

# Package: neodistr (via r-universe)

June 27, 2024

**Type** Package

**Title** Neo-Normal Distribution

**Version** 0.1.1

**Maintainer** Achmad Syahrul Choir <madsyair@stis.ac.id>

**Description** Provides functions for calculating the density, cumulative distribution, quantile, and random number of neo-normal distribution. It also interfaces with the 'brms' package, allowing the use of the neo-normal distribution as a custom family. This integration enables the application of various 'brms' formulas for neo-normal regression. The package implements the following distributions: Modified to be Stable as Normal from Burr (MSNBurr), Modified to be Stable as Normal from Burr-IIa (MSNBurr-IIa), Generalized of MSNBurr (GMSNBurr), and Jones-Faddy Skew-t. References: Choir, A. S. (2020). Unpublished Dissertation. Iriawan, N. (2000). Unpublished Dissertation. Jones, M. C. and Faddy, M. J. (2003). <doi:10.1111/1467-9868.00378>. Rigby, R. A., Stasinopoulos, M. D., Heller, G. Z., & Bastiani, F. D. (2019) <doi:10.1201/9780429298547>.

**License** GPL-3

**Depends** R (>= 3.6.0), shinythemes, plotly, brms

**Imports** rstan, stats, Rmpfr, ggplot2, shiny

**Suggests** kableExtra, knitr, testthat

**URL** <https://github.com/madsyair/neodistr>

**BugReports** <https://github.com/madsyair/neodistr/issues>

**Encoding** UTF-8

**NeedsCompilation** no

**RoxygenNote** 7.3.1

**Author** Achmad Syahrul Choir [aut, cre]

(<<https://orcid.org/0000-0001-7088-0646>>), Anisa' Faoziah

[aut], Nur Iriawan [aut]

(<<https://orcid.org/0000-0003-2833-6115>>)

**Repository** CRAN

**Date/Publication** 2024-06-26 12:40:02 UTC

## Contents

bnrm . . . . .	2
brms_custom_family . . . . .	7
gmsnburr . . . . .	8
jfst . . . . .	10
msnburr . . . . .	11
msnburr2a . . . . .	13
neoshiny . . . . .	14
stanf_gmsnburr . . . . .	15
stanf_jfst . . . . .	18
stanf_msnburr . . . . .	21
stanf_msnburr2a . . . . .	24
summary_dist . . . . .	26

**Index** 28

---

bnrm *Neo-normal model using brms*

---

## Description

Neo-normal model using brms

## Usage

```
bnrm(
  formula,
  data,
  family = msnburr(),
  prior = NULL,
  data2 = NULL,
  sample_prior = "no",
  knots = NULL,
  drop_unused_levels = TRUE,
  stanvars = NULL,
  fit = NA,
  save_pars = getOption("brms.save_pars", NULL),
  init = NULL,
  chains = 4,
  iter = 2000,
  warmup = floor(iter/2),
  thin = 1,
  cores = getOption("mc.cores", 1),
```

```

threads = getOption("brms.threads", NULL),
opencl = getOption("brms.opencl", NULL),
normalize = getOption("brms.normalize", TRUE),
control = list(adapt_delta = 0.9),
algorithm = getOption("brms.algorithm", "sampling"),
backend = getOption("brms.backend", "rstan"),
future = getOption("future", FALSE),
silent = 1,
seed = NA,
save_model = NULL,
stan_model_args = list(),
file = NULL,
file_compress = TRUE,
file_refit = getOption("brms.file_refit", "never"),
empty = FALSE,
rename = TRUE,
...
)

