) dnorm(x=x, mean=mu, sd=sigma)

nimble_data_m1_s1_w0_a0_c1 |> samples() |>
  tidyr::expand_grid(x=seq(-5,5,length.out=100)) |>
  add_predictions(f) |> str()
```

---

coda_samples *Extract a list of MCMC chains from NIMBLE output*

Description

coda_samples() extracts, as a coda::mcmc.list, the MCMC samples for one or all chains from the output generated by NIMBLE. Output from each chain is converted into a coda::mcmc data structure regardless of the NIMBLE option samplesAsCodaMCMC. Thus, the data structures used for downstream data analysis are insulated from the specifics of the NIMBLE function calls used to generate MCMC samples.

Usage

```
coda_samples(nimble_data, chain = NA, start = 1, end = numeric(0), thin = 1)
```

Arguments

nimble_data	A data structure containing MCMC samples (and optionally additional information) created by running a NIMBLE model via, for example, the function nimble::nimbleMCMC().
chain	Index(es) of chain(s) to extract. Defaults to NA, which extracts all samples. Any subset of chains (including duplications) may be extracted. (integer or integer vector)

`start`	The iteration number of the first observation. Defaults to 1. (integer)
`end`	The iteration number of the last observation. Defaults to 0. (integer)
`thin`	The thinning interval between consecutive observation. Defaults to 1. (integer)

Details

The arguments `start`, `end`, and `thin` are required to define the meaning of sampling intervals within the constructed coda::mcmc.list. This is required by, for example, downstream analysis to reliably diagnose convergence of the chain(s).

Value

A list, possibly of length 1, containing data for each of the extracted MCMC chains. (coda::mcmc.list)

Examples

```
nimble_data_m1_s1_w0_a0_c1 |> samples()
nimble_data_m1_s1_w0_a0_c1 |> samples() |> summary()
```

---

example_datasets *Sample NIMBLE datasets*

Description

NIMBLE functions can construct a variety of different native data structures depending on the arguments given. For example, for `nimble::nimbleMCMC()` the data generated is controlled by five different arguments for which there are 32 meaningful combinations (chains > 2 yield data structures identical to `chains = 2`, but with longer lists). Each of these datasets represents one of the 32 possibilities.

Usage

```
nimble_data_m0_s0_w0_a0_c1

nimble_data_m0_s0_w0_a0_c2

nimble_data_m0_s0_w0_a1_c1

nimble_data_m0_s0_w0_a1_c2

nimble_data_m0_s0_w1_a0_c1

nimble_data_m0_s0_w1_a0_c2

nimble_data_m0_s0_w1_a1_c1
```

nimble_data_m0_s0_w1_a1_c2

nimble_data_m0_s1_w0_a0_c1

nimble_data_m0_s1_w0_a0_c2

nimble_data_m0_s1_w0_a1_c1

nimble_data_m0_s1_w0_a1_c2

nimble_data_m0_s1_w1_a0_c1

nimble_data_m0_s1_w1_a0_c2

nimble_data_m0_s1_w1_a1_c1

nimble_data_m0_s1_w1_a1_c2

nimble_data_m1_s0_w0_a0_c1

nimble_data_m1_s0_w0_a0_c2

nimble_data_m1_s0_w0_a1_c1

nimble_data_m1_s0_w0_a1_c2

nimble_data_m1_s0_w1_a0_c1

nimble_data_m1_s0_w1_a0_c2

nimble_data_m1_s0_w1_a1_c1

nimble_data_m1_s0_w1_a1_c2

nimble_data_m1_s1_w0_a0_c1

nimble_data_m1_s1_w0_a0_c2

nimble_data_m1_s1_w0_a1_c1

nimble_data_m1_s1_w0_a1_c2

nimble_data_m1_s1_w1_a0_c1

nimble_data_m1_s1_w1_a0_c2

nimble_data_m1_s1_w1_a1_c1

`nimble_data_m1_s1_w1_a1_c2`

Format

`nimble_data_m*_s*_w*_a*_c*`:

The format created by `nimble::nimbleMCMC()` depends on the arguments given, which are encoded in the name of each dataset following the patterns below:

- `m0, m1`: `samples=FALSE` or `samples=TRUE` to indicate whether MCMC samples are included.
- `s0, s1`: `summary=FALSE` or `summary=TRUE` to indicate whether summaries of MCMC samples are included.
- `w0, w1`: `WAIC=FALSE` or `WAIC=TRUE` to indicate whether WAIC summaries of MCMC samples are included.
- `a0, a1`: `samplesAsCodaMCMC=FALSE` or `samplesAsCodaMCMC=TRUE` to indicate whether MCMC samples are represented as coda::mcmc objects.
- `cN` indicates the number (`N`) of MCMC chains included.

The following is a list of top-level formats for each of the datasets (in the order given above). The variation in data structures is evident from this alone; however, additional variation exists even among data structures with the same top-level structure. This is the complexity that nimbleExtra seeks to insulate downstream users from.

An object of class `NULL` of length 0.

An object of class `NULL` of length 0.

An object of class `NULL` of length 0.

An object of class `NULL` of length 0.

An object of class `waicNimbleList` of length 1.

An object of class `waicNimbleList` of length 1.

An object of class `waicNimbleList` of length 1.

An object of class `waicNimbleList` of length 1.

An object of class `matrix` (inherits from `array`) with 2 rows and 5 columns.

An object of class `list` of length 3.

An object of class `matrix` (inherits from `array`) with 2 rows and 5 columns.

An object of class `list` of length 3.

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `matrix` (inherits from `array`) with 500 rows and 2 columns.

An object of class `list` of length 2.

An object of class `mcmc` with 500 rows and 2 columns.

An object of class `mcmc.list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 3.

An object of class `list` of length 3.

An object of class `list` of length 3.

An object of class `list` of length 3.

`MCMC_initial_values` *Construct MCMC initial values*

Description

MCMC chains must be started with initial values for each ' parameter. Ideally, different, often random, values are used to initialize each MCMC chain. For this purpose, the function `nimble::nimbleMCMC()` accepts a list of lists; the outer list contains one list per chain and each inner list contains one named initial value per parameter. While flexible, constructing such a data structure is tedious especially if the number of chains is not fixed.

This function constructs the required list of lists data structure from the number of chains and a template list. Each inner list is a copy of the template list with functions replaced by the result of calling them without arguments. See details.

Usage

