Package 'rando'

February 16, 2021

Type Package

Title Context Aware Random Numbers

Version 0.2.0

Description Provides random number generating functions that are much more context aware than the built-in functions. The functions are also much safer, as they check for incompatible values, and more reproducible.

Language en-GB

License MIT + file LICENSE

URL https://github.com/MyKo101/rando

BugReports https://github.com/MyKo101/rando/issues

Imports dplyr, glue, rlang, stats, tibble

Suggests spelling, covr, testthat

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

NeedsCompilation no

Author Michael Barrowman [cre, aut]

Maintainer Michael Barrowman <myko101ab@gmail.com>

Repository CRAN

Date/Publication 2021-02-16 15:40:02 UTC

R topics documented:

`rando-package` *Context Aware Random Number Generation*

Description

rando is designed to make random number generation easier by providing the ability to set a default number of numbers to generate or to assess the context in which the functions are being ran.

`as_function` *Convert to function*

Description

This function is a wrapper around `rlang::as_function()` which adds a two extra features:

- formulas can use `.t` in place of `.x` to be easier to understand in time-based functions
- functions can take additional named arguments.

Usage

```
as_function(x, env = parent.frame())
```

Arguments

`x`	a function or formula, see `rlang::as_function()` for more information
`env`	Environment in which to fetch the function in case `x` is a string

Value

Either:

- the function as it is passed to `as_function()`, whether as a string or a name
- the function derived from a formula, where the first argument is passed as `.`, `.x` or `.t`, the second argument is passed as `.y` and any other named arguments are passed as they are named

Examples

```
f1 <- as_function(mean)
f1(1:10)

f2 <- as_function("sum")
f2(1,2,3)

f3 <- as_function(~.x + 1)
f3(9)

f4 <- as_function(~ .t + 1)
f4(10)

f5 <- as_function(~.x + .y)
f5(1,2)

f6 <- as_function(~ .t + alpha)
f6(10, alpha = 2)
```

`blueprint` *Blueprint a Dataset*

Description

Allows for the generation of population based on a prescribed set of rando functions.

Usage

```
blueprint(...)

is_blueprint(bp)
```

Arguments

`...`	arguments used to generate the blueprint, see Examples.
`bp`	Object to check

Value

A function that will produce a tibble, which matches the blueprint that was provided. The generated function will take the following arguments:

- `...` - any arguments that are used within the blueprinting
- `n` - the number of rows that the resulting tibble should be
- `.seed` - the random seed to set before generating the data

`is_blueprint()` simply checks whether a function is a blueprinting function or not and returns a logical.

Examples

```
make_tbl <- blueprint(
  x = r_norm(),
  y = r_norm()
)

make_tbl(n = 2)

make_tbl(n = 5)

# Blueprints can use additional parameters:
make_tbl2 <- blueprint(
  x = r_norm(mean = x_mu),
  y = r_unif(min = y_min, max = y_max)
)

# Which are simply passed to the generated function
```

```
make_tbl2(x_mu = 10, y_min = -10, y_max = -5)

is_blueprint(make_tbl)
```

---

bp_where *Blueprint based on a condition*

Description

Runs a blueprint function where a condition is true, otherwise returns NA values

Usage

```
bp_where(condition, bp, ...)
```

Arguments

condition	Condition to check before evaluating. Results will be given where this is TRUE, and NA when this is FALSE
bp	Blueprint function to run based on the condition
...	arguments passed on to Blueprint, such as .seed

Value

a tibble

Examples

```
make_tbl <- blueprint(
  x = r_norm(),
  y = r_unif()
)

set_n(10)
i <- r_lgl()

bp_where(i, make_tbl)

df <- tibble::tibble(
  id = 1:10,
  cnd = r_lgl()
)
dplyr::mutate(df, bp_where(cnd, make_tbl))
```

default_n *Find the Default Value for n in Context*

Description

Checks for various information surrounding the call to this function to figure out what value for n should be used

Usage

```
default_n(...)

blueprint_n()

tibble_n()

dplyr_n()

args_n(...)
```

Arguments

...	parameters to check the lengths of

Details

The `default_n()` function will run through the other functions found here until it finds a viable value for n.