```

## Arguments

formula	An object of class <a href="#">formula</a> , <a href="#">brmsformula</a> , or <a href="#">mvbrmsformula</a> (or one that can be coerced to that classes): A symbolic description of the model to be fitted. The details of model specification are explained in <a href="#">brmsformula</a> .
data	An object of class <code>data.frame</code> (or one that can be coerced to that class) containing data of all variables used in the model.
family	the neo-normal distribution as response in regression: <a href="#">msnburr()</a> , <a href="#">msnburr2a()</a> , <a href="#">gmsnburr()</a> , <a href="#">jfst()</a> default argument in family is <code>vectorize=TRUE</code> . if not <code>vectorize</code> , give argument <code>vectorize=FALSE</code> , example: <a href="#">msnburr(vectorize=FALSE)</a>
prior	One or more <code>brmsprior</code> objects created by <a href="#">set_prior</a> or related functions and combined using the <code>c</code> method or the <code>+</code> operator. See also <a href="#">default_prior</a> for more help.
data2	A named list of objects containing data, which cannot be passed via argument <code>data</code> . Required for some objects used in autocorrelation structures to specify dependency structures as well as for within-group covariance matrices.
sample_prior	Indicate if draws from priors should be drawn additionally to the posterior draws. Options are "no" (the default), "yes", and "only". Among others, these draws can be used to calculate Bayes factors for point hypotheses via <a href="#">hypothesis</a> . Please note that improper priors are not sampled, including the default improper priors used by <code>brm</code> . See <a href="#">set_prior</a> on how to set (proper) priors. Please also note that prior draws for the overall intercept are not obtained by default for technical reasons. See <a href="#">brmsformula</a> how to obtain prior draws for the intercept. If <code>sample_prior</code> is set to "only", draws are drawn solely from the priors ignoring the likelihood, which allows among others to generate draws from the prior predictive distribution. In this case, all parameters must have proper priors.
knots	Optional list containing user specified knot values to be used for basis construction of smoothing terms. See <a href="#">gamm</a> for more details.

<code>drop_unused_levels</code>	Should unused factors levels in the data be dropped? Defaults to TRUE.
<code>stanvars</code>	An optional <code>stanvars</code> object generated by function <code>stanvar</code> to define additional variables for use in <b>Stan</b> 's program blocks.
<code>fit</code>	An instance of S3 class <code>brmsfit</code> derived from a previous fit; defaults to NA. If <code>fit</code> is of class <code>brmsfit</code> , the compiled model associated with the fitted result is re-used and all arguments modifying the model code or data are ignored. It is not recommended to use this argument directly, but to call the <code>update</code> method, instead.
<code>save_pars</code>	An object generated by <code>save_pars</code> controlling which parameters should be saved in the model. The argument has no impact on the model fitting itself.
<code>init</code>	Initial values for the sampler. If NULL (the default) or "random", Stan will randomly generate initial values for parameters in a reasonable range. If 0, all parameters are initialized to zero on the unconstrained space. This option is sometimes useful for certain families, as it happens that default random initial values cause draws to be essentially constant. Generally, setting <code>init = 0</code> is worth a try, if chains do not initialize or behave well. Alternatively, <code>init</code> can be a list of lists containing the initial values, or a function (or function name) generating initial values. The latter options are mainly implemented for internal testing but are available to users if necessary. If specifying initial values using a list or a function then currently the parameter names must correspond to the names used in the generated Stan code (not the names used in R). For more details on specifying initial values you can consult the documentation of the selected backend.
<code>chains</code>	Number of Markov chains (defaults to 4).
<code>iter</code>	Number of total iterations per chain (including warmup; defaults to 2000).
<code>warmup</code>	A positive integer specifying number of warmup (aka burnin) iterations. This also specifies the number of iterations used for stepsize adaptation, so warmup draws should not be used for inference. The number of warmup should not be larger than <code>iter</code> and the default is <code>iter/2</code> .
<code>thin</code>	Thinning rate. Must be a positive integer. Set <code>thin &gt; 1</code> to save memory and computation time if <code>iter</code> is large.
<code>cores</code>	Number of cores to use when executing the chains in parallel, which defaults to 1 but we recommend setting the <code>mc.cores</code> option to be as many processors as the hardware and RAM allow (up to the number of chains). For non-Windows OS in non-interactive R sessions, forking is used instead of PSOCK clusters.
<code>threads</code>	Number of threads to use in within-chain parallelization. For more control over the threading process, <code>threads</code> may also be a <code>brmsthreads</code> object created by <code>threading</code> . Within-chain parallelization is experimental! We recommend its use only if you are experienced with Stan's <code>reduce_sum</code> function and have a slow running model that cannot be sped up by any other means. Can be set globally for the current R session via the "brms.threads" option.
<code>opencl</code>	The platform and device IDs of the OpenCL device to use for fitting using GPU support. If you don't know the IDs of your OpenCL device, <code>c(0, 0)</code> is most likely what you need. For more details, see <code>opencl</code> . Can be set globally for the current R session via the "brms.opencl" option