```
MCMC_initial_values(template, chains = 1)
```

Arguments

`template`	The template list. See details. (list of lists)
`chains`	The number of chains. Defaults to 1. (integer)

Details

This function constructs a list of lists data structure suitable for initializing the MCMC chains for the function `nimbleMCMC()`. The outer list contains one inner list per MCMC chain (`chains`). Each inner list is a copy of the template list (`template`); however, if a template element is a function, the corresponding value in each inner list is replaced by the result of calling the function with no arguments. The template list shall be a list of elements, each named for a parameter in the model; each element shall be either a constant initial value for all the MCMC chains or a generator function that will construct an initial value for each MCMC chain. Note that in the latter case, the generator function is called once per chain and may construct distinct initial values each time, e.g., with a random number generator. Chains can thereby be initialized with different initial values.

Value

A data structure suitable for initializing MCMC chains. See details. (list of lists)

Examples

```
chains <- 2
template <- list(
  sigma = 1,
  a0 = function(){ runif(1,-1,+1) },
  a1 = function(){ runif(1,-1,+1) })
template |> MCMC_initial_values(chains=chains)
```

MCMC_iterations *Construct the total number of MCMC iterations*

Description

A natural parameter for describing an MCMC chain is the number of samples retained for later analysis. However, the function `nimbleMCMC()` parameterizes the length of each chain in terms of the total number of samples together with the initial number of burnin iterations and the thinning interval between samples. Thus, the total number of iterations depends on three different quantities; modification of any one requires adjustment of the total number of iterations. The function `MCMC_iterations()` constructs the total number of iterations from the three core components so that their relationship is maintained.

Usage

```
MCMC_iterations(samples = 10000, burnin = 0, thin = 1)
```

Arguments

`samples`	Number of MCMC samples to retain. Defaults to 10000, same as `nimbleMCMC()`. (integer)
`burnin`	Number of initial MCMC samples to discard. Defaults to 0, same as `nimbleMCMC()`. (integer)
`thin`	Thinning interval between MCMC samples. Defaults to 1, same as `nimbleMCMC()`. (integer)

Value

The total number of MCMC iterations required to generate a set of `samples` retained samples, after discarding `burnin` samples and thinning at a rate of `thin`. (integer)

Examples

```
MCMC_iterations()
MCMC_iterations(samples=1000)
MCMC_iterations(burnin=1000)
MCMC_iterations(thin=10)
MCMC_iterations(burnin=1000,thin=10)
MCMC_iterations(samples=1000,burnin=1000,thin=10)
```

`mcmc_quantile` *Quantile(s) of a quantity across a joint distribution*

Description

`mcmc_quantile()` selects elements from a (sorted) probability distribution based upon either their quantiles or rank.

Usage

```
mcmc_quantile(df, f = NULL, p = NULL, rank = NULL, na.last = NA)
```

Arguments

`df`	A dataframe of MCMC samples, e.g., as constructed by `samples()`, or a vector of MCMC samples.
`f`	An expression returning a vector of values, one for each row of `df`. The expression may reference variables within `df` directly without quoting. Only required if `df` is a dataframe.
`p`	The cumulative probability of the elements(s) selected from the sorted expression values. Values must be in the range `[0,1]`. (numeric or numeric vector)
`rank`	The index of the element(s) selected from the sorted expression values. Values must be in the range `[1,n]`, where `n` is the number of rows in `df`; any value of `+Inf` is replaced with `n` so that the largest value can be selected independently of the size of the dataframe. (integer or integer vector)
`na.last`	Controls the treatment of `NA`s when sorting elements. If `TRUE`, missing values in the data are put last; if `FALSE`, they are put first; if `NA` (default), they are removed.

Details

`mcmc_quantile()` evaluates the expression in the data masked environment of `df`, which should be a set of MCMC samples representing an empirical (joint) distribution. The expression should return a vector of values representing an empirical distribution of interest. Alternatively, the vector of values may be given directly as the first argument, e.g., as `df`. After sorting the expression values, `mcmc_quantile()` returns either (i) the element(s) from the distribution such that Pr(X < x) <= p and Pr(X <= x) >= p (if p is not NULL) or (ii) the specific element(s) from the distribution selected by `rank` (otherwise). At least one of `p` and `rank` shall be specified as an argument.

Value

One or more elements (one per element in p or rank) from the empirical distribution given by the sorted expression values. See below for details.

Examples

```
nimble_data_m1_s0_w0_a0_c1 |> samples() |> mcmc_quantile(mu, p=c(0,0.25,0.5,0.75,1))
nimble_data_m1_s0_w0_a0_c1 |> samples() |> mcmc_quantile(mu, rank=1)
1:10 |> mcmc_quantile(rank=+Inf)
```

---

nimble_summary *Extract MCMC summaries from NIMBLE output*

Description

nimble_summary() extracts summaries of the MCMC samples from the output generated by NIMBLE.

Usage