It first checks for contxt to see if calls external to `default_n()` indicate which value should be used:

- `blueprint_n()` - Checks if the function is being called within a blueprinting function, and returns the value supplied to that function, see `blueprint()`.
- `tibble_n()` - Checks if the function is being called within the declaration of a tibble. It then checks the lengths of the other arguments being passed to the call. If you want to specify how many rows should be generate you can use the `.rows` argument in your `tibble()` call, see `tibble()`
- `dplyr_n()` - Checks if the function is being used within a `dplyr` verb, if so, it returns the value of `n()`

It then checks the lengths of the arguments supplied via ..., if there is a discrepancy between these arguments and the context aware value found above, it will throw an error.

If all the above values return `1` or `NULL`, we then check for a global n assigned by `set_n()`, if none is set then `default_n()` will return `1`.

Value

The context aware value for n

Examples

```
# Global Values:
set_n(NULL)
default_n()
set_n(10)
default_n()

# In a blueprint:
bp <- blueprint(x=r_norm(),n=default_n())
bp(n=7)
bp <- blueprint(x=r_norm(),n=blueprint_n())
bp(n=8)

# In a tibble:
tibble::tibble(id = 1:3, n = default_n())
tibble::tibble(id = 1:5, n = tibble_n())

# In a dplyr verb:
df <- tibble::tibble(id = 1:4)
dplyr::mutate(df, n = default_n())
dplyr::mutate(df, n = dplyr_n())

# From arguments:
default_n(1:5)
default_n(1:5,c("a","b","c","d","e"))
args_n(1:3,c("a","b","d"))
args_n(1:3, 1:4)

## Not run:
default_n(1:3, 1:4)
tibble::tibble(id=1:5,n=default_n(1:4))

## End(Not run)
```

---

extract_dots *Extract the ellipsis inside a function*

Description

Allow the named entries in ... to be used easily within a function by attaching them to the function's environment

Usage

```
extract_dots()
```

Value

No return value, called for it's side effect

Examples

```
f <- function(...) {
  a + b
}

## Not run:
# Throws an error because a and b are trapped inside `...`
 f(a = 1, b = 2)

## End(Not run)

f <- function(...) {
  extract_dots()
  a + b
}
f(a = 1, b = 2)
```

`is_wholenumber` *Check if a Number is Whole*

Description

The built-in function `is.integer()` will check if a number is of the `integer` class. However, we would usually want a function that can check if a number is a *whole number*. It is also vectorised over the input.

Usage

```
is_wholenumber(x, tol = .Machine$double.eps^0.5)
```

Arguments

`x`	Number to check
`tol`	tolerance to check the values

Value

A logical vector the same length as `x`

Examples

```
is.integer(2)
is_wholenumber(2)

is.integer(seq(2, 3, 0.25))
is_wholenumber(seq(2, 3, 0.25))
```

`logit` *The logit and inverse logit functions*

Description

Calculates the logit or the inverse logit of a value

Usage

```
logit(prob, base = exp(1))

invlogit(alpha, base = exp(1))
```

Arguments

`prob`	vector of probabilities
`base`	base of the logarithmic function to use
`alpha`	vector of values to find the inverse logit of

Value

A numeric vector

Examples

```
logit(0.5)

logit(seq(0.01, 0.99, 0.01))
invlogit(-10:10)
```

`match.call2` *Alternate Parametrisation of* `match.call()`

Description

Alters the built-in function `match.call()` by providing an additional argument which means that by default a user can specify how far up the call stack they want to match a call of. See match.call() for more details.