normalize	Logical. Indicates whether normalization constants should be included in the Stan code (defaults to TRUE). Setting it to FALSE requires Stan version $\geq 2.25$ to work. If FALSE, sampling efficiency may be increased but some post processing functions such as <code>bridge_sampler</code> will not be available. Can be controlled globally for the current R session via the 'brms.normalize' option.
control	A named list of parameters to control the sampler's behavior. It defaults to NULL so all the default values are used. The most important control parameters are discussed in the 'Details' section below. For a comprehensive overview see <a href="#">stan</a> .
algorithm	Character string naming the estimation approach to use. Options are "sampling" for MCMC (the default), "meanfield" for variational inference with independent normal distributions, "fullrank" for variational inference with a multivariate normal distribution, or "fixed_param" for sampling from fixed parameter values. Can be set globally for the current R session via the "brms.algorithm" option.
backend	Character string naming the package to use as the backend for fitting the Stan model. Options are "rstan" (the default) or "cmdstanr". Can be set globally for the current R session via the "brms.backend" option. Details on the <b>rstan</b> and <b>cmdstanr</b> packages are available at <a href="https://mc-stan.org/rstan/">https://mc-stan.org/rstan/</a> and <a href="https://mc-stan.org/cmdstanr/">https://mc-stan.org/cmdstanr/</a> , respectively. Additionally a "mock" backend is available to make testing <b>brms</b> and packages that depend on it easier. The "mock" backend does not actually do any fitting, it only checks the generated Stan code for correctness and then returns whatever is passed in an additional <code>mock_fit</code> argument as the result of the fit.
future	Logical; If TRUE, the <b>future</b> package is used for parallel execution of the chains and argument cores will be ignored. Can be set globally for the current R session via the "future" option. The execution type is controlled via <a href="#">plan</a> (see the examples section below).
silent	Verbosity level between 0 and 2. If 1 (the default), most of the informational messages of compiler and sampler are suppressed. If 2, even more messages are suppressed. The actual sampling progress is still printed. Set <code>refresh = 0</code> to turn this off as well. If using <code>backend = "rstan"</code> you can also set <code>open_progress = FALSE</code> to prevent opening additional progress bars.
seed	The seed for random number generation to make results reproducible. If NA (the default), <b>Stan</b> will set the seed randomly.
save_model	Either NULL or a character string. In the latter case, the model's Stan code is saved via <a href="#">cat</a> in a text file named after the string supplied in <code>save_model</code> .
stan_model_args	A list of further arguments passed to <code>rstan::stan_model</code> for <code>backend = "rstan"</code> or to <code>cmdstanr::cmdstan_model</code> for <code>backend = "cmdstanr"</code> , which allows to change how models are compiled.
file	Either NULL or a character string. In the latter case, the fitted model object is saved via <a href="#">saveRDS</a> in a file named after the string supplied in <code>file</code> . The <code>.rds</code> extension is added automatically. If the file already exists, <b>brm</b> will load and return the saved model object instead of refitting the model. Unless you specify the <code>file_refit</code> argument as well, the existing files won't be overwritten, you

	have to manually remove the file in order to refit and save the model under an existing file name. The file name is stored in the <code>brmsfit</code> object for later usage.
<code>file_compress</code>	Logical or a character string, specifying one of the compression algorithms supported by <code>saveRDS</code> . If the <code>file</code> argument is provided, this compression will be used when saving the fitted model object.
<code>file_refit</code>	Modifies when the fit stored via the <code>file</code> argument is re-used. Can be set globally for the current R session via the <code>"brms.file_refit"</code> option. For "never" (default) the fit is always loaded if it exists and fitting is skipped. For "always" the model is always refitted. If set to "on_change", <code>brms</code> will refit the model if model, data or algorithm as passed to Stan differ from what is stored in the file. This also covers changes in priors, <code>sample_prior</code> , <code>stanvars</code> , covariance structure, etc. If you believe there was a false positive, you can use <code>brmsfit_needs_refit</code> to see why refit is deemed necessary. Refit will not be triggered for changes in additional parameters of the fit (e.g., initial values, number of iterations, control arguments, ...). A known limitation is that a refit will be triggered if within-chain parallelization is switched on/off.
<code>empty</code>	Logical. If TRUE, the Stan model is not created and compiled and the corresponding 'fit' slot of the <code>brmsfit</code> object will be empty. This is useful if you have estimated a <code>brms</code> -created Stan model outside of <code>brms</code> and want to feed it back into the package.
<code>rename</code>	For internal use only.
<code>...</code>	Further arguments passed to Stan. For <code>backend = "rstan"</code> the arguments are passed to <code>sampling</code> or <code>vb</code> . For <code>backend = "cmdstanr"</code> the arguments are passed to the <code>cmdstanr::sample</code> or <code>cmdstanr::variational</code> method.