```
nimble_summary(nimble_data)
```

Arguments

nimble_data	Output containing MCMC samples (and optionally additional information) created by running a NIMBLE model via, for example, nimble::nimbleMCMC()

Value

NIMBLE summary information if available; NULL otherwise

Examples

```
nimble_data_m1_s1_w1_a0_c1 |> nimble_summary()
```

`nimble_test` *Test NIMBLE*

Description

NIMBLE is more complicated than most R packages to install. In particular, it requires a functional C++ compiler toolchain in order to compile and run models. As a result, it is often challenging to install. The `nimble_test()` function tests whether or not the NIMBLE installation is functional by compiling a simple model.

Usage

```
nimble_test(...)
```

Arguments

`...`	Additional arguments passed to `nimble::compileNimble()`. The most likely to be of use is `showCompilerOutput=TRUE`, which instructs NIMBLE to output the C++ compiler messages. This may be helpful for debugging non-functional installations.

Details

`nimble_test()` loads the NIMBLE package and compiles a simple test model to verify that the C++ compiler toolchain is functional and accessible to NIMBLE. This is the only function in the nimbleExtra package that requires any NIMBLE functions. As a result, the rest of the package can be used without loading NIMBLE, even to facilitate the use of data structures produced by NIMBLE functions.

Value

invisible `TRUE`

Examples

```
nimble_test()
```

nimble_waic *Extract WAIC summaries from* Rhrefhttps://r-nimble.orgNIMBLE *output*

Description

nimble_waic() extracts the WAIC summaries of MCMC samples from the output generated by NIMBLE.

Usage

```
nimble_waic(nimble_data)
```

Arguments

nimble_data	Output containing MCMC samples (and optionally additional information) created by running a NIMBLE model via, for example, nimble::nimbleMCMC.

Value

nimble::WAIC information if available; NULL otherwise.

Examples

```
nimble_data_m1_s1_w1_a0_c1 |> nimble_waic()
```

n_chains *Number of MCMC chains*

Description

n_chains(x) returns the number of MCMC chains in a NIMBLE data structure.

Usage

```
n_chains(nimble_data)
```

Arguments

nimble_data	A data structure constructed by nimble::nimbleMCMC() or other similar NIMBLE function.

Value

The number of MCMC chains.

Examples

```
nimble_data_m1_s1_w0_a0_c2 |> n_chains()
```

`posterior_key` *Construct a primary key for MCMC samples in posterior::draws_df format*

Description

Construct a primary key for MCMC samples in posterior::draws_df format

Usage

```
posterior_key()
```

Value

A tidyselect object representing the primary key in the posterior::draws_df format

Examples

```
nimble_data_m1_s1_w0_a0_c1 |> samples() |>
  tidyr::pivot_longer(!posterior_key())
```

`posterior_summary_key` *Construct a primary key for MCMC samples in posterior::draws_summary format*

Description

Construct a primary key for MCMC samples in posterior::draws_summary format

Usage

```
posterior_summary_key()
```

Value

A tidyselect object representing the primary key in the posterior::draws_summary format

Examples

```
nimble_data_m1_s1_w0_a0_c1 |> samples() |> summary() |>
  tidyr::pivot_longer(!posterior_summary_key())
```

print_dataframe *Print information about a dataframe*

Description

Print some or all of the following information about a dataframe:

- A title
- The result of calling utils::str()
- The result of calling summary()
- The entire dataframe (if small enough) or n rows from the head and tail

Usage

```
print_dataframe(
  data,
  ...,
  n = 6,
  output = "str,summary,df",
  as_dataframe = FALSE,
  sep = ""
)
```

Arguments

data	Dataframe to print (data.frame)
...	Optional information for composing the title. Each element is passed as the ellipsis argument to cat(). If empty, no title is generated.
n	Number of head/tail rows to print. If df has no more than 2n rows, the entire dataframe will be printed. Defaults to 6. (integer)
output	Which components of the dataframe to include in the printed output (see details). Defaults to "str, summary, df". (character, comma-delimited list)
as_dataframe	Convert the dataframe argument df to a data.frame prior to output. (logical)
sep	Strings to append after each element in the title (i.e., in the ellipsis argument). Passed to cat(). Defaults to "". (character)

Details

print_dataframe() prints various summaries of a dataframe following an optional title. Which summaries are printed is determined by the value of the argument output, which must be a character string containing a comma-delimited list of any combination (including none) of the following:

- "str" The result of calling str() is output.
- "summary" The result of calling summary() is output.
- "head" The result of calling head(n=n) is output.

- "tail" The result of calling tail(n=n) is output.
- "df" Equivalent to "head,tail".

If data is not actually a dataframe, at most the output will include a title and the result of calling str(df).

Value

The dataframe argument df invisibly.

Examples

```
nimble_data_m1_s1_w0_a0_c2 |> samples() |> print_dataframe("===> 2 chains:")
```

---

print_figures *Render a list of figures*

Description

print_figures() renders a figure for each combination of (i) element in figures, (ii) element in format, and (iii) element in output. These figures may be pages within a single file, individual files, or displayed on the default (or current) graphics device. The output argument selects which of the possibilities are rendered (see details).

- A single file containing all the figures of each format.
- A set of individual files, one per figure for each format.
- A set of figures on the default (or current) graphic device.

The format of files created is determined by the elements in the format argument, so it is possible to create multiple files of each figure but of different formats.

Usage