Usage

```
match.call2(
  n = 0L,
  definition = sys.function(sys.parent(n + 1L)),
  call = sys.call(sys.parent(n + 1L)),
  expand.dots = TRUE,
  envir = parent.frame(n + 3L)
)
```

Arguments

`n`	How far up the call-stack they would like to extract. The default, `n=0` produces the same result as `match.call()` so this can be inserted wherever `match.call()` is used.
`definition`	a function, by default the function from which `match.call2()` is called.
`call`	an unevaluated call to the function specified by `definition`, as generated by `call`
`expand.dots`	logical. Should arguments matching `...` in the call be included or left as a `...` argument?
`envir`	an environment, from which the `...` in `call` are retrieved, if any.

Value

An object of class `call`

Examples

```
f <- function(n) {
  g(n)
}

g <- function(n) {
  h(n)
}

h <- function(n) {
  match.call2(n)
```

```
}

f(0)
f(1)
f(2)
```

`null_switch` *Evaluate Expressions until not NULL*

Description

Evaluates expressions until one that is not `NULL` is encountered and returns that. Expressions after the first non-`NULL` result are not evaluated. If all expressions are `NULL`, it will return `NULL`

Usage

```
null_switch(...)
```

Arguments

`...`	expressions to try to evaluate

Value

The result of evaluating one of the expressions. Will only be `NULL` if they *all* evaluated to `NULL`

Examples

```
f <- function() {
  cat("Evaluating f\n")
  NULL
}
g <- function() {
  cat("Evaluating g\n")
  2
}

null_switch(NULL, f(), g())
null_switch(NULL, g(), f())
null_switch(f(), f(), f())
```

r_bern *Generate Bernoulli Distributed Values*

Description

Generates a set of Bernoulli distributed values.

Usage

```
r_bern(prob = 0.5, ..., n = default_n(prob), .seed = NULL)
```

Arguments

prob	vector of probability of successes, between 0 & 1
...	Unused
n	number of observations to generate. The default_n() function will provide a default value within context
.seed	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use pull_seed()

Value

A numeric vector of length n

Examples

```
set_n(5)

r_bern(0.9)

r_bern(seq(0, 1, 0.1))

r_bern(1 / 4, n = 10)
```

`r_beta` *Generate Beta Distributed Values*

Description

Generates a set of Beta distributed values.

Usage

```
r_beta(alpha, beta, ..., n = default_n(alpha, beta), .seed = NULL)
```

Arguments

`alpha, beta`	vectors of shape parameters, strictly positive
`...`	Unused
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A numeric vector of length `n`

Examples

```
set_n(5)

r_beta(1, 1)

r_beta(1:10, 2)

r_beta(1, 2, n = 10)
```

r_binom *Generate Binomial Distributed Values*

Description

Generates a set of Binomial distributed values.

Usage

```
r_binom(size, prob = 0.5, ..., n = default_n(size, prob), .seed = NULL)
```

Arguments

size	vector of number of trials, positive integer
prob	vector of probabilities of success on each trial, between 0 & 1
...	Unused
n	number of observations to generate. The default_n() function will provide a default value within context
.seed	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use pull_seed()

Value

A numeric vector of length n

Examples

```
set_n(5)

r_binom(10)

r_binom(1:10)

r_binom(10, 0.2)

r_binom(1, 0.2, n = 10)
```

r_cauchy *Generate Cauchy Distributed Values*

Description

Generates a set of Cauchy distributed values.

Usage

```
r_cauchy(
  location = 0,
  scale = 1,
  ...,
  n = default_n(location, scale),
  .seed = NULL
)
```

Arguments

location	vector of locations
scale	vector of scales, strictly positive
...	Unused
n	number of observations to generate. The default_n() function will provide a default value within context
.seed	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use pull_seed()

Value

A numeric vector of length n

Examples

```
set_n(5)

r_cauchy(10)

r_cauchy(1:10)
```

```
r_cauchy(10, 2)

r_cauchy(10, 2, n = 10)
```

`r_cdf` *Generate Random Numbers Based on an arbitrary CDF*

Description

Generates Random Numbers based on a distribution defined by any arbitrary cumulative distribution function