## Details

Fit a neo-normal model that using `brm` function in `brms` package. All arguments in this functions follow arguments of `brm` function, except `family`

## Value

An object of class `brmsfit`, which contains the posterior draws along with many other useful information about the model. Use `methods(class = "brmsfit")` for an overview on available methods.

## Author(s)

Achmad Syahrul Choir

## References

- Buerkner, P-C (2017). `brms`: An R Package for Bayesian Multilevel Models Using Stan. *Journal of Statistical Software*, 80(1), 1-28. doi:10.18637/jss.v080.i01
- Choir, A. S. (2020). The New Neo-Normal Distributions and their Properties. Dissertation. Institut Teknologi Sepuluh Nopember.
- Iriawan, N. (2000). Computationally Intensive Approaches to Inference in Neo-Normal Linear Models. Curtin University of Technology.

## Examples

```
library(neodistr)
x<-runif(100)
e<-rmsnburr(100,0,1,0.8)
y<-0.5+0.8*x+e
data<-data.frame(y,x)
fit <- bnm(
  y ~ x, data = data,
  family = msnburr())
summary(fit)
pp <- posterior_predict(fit)
ppe <- posterior_epred(fit)
loo(fit)
```

---

brms\_custom\_family      *Neonormal as custom distribution family in brms*

---

## Description

Neonormal as custom distribution family in brms

## Usage

```
brms_custom_family(family = "msnburr", vectorize = TRUE)
```

## Arguments

family	distribution neo-normal option: "msnburr", "msnburr2a", "gmsnburr", and "jfst"
vectorize	logical; if TRUE, Stan code of family distribution is vectorize The default value of this parameter is TRUE

## Value

custom\_family is an object of class custom family of brms and stanvars\_family is stanvars object (the Stan code of function of neo-normal distributions (lpdf,cdf,lcdf,lccdf,quantile and rng))

## Author(s)

Achmad Syahrul Choir

## Examples

```
library(brms)
library(neodistr)
x<-runif(100)
e<-rmsnburr(100,0,1,0.8)
y<-0.5+0.8*x+e
```

```

data<-data.frame(y,x)
msnburr<-brms_custom_family("msnburr")
fit <- brm(
  y ~ x, data = data,
  family = msnburr$custom_family, stanvars = msnburr$stanvars_family,
  prior=c(set_prior("cauchy(0,5)",class="alpha"),set_prior("cauchy(0,1)",class="sigma"))
)
summary(fit)
pp <- posterior_predict(fit)
ppe <- posterior_epred(fit)
loo(fit)

```

---

gmsnburr

*GMSNBurr distribution*


---

### Description

To calculate density function, distribution function, quantile function, and build data from random generator function for the GMSNBurr Distribution.

### Usage

```
dgmsnburr(x, mu = 0, sigma = 1, alpha = 1, beta = 1, log = FALSE)
```

```

pgmsnburr(
  q,
  mu = 0,
  sigma = 1,
  alpha = 1,
  beta = 1,
  lower.tail = TRUE,
  log.p = FALSE
)

```

```

qgmsnburr(
  p,
  mu = 0,
  sigma = 1,
  alpha = 1,
  beta = 1,
  lower.tail = TRUE,
  log.p = FALSE
)

```

```
rgmsnburr(n, mu = 0, sigma = 1, alpha = 1, beta = 1)
```



**Arguments**

x, q	vector of quantiles.
mu	a location parameter.
sigma	a scale parameter.
alpha	a shape parameter.
beta	a shape parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p) The default value of this parameter is FALSE.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ .
p	vectors of probabilities.
n	number of observations.