```
print_figures(
  figures,
  basename = NULL,
  format = "pdf",
  output = "device",
  width = NA,
  height = NA,
  ...
)
```

Arguments

figures	The figures to render. Elements of the list may be (i) figures created by ggplot2::ggplot() or (ii) scaled figures created with the function scaled(); anything else in the list is ignored. (list, with named elements if output contains "files")
basename	The, possibly optional (see below), basename used with a format-specific extension to construct filenames. (character)
format	The graphical format(s), i.e., filename extension(s), for the rendered file(s). If a vector of formats, each figure will be rendered in each format. Possible vector formats are "pdf", "ps", and "svg"; possible raster formats are "jpeg" and "png". Note that the meaning of width, height, and perhaps other arguments may vary among these graphic devices, so using multiple formats simultaneously in a meaningful way requires care. Defaults to "pdf" for single Portable Document Format files. (character; comma-delimited list)
output	Select the type of files or figures to create; see details. (character; comma-delimited list)
width, height	The default width and height of the figures. Either dimension may be overruled for any specific figure by inserting a scaled version (e.g., created by the function scaled()) into the list figures. Defaults to NA and NA, which select the default dimensions for each graphic device. (numeric)
...	Additional arguments passed to the graphical device(s) for "bundled" output and to ggplot2::ggsave() for "files" output.

Details

print_figures renders a list of figures, each created by calls to ggplot2::ggplot(), as a set of files or figures on the default (or current) graphic device. The formats of the files are determined by elements in format, which is also used as the filename extension. The set of files or figures is determined by output, which is a comma-delimited list of any of the following values:

- "all" Render all of the types of figures listed below.
- "bundled" Render each figure with print() on a graphic device determined by each element of format. All figures are printed sequentially to each format-specific graphic device. For graphic devices that support it (e.g., PDF), the result is figures on distinct pages within a single file. The name of the file is taken from basename, which must be given, and the format-specific extension. For example, the following will create the files basename.pdf and basename.png, each of which will contain all the figures as rendered by print().

```
figures |>
  print_figures("basename",format="pdf,png",output="bundled")
```

- "device" Render each figure on the default (or current) graphic device with the print() function. This is the default and replicates the handling of ggplot2 figures when created directly in an interactive environment.
- "files" Render each figure in its own file using ggplot2::ggsave(). In this case, figures shall be a named list, and the name of each file is taken from the name of the corresponding list element, optionally prefixed by "<basename>-" if given. The filename extension is taken from elements in format.

Value

The list of figures invisibly.

Examples

```
x <- runif(10)
y1 <- runif(10)
y2 <- runif(10)

figure_1 <- ggplot2::ggplot(data.frame(x=x,y=y1),ggplot2::aes(x,y)) +
            ggplot2::geom_point()
figure_2 <- ggplot2::ggplot(data.frame(x=x,y=y2),ggplot2::aes(x,y)) +
            ggplot2::geom_point()

l1 <- list(figure=figure_1)
l2 <- list(figure_1=figure_1, figure_2=figure_2)

# same figure, two different heights
l3 <- list(figure_1=figure_1, figure_1_short=scaled(figure_1, height=4))

l1 |> print_figures(output="files")                       # 1 pdf file
l2 |> print_figures(output="files")                       # 2 pdf files
l3 |> print_figures(output="files", format="pdf")         # 2 pdf files
l3 |> print_figures(output="bundled", format="pdf,png")   # no files; basename missing

file.remove(c("figure.pdf",                               # l1
  "figure_1.pdf", "figure_2.pdf",                         # l2
  "figure_1.pdf", "figure_1_short.pdf"                    # l3
  ))
```

`probability` *Probability of satisfying a predicate across a joint distribution*

Description

`probability()` calculates the probability of a predicate expression across a set of MCMC samples representing the joint distribution of a set of quantities.

Usage

```
probability(df, predicate)
```

Arguments

`df`	A dataframe of MCMC samples, e.g., as constructed by `samples()`.
`predicate`	A predicate expression, i.e., returning a vector of logicals, applying a condition to each row of `df`. The expression may reference variables within `df` directly without quoting.

Details

probability() evaluates the predicate expression in the data masked environment of df, which should be a set of MCMC samples representing an empirical (joint) distribution. The expression should return a vector of logical values indicating whether each row of df satisfies the condition. probability() returns the probability of satisfying the condition of the predicate function as the fraction of rows in df for which the condition is satisfied.

Value

The probability of satisfying the predicate condition across rows of df.

Examples

```
nimble_data_m1_s0_w0_a0_c1 |> samples() |> probability(mu < 0.5)
nimble_data_m1_s0_w0_a0_c1 |> samples() |> probability(mu < 0.5 & sigma < 1)
```

---

samples *Extract a chain of MCMC samples from NIMBLE output*

Description

samples() extracts, as a wide data.frame, the MCMC samples for one or all chains from the output generated by, for example, nimble::nimbleMCMC().

Usage

```
samples(nimble_data, chain = NA)
```

Arguments

nimble_data	Output containing MCMC samples (and optionally additional information) created by running a NIMBLE model with, for example, nimble::nimbleMCMC().
chain	Index(es) of chain(s) to extract. Defaults to NA, which extracts all samples. Any subset of chains (including duplications) may be extracted. (integer or integer vector)

Details

The dataframe returned has one row per MCMC sample in the set of chains selected and is thus a wide format. Use tidyr::pivot_longer() to transform this wide dataframe into a long one.

The dataframe returned is in the posterior::draws_df format], which has one column for each of the random variables monitored by NIMBLE. Additionally, it includes the following metadata columns, which are often hidden:

- .chain An identifier for the MCMC chain
- .iteration An identifier for the MCMC sample within the chain
- .draw A unique identifier for the MCMC sample across all chains

Value

A dataframe containing the extracted MCMC samples. (posterior::draws_df)

Examples