Usage

```
r_cdf(
  Fun,
  min = -Inf,
  max = Inf,
  ...,
  data = NULL,
  n = default_n(..., data),
  .seed = NULL
)
```

Arguments

`Fun`	function to use as the cdf. See details
`min, max`	range values for the domain of the `Fun`
`...`	arguments that can be passed to `Fun`
`data`	data set containing arguments to be passed to `Fun`
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use `pull_seed()`

Details

The `Fun` argument accepts `purrr` style inputs. Must be vectorised, defined on the whole Real line and return a single numeric value between 0 and 1 for any input. The random variable will be passed to `Fun` as the first argument. This means that R's argument matching can be used with named arguments in `...` if a different positional argument is wanted.

Value

A numeric vector of length n

Examples

```
set_n(5)

my_fun <- function(x, beta = 1) {
  1 - exp(-beta * x)
}

r_cdf(my_fun)


r_cdf(~ 1 - exp(-.x), min = 0)

r_cdf(~ 1 - exp(-.x * beta), beta = 1:10, min = 0)
```

---

r_chisq *Generate Chi-Squared Distributed Values*

Description

Generates a set of Chi-Squared distributed values.

Usage

```
r_chisq(df, ..., n = default_n(df), .seed = NULL)
```

Arguments

df	degrees of freedom, strictly positive
...	Unused
n	number of observations to generate. The default_n() function will provide a default value within context
.seed	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use pull_seed()

Value

A numeric vector of length n

Examples

```
set_n(5)

r_chisq(10)

r_chisq(1:10)

r_chisq(10, n = 10)
```

---

r_exp *Generate Exponentially Distributed Values*

Description

Generates a set of Exponentially distributed values.

Usage

```
r_exp(rate = 1, ..., n = default_n(rate), .seed = NULL)
```

Arguments

rate	vector of rates, strictly positive
...	Unused
n	number of observations to generate. The default_n() function will provide a default value within context
.seed	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use pull_seed()

Value

A numeric vector of length n

Examples

```
set_n(5)

r_exp(10)

r_exp(1:10)

r_exp(10, n = 10)
```

`r_fdist` *Generate F Distributed Values*

Description

Generates a set of F distributed values.

Usage

```
r_fdist(df1, df2, ..., n = default_n(df1, df2), .seed = NULL)
```

Arguments

`df1, df2`	vectors of degrees of freedom, strictly positive
`...`	Unused
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A numeric vector of length `n`

Examples

```
set_n(5)

r_fdist(1, 1)

r_fdist(1:10, 2)

r_fdist(10, 2)

r_fdist(10, 2, n = 10)
```

---

r_gamma *Generate Gamma Distributed Values*

Description

Generates a set of Gamma distributed values. Can be defined by one and only one of `scale`, `rate` or `mean`. This *must* be named in the call.

Usage

```
r_gamma(
  shape,
  ...,
  scale = 1,
  rate = NULL,
  mean = NULL,
  n = default_n(shape, scale, rate, mean),
  .seed = NULL
)
```

Arguments

`shape`	vector of shape parameters, strictly positive
`...`	Unused
`scale`	vector of scale parameters, cannot be specified with `rate` and `mean`, strictly positive
`rate`	vector of rate parameters, cannot be specified with `scale` and `mean`, strictly positive
`mean`	vector of mean parameters, cannot be specified with `scale` and `rate`, strictly positive
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following:

- NULL (default) will not change the current seed. This is the usual case for generating random numbers.
- A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results.
- TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results.

To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A numeric vector of length `n`

Examples

```
set_n(5)

r_gamma(10)

r_gamma(1:10, scale = 2)
r_gamma(1:10, rate = 1 / 2)
r_gamma(1:10, mean = 5)

r_gamma(10, n = 10)
```

---

r_geom *Generate Geometric Distributed Values*

Description

Generates a set of Geometric distributed values.