**Details****GMSNBurr Distribution**

The GMSNBurr distribution with parameters  $\mu, \sigma, \alpha$ , and  $\beta$  has density:

$$f(x|\mu, \sigma, \alpha, \beta) = \frac{\omega}{B(\alpha, \beta)\sigma} \left(\frac{\beta}{\alpha}\right)^\beta \exp\left(-\beta\omega\left(\frac{x-\mu}{\sigma}\right)\right) \left(1 + \frac{\beta}{\alpha}\exp\left(-\omega\left(\frac{x-\mu}{\sigma}\right)\right)\right)^{-(\alpha+\beta)}$$

where  $-\infty < x < \infty, -\infty < \mu < \infty, \sigma > 0, \alpha > 0, \beta > 0$  and  $\omega = \frac{B(\alpha, \beta)}{\sqrt{2\pi}} \left(1 + \frac{\beta}{\alpha}\right)^{\alpha+\beta} \left(\frac{\beta}{\alpha}\right)^{-\beta}$

**Value**

dgmsnburr gives the density , pgmasnburr gives the distribution function, qgmsnburr gives quantiles function, rgmsnburr generates random numbers.

**Author(s)**

Achmad Syahrul Choir

**References**

Choir, A. S. (2020). The New Neo-Normal Distributions and their Properties. Disertation. Institut Teknologi Sepuluh Nopember.

Iriawan, N. (2000). Computationally Intensive Approaches to Inference in Neo-Normal Linear Models. Curtin University of Technology.

**Examples**

```
library("neodistr")
dgmsnburr(0, mu=0, sigma=1, alpha=1, beta=1)
pgmsnburr(4, mu=0, sigma=1, alpha=1, beta=1)
qgmsnburr(0.4, mu=0, sigma=1, alpha=1, beta=1)
r=rgmsnburr(10000, mu=0, sigma=1, alpha=1, beta=1)
head(r)
hist(r, xlab = 'GMSNBurr random number', ylab = 'Frequency',
main = 'Distribution of GMSNBurr Random Number ')
```

jfst

*Jones Faddy's Skew-t Distribution***Description**

To calculate density function, distribution function, quantile function, and build data from random generator function for the Jones-Faddy's Skew-t Distribution.

**Usage**

```
djfst(x, mu = 0, sigma = 1, alpha = 2, beta = 2, log = FALSE)
```

```
pjfst(
  q,
  mu = 0,
  sigma = 1,
  alpha = 2,
  beta = 2,
  lower.tail = TRUE,
  log.p = FALSE
)
```

```
qjfst(
  p,
  mu = 0,
  sigma = 1,
  alpha = 2,
  beta = 2,
  lower.tail = TRUE,
  log.p = FALSE
)
```

```
rjfst(n, mu = 0, sigma = 1, alpha = 2, beta = 2)
```

**Arguments**

x, q	vector of quantiles.
mu	a location parameter.
sigma	a scale parameter.
alpha	a shape parameter (skewness).
beta	a shape parameter (kurtosis).
log, log.p	logical; if TRUE, probabilities p are given as log(p) The default value of this parameter is FALSE
lower.tail	logical;if TRUE (default), probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ .
p	vectors of probabilities.
n	number of observations.

**Details****Jones-Faddy's Skew-t Distribution**

The Jones-Faddy's Skew-t distribution with parameters  $\mu$ ,  $\sigma$ ,  $\alpha$ , and  $\beta$  has density:

$$f(x|\mu, \sigma, \beta, \alpha) = \frac{c}{\sigma} \left[ 1 + \frac{z}{\sqrt{\alpha + \beta + z^2}} \right]^{\alpha + \frac{1}{2}} \left[ 1 - \frac{z}{\sqrt{\alpha + \beta + z^2}} \right]^{\beta + \frac{1}{2}}$$

where  $-\infty < x < \infty$ ,  $-\infty < \mu < \infty$ ,  $\sigma > 0$ ,  $\alpha > 0$ ,  $\beta > 0$ ,  $z = \frac{x-\mu}{\sigma}$ ,  $c = \left[ 2^{(\alpha+\beta-1)} (\alpha + \beta)^{\frac{1}{2}} B(a, b) \right]^{-1}$ ,

**Value**

djfst gives the density , pjfst gives the distribution function, qjfst gives quantiles function, rjfst generates random numbers.