```
nimble_data_m1_s1_w0_a0_c1 |> samples()
nimble_data_m1_s1_w0_a0_c1 |> samples() |> summary()
```

`scaled` *Add scale information to a figure*

Description

Combine scale information with a figure so that the figure can be rendered differently by `print_figures()` from others in the same list. Note that figures can be scaled independently only if they are rendered as individual files; otherwise, the default scaling is applied to all figures.

Usage

```
scaled(figure, width = NA, height = NA)
```

Arguments

`figure`	A figure created by `ggplot2::ggplot()` or the result of calling `scaled()` on one.
`width, height`	The width and height of the scaled figure. Defaults to `NA` and `NA`, which select the default dimension for each graphic device. (numeric)

Value

An object of class scaled containing the following named elements:

- `figure` The figure created by `ggplot2::ggplot()`.
- `width,height` The scaled width and height of the figure.

Examples

```
x <- runif(10)
y <- runif(10)
figure <- ggplot2::ggplot(data.frame(x=x,y=y),ggplot2::aes(x,y)) +
          ggplot2::geom_point()
scaled(figure, width=7)
scaled(figure, height=4)
scaled(figure, width=7, height=4)
```

`select_samples` *Select an evenly arrayed subset of MCMC samples from a dataframe*

Description

`select_samples()` selects a subset of rows, from a, possibly grouped, dataframe. The selected rows are evenly arrayed throughout the dataframe.

Usage

```
select_samples(data, n = NULL, prop = NULL)
```

Arguments

`data`	A dataframe from which rows are to be selected. (dataframe)
`n`	The number of rows to select from `data`. If `data` is a grouped dataframe, the number of rows to select per group. If less than zero, the number of rows to remove. Default `NULL`. (integer)
`prop`	The fraction (proportion) of rows to select from `data`. If `data` is a grouped dataframe, the fraction of rows to select per group. If less than zero, the fraction of rows to remove. The number of rows will be rounded to generate an integer number of rows to select or remove. Default `NULL`. (numeric)

Details

Select a subset of rows from a dataframe. If `n` is given and positive, the resulting dataframe will contain at most `n` rows (or `n` rows per group for a grouped dataframe); if `n` is given and negative, the resulting dataframe will be reduced by at most `n` rows (or `n` rows per group for a grouped dataframe). Otherwise (i.e., `n` is `NULL`) if `prop` is given, the resulting dataframe (or group) will be reduced by or retain the given fraction of rows. If neither `n` nor `prop` are given, the resulting dataframe will be empty. In all cases, the selected subset of rows will be evenly arrayed across the rows in the original dataframe.

Value

An object of the same type as `data` containing a subset of rows.

Examples

```
nimble_data_m1_s1_w0_a0_c1 |> samples() |> select_samples()
nimble_data_m1_s1_w0_a0_c1 |> samples() |> select_samples(10)
nimble_data_m1_s1_w0_a0_c1 |> samples() |> select_samples(-400)
nimble_data_m1_s1_w0_a0_c1 |> samples() |> select_samples(prop=0.1)
nimble_data_m1_s1_w0_a0_c1 |> samples() |> select_samples(prop=-0.9)

nimble_data_m1_s0_w0_a0_c2 |> samples() |>
  dplyr::group_by(.chain) |>
  select_samples(n=3)
```

stats *Extract MCMC summary statistics from MCMC samples*

Description

stats() extracts a selected summary statistic for all variables from a set of MCMC samples.

Usage

```
stats(mcmc_data, stat = "mean", type = "vector")
```

Arguments

mcmc_data	A set of MCMC samples, e.g., created by samples(). (dataframe)
stat	The name of the summary statistic to extract. This name shall be either one of the columns of the summary data frame or any of the values "min", "max", or "cv", i.e., the coefficient of variation. Defaults to "mean". (character)
type	Indicate the result type from one of "numeric" (a numeric vector; default) or "df" (a single-row dataframe). In either case, the elements/columns are named as the variables in mcmc_data. (character)

Value

A named vector or dataframe of the selected summary statistic.

Examples

```
nimble_data_m1_s1_w1_a0_c1 |> samples() |> stats()
nimble_data_m1_s1_w1_a0_c1 |> samples() |> stats("mean")
nimble_data_m1_s1_w1_a0_c1 |> samples() |> stats("sd")
nimble_data_m1_s1_w1_a0_c1 |> samples() |> stats("cv")
nimble_data_m1_s1_w1_a0_c1 |> samples() |> stats("rhat")
nimble_data_m1_s1_w1_a0_c1 |> samples() |> stats(type="df")
```

stat_mean_abline *A ggplot2::ggplot2 stat function for the mean linear relation*

Description

The function stat_mean_abline() may be called, possibly with a geom argument, to replace the aesthetic mappings intercept and slope with the corresponding means of a set of MCMC samples prior to plotting with the geom. Alternatively, some geoms may be called with stat="mean_abline" to accomplish the same. Generally, stat_mean_abline() greatly simplifies the process of adding summary statistics to a figure.

Usage

```
stat_mean_abline(
  mapping = NULL,
  data = NULL,
  geom = "abline",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: • If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot2::ggplot(). • A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. • A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use to display the data for this layer; defaults to "abline". When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use ggplot2::geom_point(), give the geom as "point". For more information and other ways to specify the geom, see the layer geom documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as ggplot2::position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use ggplot2::position_jitter(), give the position as "jitter". For more information and other ways to specify the position, see the layer position documentation.

na.rm	Additional parameter passed as params to the geom and stat.
show.legend	Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted. (logical)
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. ggplot2::annotation_borders().
...	Additional arguments passed to ggplot2::layer().

Details

stat_mean_abline() calculates the mean intercept and slope of MCMC samples to use for locating graphical elements defined by a geom.