Usage

```
r_geom(prob = 0.5, ..., n = default_n(prob), .seed = NULL)
```

Arguments

`prob`	vector of probability of success, must strictly greater than 0 and (non-strictly) less than 1, i.e. 0 < prob <= 1
`...`	Unused
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers.

- A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results.
- TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results.

To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A numeric vector of length `n`

Examples

```
set_n(5)

r_geom(0.1)

r_geom(seq(0.1, 1, 0.1))

r_geom(0.1, n = 10)
```

---

r_hyper *Generate Hypergeometric Distributed Values*

Description

Generates a set of Hypergeometric distributed values.

Usage

```
r_hyper(
  total,
  positives,
  num,
  ...,
  n = default_n(total, positives, num),
  .seed = NULL
)
```

Arguments

`total`	size of the population (e.g. number of balls)
`positives`	number of elements with the desirable feature (e.g number of black balls)
`num`	number of draws to make
`...`	Unused
`n`	number of observations to generate. The `default_n()` function will provide a default value within context

`.seed` One of the following:

- NULL (default) will not change the current seed. This is the usual case for generating random numbers.
- A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results.
- TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results.

To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A numeric vector of length `n`

Examples

```
set_n(5)

r_hyper(10, 5, 5)

r_hyper(10:20, 10, 5)

r_hyper(10, 5, 5, n = 10)
```

---

r_letters *Generate Random Letters*

Description

Generates a set of Random Letters.

Usage

```
r_letters(nchar = 1, ..., n = default_n(nchar), .seed = NULL)

r_LETTERS(nchar = 1, ..., n = default_n(nchar), .seed = NULL)

r_Letters(nchar = 1, ..., n = default_n(nchar), .seed = NULL)
```

Arguments

`nchar`	vector of number of characters to return, positive integer
`...`	Unused
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following:

- NULL (default) will not change the current seed. This is the usual case for generating random numbers.
- A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results.
- TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results.

To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A character vector of length `n`

Functions

- `r_letters`: Uses only lower-case letters
- `r_LETTERS`: Uses only upper-case letters
- `r_Letters`: Uses lower- & upper-case letters

Examples

```
set_n(5)

r_letters(3)

r_letters(1:10)

r_letters(3, n = 10)

r_LETTERS(3)

r_LETTERS(1:10)

r_LETTERS(3, n = 10)

r_Letters(3)

r_Letters(1:10)

r_Letters(3, n = 10)
```

---

r_lgl *Generate Logical Values*

Description

Generates a set of Logical values.

Usage

```
r_lgl(prob = 0.5, ..., n = default_n(prob), .seed = NULL)
```

Arguments

prob	vector of probability of TRUE results, between 0 & 1
...	Unused
n	number of observations to generate. The default_n() function will provide a default value within context
.seed	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use pull_seed()

Value

A logical vector of length n

Examples

```
set_n(5)

r_lgl(0.9)

r_lgl(seq(0, 1, 0.1))

r_lgl(1 / 4, n = 10)
```

---

r_lnorm *Generate Log Normal Distributed Values*

Description

Generates a set of Log Normal distributed values.

Usage

```
r_lnorm(
  mean_log = 0,
  sd_log = 1,
  ...,
  n = default_n(mean_log, sd_log),
  .seed = NULL
)
```

Arguments

mean_log	vector of means (on the log scale)
sd_log	vector of standard deviations (on the log scale), strictly positive
...	Unused
n	number of observations to generate. The default_n() function will provide a default value within context
.seed	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use pull_seed()

Value

A numeric vector of length n

Examples

```
set_n(5)

r_lnorm(10)

r_lnorm(10, 2)

r_lnorm(1:10)

r_lnorm(-2, n = 10)
```

`r_matrix` *Generate a random Matrix*

Description

Generate a random matrix, given a rando function and it's dimensions. By default, this will generate a square matrix.