For the details of stat functions, see

https://bookdown.org/rdpeng/RProgDA/building-new-graphical-elements.html.

Value

The result of calling ggplot2::layer().

Aesthetics

The following aesthetics are required:

- intercept and slope – The quantities to summarize

Calculated quantities

The following quantities are calculated by stat_mean_abline():

- intercept – The mean of the required intercept aesthetic
- slope – The mean of the required slope aesthetic

Examples

```
nimble_data_m1_s0_w0_a0_c1 |> samples() |>
  ggplot2::ggplot() +
  ggplot2::geom_point(ggplot2::aes(x=mu, y=sigma)) +
  ggplot2::geom_abline(ggplot2::aes(intercept=mu, slope=sigma), stat="mean_abline")

nimble_data_m1_s0_w0_a0_c1 |> samples() |>
  ggplot2::ggplot() +
  ggplot2::geom_point(ggplot2::aes(x=mu, y=sigma)) +
  stat_mean_abline(ggplot2::aes(intercept=mu, slope=sigma))

nimble_data_m1_s0_w0_a0_c1 |> samples() |>
  ggplot2::ggplot() +
```

```
ggplot2::geom_point(ggplot2::aes(x=mu, y=sigma)) +
stat_sample_mean(ggplot2::aes(intercept=mu, slope=sigma), geom="abline")
```

`stat_mean_hline` *A ggplot2::ggplot2 stat function for the sample mean*

Description

The function `stat_mean_hline()` may be called, possibly with a geom argument, to replace the aesthetic mappings `yintercept` with the means of a set of MCMC samples prior to plotting with the geom. Alternatively, some geoms may be called with `stat="mean_hline"` to accomplish the same. Generally, `stat_mean_hline()` greatly simplifies the process of adding summary statistics to a figure.

Usage

```
stat_mean_hline(
  mapping = NULL,
  data = NULL,
  geom = "hline",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `ggplot2::aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: • If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot2::ggplot()`. • A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. • A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).
`geom`	The geometric object to use to display the data for this layer; defaults to `"line"`. When using a `stat_*()` function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following:

- A Geom ggproto subclass, for example GeomPoint.
- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `ggplot2::geom_point()`, give the geom as `"point"`.

For more information and other ways to specify the geom, see the layer geom documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as `ggplot2::position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `ggplot2::position_jitter()`, give the position as `"jitter"`.

For more information and other ways to specify the position, see the layer position documentation.

`na.rm` Additional parameter passed as `params()` to the geom and stat.

`show.legend` Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted. (logical)

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `ggplot2::annotation_borders()`.

`...` Additional arguments passed to `ggplot2::layer()`.

Details

`stat_mean_hline()` calculates the mean y intercept of MCMC samples to use for locating graphical elements defined by a geom.

For the details of stat functions, see

https://bookdown.org/rdpeng/RProgDA/building-new-graphical-elements.html.

Value

The result of calling `ggplot2::layer()`.

Aesthetics

The following aesthetics are required:

- yintercept – The quantity to summarize

Calculated quantities

The following quantities are calculated by `stat_mean_hline()`:

- yintercept – The mean of the required `yintercept`

Examples

```
nimble_data_m1_s0_w0_a0_c1 |> samples() |>
  ggplot2::ggplot() +
  ggplot2::geom_point(ggplot2::aes(x=mu, y=sigma)) +
  ggplot2::geom_hline(ggplot2::aes(yintercept=sigma), stat="mean_hline")

nimble_data_m1_s0_w0_a0_c1 |> samples() |>
  ggplot2::ggplot() +
  ggplot2::geom_point(ggplot2::aes(x=mu, y=sigma)) +
  stat_mean_hline(ggplot2::aes(yintercept=sigma))

nimble_data_m1_s0_w0_a0_c1 |> samples() |>
  ggplot2::ggplot() +
  ggplot2::geom_point(ggplot2::aes(x=mu, y=sigma)) +
  stat_sample_mean(ggplot2::aes(yintercept=sigma), geom="hline")
```

---

stat_mean_vline *A ggplot2::ggplot2 stat function for the mean vline, i.e., x intercept*

Description

The function `stat_mean_vline()` may be called, possibly with a geom argument, to replace the aesthetic mappings `xintercept` with the means of a set of MCMC samples prior to plotting with the geom. Alternatively, some geoms may be called with `stat="mean_vline"` to accomplish the same. Generally, `stat_mean_vline()` greatly simplifies the process of adding summary statistics to a figure.

Usage

```
stat_mean_vline(
  mapping = NULL,
  data = NULL,
  geom = "vline",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: • If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot2::ggplot(). • A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. • A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use to display the data for this layer; defaults to "vline". When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use ggplot2::geom_point(), give the geom as "point". For more information and other ways to specify the geom, see the layer geom documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as ggplot2::position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". For more information and other ways to specify the position, see the layer position documentation.
na.rm	Additional parameter passed as params to the geom and stat.
show.legend	Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted. (logical)
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. ggplot2::annotation_borders().
...	Additional arguments passed to ggplot2::layer().

Details

stat_mean_vline() calculates the mean x intercept of MCMC samples to use for locating graphical elements defined by a geom.

For the details of stat functions, see

https://bookdown.org/rdpeng/RProgDA/building-new-graphical-elements.html.

Value

The result of calling ggplot2::layer().

Aesthetics

The following aesthetics are required:

- xintercept – The quantity to summarize

Calculated quantities

The following quantities are calculated by stat_sample_mean():

- xintercept – The mean of the required xintercept aesthetic

Examples

```
nimble_data_m1_s0_w0_a0_c1 |> samples() |>
  ggplot2::ggplot() +
  ggplot2::geom_point(ggplot2::aes(x=mu, y=sigma)) +
  ggplot2::geom_vline(ggplot2::aes(xintercept=mu), stat="mean_vline")

nimble_data_m1_s0_w0_a0_c1 |> samples() |>
  ggplot2::ggplot() +
  ggplot2::geom_point(ggplot2::aes(x=mu, y=sigma)) +
  stat_mean_vline(ggplot2::aes(xintercept=mu))

nimble_data_m1_s0_w0_a0_c1 |> samples() |>
  ggplot2::ggplot() +
  ggplot2::geom_point(ggplot2::aes(x=mu, y=sigma)) +
  stat_sample_mean(ggplot2::aes(xintercept=mu), geom="vline")
```

---

stat_quantile_line *A* ggplot *stat function for a single quantile*

Description

stat_quantile_line() is a ggplot2::ggplot2 stat function to calculate a single quantile as a function of either x or y, depending on the orientation argument.

Usage

```
stat_quantile_line(
  mapping = NULL,
  data = NULL,
  geom = "line",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  p = NULL,
  rank = NULL,
  orientation = "horizontal",
  na.last = NA,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: • If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot2::ggplot(). • A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. • A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use to display the data for this layer; defaults to "line". When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use ggplot2::geom_point(), give the geom as "point". For more information and other ways to specify the geom, see the layer geom documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as ggplot2::position_jitter(). This method allows for passing extra arguments to the position.

- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `ggplot2::position_jitter()`, give the position as `"jitter"`.

For more information and other ways to specify the position, see the layer position documentation.

`na.rm`	Additional parameter passed as `params` to the geom and stat.
`show.legend`	Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted. (logical)
`inherit.aes`	If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `ggplot2::annotation_borders()`.
`p`	The value of the quantile to construct. Defaults to `NULL`. (numeric)
`rank`	The rank of the quantile to construct. Defaults to `NULL`. (numeric)
`orientation`	The orientation of the transformation, which defines which variable, `x` or `y`, is taken as the independent variable. A value of `horizontal` (default) calculates quantiles of `y` as a function of `x`; a value of `vertical` calculates the opposite. (character)
`na.last`	Controls the treatment of `NA`s when calculating quantiles. If `TRUE`, missing values in the data are put last; if `FALSE`, they are put first; if `NA` (default), they are removed.
`...`	Additional arguments passed to `ggplot2::layer()`.

Details

For the details of stat functions, see

https://bookdown.org/rdpeng/RProgDA/building-new-graphical-elements.html.

Value

The result of calling `ggplot2::layer()`.

Aesthetics

The following aesthetics are required:

- x and y – The quantities to summarize. Depending on the value of `orientation`, one is taken to be the independent variable, and quantiles of the other are calculated.

Calculated quantities

The following quantities are calculated by `stat_quantile_line()`:

- x or y – The calculated quantile. Which is calculated is determined by the value of `orientation`.

Examples

```
df <- data.frame(x=rep(0:10,100)) |>
   dplyr::mutate(y=rnorm((10+1)*100))

ggplot2::ggplot(df, ggplot2::aes(x,y)) +
  stat_quantile_line(p=0.25) +
  stat_quantile_line(p=0.50, color="blue") +
  stat_quantile_line(p=0.75)

ggplot2::ggplot(df, ggplot2::aes(x,y)) +
  stat_quantile_line(geom="step", p=0.25) +
  stat_quantile_line(geom="step", p=0.50, color="blue") +
  stat_quantile_line(geom="step", p=0.75)
```

---

stat_quantile_ribbon *A ggplot2::ggplot2 stat function for two quantiles, min and max, defining the bounds of a ribbon*

Description

`stat_quantile_ribbon()` is a ggplot2::ggplot2 stat function to calculate two quantiles as a function of either `x` or `y`, depending on the `orientation` argument. The two quantiles define the lower and upper bounds of a ribbon. One of `quantiles` or `ranks` shall be defined.

Usage

```
stat_quantile_ribbon(
  mapping = NULL,
  data = NULL,
  geom = "ribbon",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  quantiles = NULL,
  ranks = NULL,
  orientation = "horizontal",
  na.last = NA,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `ggplot2::aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.

`data` The data to be displayed in this layer. There are three options:

- If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot2::ggplot()`.
- A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
- A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).

`geom` The geometric object to use to display the data for this layer; defaults to "ribbon". When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following:

- A Geom ggproto subclass, for example GeomPoint.
- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `ggplot2::geom_point()`, give the geom as `"point"`.

For more information and other ways to specify the geom, see the layer geom documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as `ggplot2::position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `ggplot2::position_jitter()`, give the position as `"jitter"`.

For more information and other ways to specify the position, see the layer position documentation.

`na.rm` Additional parameter passed as `params` to the geom and stat.

`show.legend` Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted. (logical)

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `ggplot2::annotation_borders()`.

`quantiles` The values of the two quantiles to construct. Defaults to `NULL`. (numeric vector)

`ranks` The ranks of the two quantiles to construct. Defaults to `NULL`. (numeric vector)

`orientation` The orientation of the transformation, which defines which variable, `x` or `y`, is taken as the independent variable. A value of `horizontal` (default) calculates quantiles of `y` as a function of `x`; a value of `vertical` calculates the opposite. (character)

na.last	Controls the treatment of NAs when calculating quantiles. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA (default), they are removed.
...	Additional arguments passed to ggplot2::layer().

Details

For the details of stat functions, see

https://bookdown.org/rdpeng/RProgDA/building-new-graphical-elements.html.

Value

The result of calling ggplot2::layer().