Usage

```
r_matrix(
  engine,
  row_names = NULL,
  col_names = NULL,
  ...,
  nrow = default_n(row_names),
  ncol = default_n(col_names),
  .seed = NULL
)
```

Arguments

`engine`	The rando function that will be used to generate the random numbers
`col_names, row_names`	names to be assigned to the rows or columns. This is also used in deciding the dimensions of the result.
`...`	Unused
`nrow, ncol`	dimensions of the matrix. The `default_n()` function will provide a default value within context.
`.seed`	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A matrix with `nrow` rows and `ncol` columns an a type as decided by the function passed to `engine`.

Examples

```
set_n(5)

r_matrix(r_norm)


r_matrix(r_unif,min=1,max=2)


r_matrix(r_norm,mean=10,sd=2,ncol=2)
```

`r_nbinom` *Generate Negative Binomial Distributed Values*

Description

Generates a set of Negative Binomial distributed values. Only two of `r`, `prob` and `mu` can be provided.

Usage

```
r_nbinom(
  r = NULL,
  prob = 0.5,
  ...,
  mu = NULL,
  n = default_n(r, prob, mu),
  .seed = NULL
)
```

Arguments

`r`	number of failure trials until stopping, strictly positive
`prob`	vector of probabilities of success on each trial, between 0 & 1
`...`	Unused
`mu`	vector of means
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results.

- TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results.

To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A numeric vector of length `n`

Note

It is important to note that this is the number of *failures*, and not the number of *successes*, as in `rnbinom()`, so `rnbinom(prob = x, ...)` is equivalent to `r_nbinom(prob=1-x, ...)`

Examples

```
set_n(5)

r_nbinom(10, 0.5)

r_nbinom(1:10, mu = 2)
#'
r_nbinom(10, 0.2, n = 10)
```

`r_norm` *Generate Normally Distributed Values*

Description

Generates a set of Normally distributed values.

Usage

```
r_norm(mean = 0, sd = 1, ..., n = default_n(mean, sd), .seed = NULL)
```

Arguments

`mean`	vector of means
`sd`	vector of standard deviations, strictly positive
`...`	Unused
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results.

- TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results.

To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A numeric vector of length `n`

Examples

```
set_n(5)

r_norm(10)

r_norm(10, 2)

r_norm(1:10)

r_norm(-2, n = 10)
```

---

r_pois *Generate Poisson Distributed Values*

Description

Generates a set of Poisson distributed values.

Usage

```
r_pois(rate, ..., n = default_n(rate), .seed = NULL)
```

Arguments

`rate`	vector of rates, strictly positive
`...`	Unused
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A numeric vector of length n

Examples

```
set_n(5)

r_pois(10)

r_pois(1:10)

r_pois(10, n = 10)
```

`r_sample` *Generate Random Sample*

Description

Generates a Sample from a set, with replacement

Usage

```
r_sample(sample, weights = NULL, ..., n = default_n(), .seed = NULL)
```

Arguments

`sample`	a set of values to choose from
`weights`	a vector of weights, must be the same length as `sample`, between 0 & 1
`...`	Unused
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A vector of length n of the same type as `sample`

Examples

```
set_n(15)

r_sample(c("blue", "red", "yellow"))

r_sample(c("blue", "red", "yellow"),
  weights = c(1, 5, 1)
)

r_sample(c("blue", "red", "yellow"), n = 10)
```

`r_tdist` *Generate T Distributed Values*

Description

Generates a set of Student's T distributed values.

Usage

```
r_tdist(df, ..., n = default_n(df), .seed = NULL)
```

Arguments

`df`	vector of degrees of freedom
`...`	Unused
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A numeric vector of length `n`

Examples

```
set_n(5)

r_tdist(10)

r_tdist(1:10)

r_tdist(10, n = 10)
```

`r_unif` *Generate Uniformly Distributed Values*

Description

Generates a set of Uniformly distributed values.