Aesthetics

The following aesthetics are required:

- x and y – The quantities to summarize. Depending on the value of orientation, one is taken to be the independent variable, and quantiles of the other are calculated.

Calculated quantities

The following quantities are calculated by stat_quantile_ribbon():

- xmin, xmax or ymin, ymax – The calculated quantiles, with *min corresponding to the first quantile and *max corresponding to the second. Which is calculated is determined by the value of orientation.

Examples

```
df <- data.frame(x=rep(0:10,100)) |>
   dplyr::mutate(y=rnorm((10+1)*100))

ggplot2::ggplot(df, ggplot2::aes(x,y)) +
  stat_quantile_ribbon(quantiles=c(0.025,0.975),fill="gray90") +
  stat_quantile_ribbon(quantiles=c(0.1,0.9),fill="gray70") +
  stat_quantile_ribbon(quantiles=c(0.25,0.75),fill="gray50")
```

---

stat_sample_mean *A ggplot2::ggplot2 stat function for the sample mean*

Description

The function stat_sample_mean() may be called, possibly with a geom argument, to replace aesthetic mappings with the corresponding means of a set of MCMC samples prior to plotting with the geom. Alternatively, some geoms may be called with stat="sample_mean" to accomplish the same; notably, this is not possible for ggplot2::geom_hline(), ggplot2::geom_vline(), or ggplot2::geom_abline(), because of their special use of the aesthetic mappings. Instead, use the corresponding stat_mean_*() functions; see details for additional information. Generally, stat_sample_mean() greatly simplifies the process of adding summary statistics to a figure.

Usage

```
stat_sample_mean(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: • If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot2::ggplot(). • A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. • A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use to display the data for this layer; defaults to "line". When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". For more information and other ways to specify the geom, see the layer geom documentation.

position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as ggplot2::position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use [ggplot2::position_jitter()[, give the position as "jitter". For more information and other ways to specify the position, see the layer position documentation.
na.rm	Additional parameter passed as params to the geom and stat.
show.legend	Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted. (logical)
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. ggplot2::annotation_borders().
...	Additional arguments passed to ggplot2::layer().

Details

stat_sample_mean() calculates the mean summary statistic of MCMC samples to use for locating graphical elements defined by a geom. To simplify use, the type of layer returned depends on the geom argument; thus, stat_sample_mean() may be called with any appropriate geom argument. If geom is "hline", "vline", or "abline", stat_sample_mean() is equivalent to the corresponding stat_mean_*() function; otherwise, it returns a stat layer that calculates the mean of the x or y aesthetics. However, "sample_mean" cannot be used as a stat argument for those geoms; instead, use the corresponding "mean_*" stats.

For the details of stat functions, see

https://bookdown.org/rdpeng/RProgDA/building-new-graphical-elements.html.

Value

The result of calling ggplot2::layer(). See details.

Aesthetics

The following aesthetics are required:

- x or y – The quantity to summarize

Calculated quantities

The following quantities are calculated by stat_sample_mean():

- x – The mean of the x aesthetic to summarize
- y – The mean of the y aesthetic to summarize

Examples

```
nimble_data_m1_s0_w0_a0_c1 |> samples() |>
  ggplot2::ggplot(ggplot2::aes(x=mu, y=sigma)) +
  ggplot2::geom_point() +
  ggplot2::geom_point(stat="sample_mean",color="red", size=3)

nimble_data_m1_s0_w0_a0_c1 |> samples() |>
  ggplot2::ggplot(ggplot2::aes(x=mu, y=sigma)) +
  ggplot2::geom_point() +
  stat_sample_mean(color="red", size=3)
```

`stat_sample_quantile` *A ggplot2::ggplot2 stat function for a single quantile*

Description

`stat_sample_quantile()` is a ggplot2::ggplot2 stat function to calculate a single quantile as a function of either `x` or `y`, depending on which of the two aesthetics is defined.

Usage

```
stat_sample_quantile(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  p = NULL,
  rank = NULL,
  na.last = NA,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `ggplot2::aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: • If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot2::ggplot()`. • A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

- A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).

`geom` The geometric object to use to display the data for this layer; defaults to `"line"`. When using a `stat_*()` function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following:

- A Geom ggproto subclass, for example GeomPoint.
- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`.

For more information and other ways to specify the geom, see the layer geom documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as `ggplot2::position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.

For more information and other ways to specify the position, see the layer position documentation.

`na.rm` Additional parameter passed as `params` to the geom and stat.

`show.legend` Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted. (logical)

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

`p` The value of the quantile to construct. Defaults to `NULL`. (numeric)

`rank` The rank of the quantile to construct. Defaults to `NULL`. (numeric)

`na.last` Controls the treatment of `NA`s when calculating quantiles. If `TRUE`, missing values in the data are put last; if `FALSE`, they are put first; if `NA` (default), they are removed.

`...` Additional arguments passed to `ggplot2::layer()`.

Details

For the details of stat functions, see

https://bookdown.org/rdpeng/RProgDA/building-new-graphical-elements.html.

Value

The result of calling ggplot2::layer().

Aesthetics

The following aesthetics are required:

- x or y – The quantities to summarize. Both may be specified, in which case the same quantile is calculated for both dimensions.

Calculated quantities

The following quantities are calculated by stat_sample_quantile():

- x or y – The calculated quantile. Which is calculated is determined by the aesthetic.
- xintercept or yintercept – The calculated quantile. Which is calculated is determined by the aesthetic.

Examples

```
nimble_data_m1_s0_w0_a0_c1 |> samples() |>
  ggplot2::ggplot(ggplot2::aes(x=mu, y=sigma)) +
  ggplot2::geom_point() +
  stat_sample_quantile(p=0.5, color="red", size=3)
```

Index