Usage

```
r_unif(min = 0, max = 1, ..., n = default_n(min, max), .seed = NULL)
```

Arguments

`min, max`	vectors of lower and upper limits of the distribution
`...`	Unused
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results. • TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results. To extract the random seed from a previously generated set of values, use `pull_seed()`

Value

A numeric vector of length `n`

Examples

```
set_n(5)

r_unif()

r_unif(1:5, 6:10)

r_unif(1:5, 10)

r_unif(n = 10)
```

`r_weibull` *Generate Weibull Distributed Values*

Description

Generates a set of Weibull distributed values.

Usage

```
r_weibull(
  shape,
  scale = 1,
  ...,
  b_scale = NULL,
  B_scale = NULL,
  n = default_n(shape, scale, b_scale, B_scale),
  .seed = NULL
)
```

Arguments

`shape`	vector of shape parameters, strictly positive
`scale`	vector of scale parameters, strictly positive
`...`	Unused
`b_scale, B_scale`	alternative definition of scale parameter, cannot be provided with `scale`, strictly positive.
`n`	number of observations to generate. The `default_n()` function will provide a default value within context
`.seed`	One of the following: • NULL (default) will not change the current seed. This is the usual case for generating random numbers. • A numeric value. This will be used to set the seed before generating the random numbers. This seed will be stored with the results.

- TRUE. A random seed value will be generated and set as the seed before the results are generated. Again, this will be stored with the results.

To extract the random seed from a previously generated set of values, use `pull_seed()`

Details

This function provides alternative definitions for the `scale` parameter depending on the user's parametrisation of the Weibull distribution, with k = `shape`.

Using λ = `scale`:

$$F(x) = 1 - exp(-(x/\lambda)^k)$$

Using b = `b_scale`:

$$F(x) = 1 - exp(-bx^k)$$

Using β = `B_scale`:

$$F(x) = 1 - exp(-(\beta x)^k)$$

Value

A numeric vector of length `n`

Examples

```
set_n(5)

r_weibull(10)

r_weibull(1:10, 2)

r_weibull(1:10, scale = 2)
r_weibull(1:10, b_scale = 2)
r_weibull(1:10, B_scale = 2)

r_weibull(10, 2, n = 10)
```

---

seed *Random Seed Defining Functions*

Description

Functions related to generating random seeds and utilising them for reproducibility.

Usage

```
gen_seed()

set_seed(seed)

fix_seed(reset = FALSE)

with_seed(seed, expression)

pull_seed(x)
```

Arguments

seed	The random seed to be used
reset	Should the fixed seed be forced to reset
expression	expression to be evaluated
x	object to extract the seed from

Details

Random values are generated based on the current seed used by the R system. This means by deliberately setting a seed in R, we can make work reproducible.

Value

`gen_seed()` returns a single numeric value

`with_seed()` returns the value of the evaluated expression after with the relevant seed as an attribute (if required)

`pull_seed()` returns a single numeric value

`fix_seed()` and `set_seed()` do not return anything

Functions

- `gen_seed`: Generates a random seed, which can be used in `set_seed()`
- `set_seed`: Sets the current seed
- `fix_seed`: Resets the seed to re-run code
- `with_seed`: Evaluates the expression after setting the seed. If `seed` is `TRUE`, then it first generates a seed using `gen_seed()`. Results are output with the `seed` attached (if set).#'
- `pull_seed`: Extracts the seed used to generate the results of `with_seed()`

Examples

```
my_seed <- gen_seed()


set_seed(my_seed)
```

```
r_norm(n=10)
set_seed(my_seed)
r_norm(n=10)


fix_seed()
r_norm(n=3)

fix_seed()
r_norm(n=3)

fix_seed(reset=TRUE)
r_norm(n=3)


res <- with_seed(my_seed, r_norm(n = 10))
res


pull_seed(res)
```

`set_n` *Set and Get the Default Value for n*

Description

Set and get the global value for n for rando functions

Usage

```
set_n(n)

get_n()
```

Arguments

`n`	value to set as the default n

Value

The current *global* default value for n.
`set_n()` returns this value invisibly

Examples

```
set_n(100)

get_n()
```

Index