**Author(s)**

Anisa' Faoziah

**References**

Jones, M.C. and Faddy, M. J. (2003) A skew extension of the t distribution, with applications. Journal of the Royal Statistical Society, Series B, 65, pp 159-174

Rigby, R.A. and Stasinopoulos, M.D. and Heller, G.Z. and De Bastiani, F. (2019) Distributions for Modeling Location, Scale, and Shape: Using GAMLSS in R. CRC Press

**Examples**

```
djfst(4, mu=0, sigma=1, alpha=2, beta=2)
pjfst(4, mu=0, sigma=1, alpha=2, beta=2)
qjfst(0.4, mu=0, sigma=1, alpha=2, beta=2)
r=rjfst(10000, mu=0, sigma=1, alpha=2, beta=2)
head(r)
hist(r, xlab = 'jfst random number', ylab = 'Frequency',
main = 'Distribution of jfst Random Number ')
```

**Description**

To calculate density function, distribution function, quantile function, and build data from random generator function for the MSNBurr Distribution.

**Usage**

```
dmsnburr(x, mu = 0, sigma = 1, alpha = 1, log = FALSE)
```

```
pmsnburr(q, mu = 0, sigma = 1, alpha = 1, lower.tail = TRUE, log.p = FALSE)
```

```
qmsnburr(p, mu = 0, sigma = 1, alpha = 1, lower.tail = TRUE, log.p = FALSE)
```

```
rmsnburr(n, mu = 0, sigma = 1, alpha = 1)
```

**Arguments**

x, q	vector of quantiles.
mu	a location parameter.
sigma	a scale parameter.
alpha	a shape parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p) The default value of this parameter is FALSE.
lower.tail	logical;if TRUE (default), probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ .
p	vectors of probabilities.
n	number of observations.

**Details****MSNBurr Distribution**

The MSNBurr distribution with parameters  $\mu$ ,  $\sigma$ , and  $\alpha$  has density:

$$f(x|\mu, \sigma, \alpha) = \frac{\omega}{\sigma} \exp\left(\omega\left(\frac{x-\mu}{\sigma}\right)\right) \left(1 + \frac{\exp\left(\omega\left(\frac{x-\mu}{\sigma}\right)\right)}{\alpha}\right)^{-(\alpha+1)}$$

where  $-\infty < x < \infty$ ,  $-\infty < \mu < \infty$ ,  $\sigma > 0$ ,  $\alpha > 0$ ,  $\omega = \frac{1}{\sqrt{2\pi}}\left(1 + \frac{1}{\alpha}\right)^{\alpha+1}$

**Value**

dmsnburr gives the density, pmsnburr gives the distribution function, qmsnburr gives quantiles function, rmsnburr generates random numbers.

**Author(s)**

Achmad Syahrul Choir and Nur Iriawan

**References**

- Iriawan, N. (2000). Computationally Intensive Approaches to Inference in Neo-Normal Linear Models. Curtin University of Technology.
- Choir, A. S. (2020). The New Neo-Normal Distributions and their Properties. Disertation. Institut Teknologi Sepuluh Nopember.

**Examples**

```

library("neodistr")
dmsnburr(0, mu=0, sigma=1, alpha=0.1)
plot(function(x) dmsnburr(x, alpha=0.1), -20, 3,
main = "Left Skew MSNBurr Density ",ylab="density")
pmsnburr(7, mu=0, sigma=1, alpha=1)
qmsnburr(0.6, mu=0, sigma=1, alpha=1)
r<- rmsnburr(10000, mu=0, sigma=1, alpha=1)
head(r)
hist(r, xlab = 'MSNBurr random number', ylab = 'Frequency',
main = 'Distribution of MSNBurr Random Number ')

```

msnburr2a

*MSNBurr-IIa distribution.***Description**

To calculate density function, distribution function, quantile function, and build data from random generator function for the MSNBurr distribution.

**Usage**

```

dmsnburr2a(x, mu = 0, sigma = 1, alpha = 1, log = FALSE)

pmsnburr2a(q, mu = 0, sigma = 1, alpha = 1, lower.tail = TRUE, log.p = FALSE)

qmsnburr2a(p, mu = 0, sigma = 1, alpha = 1, lower.tail = TRUE, log.p = FALSE)

rmsnburr2a(n, mu = 0, sigma = 1, alpha = 1)

```

**Arguments**

x, q	vector of quantiles.
mu	a location parameter.
sigma	a scale parameter.
alpha	a shape parameter
log, log.p	logical; if TRUE, probabilities p are given as log(p), The default value of this parameter is FALSE.
lower.tail	logical;if TRUE (default), probabilities are $P[X \leq x]$ , otherwise, $P[X > x]$ .
p	vectors of probabilities.
n	number of observations.

**Details****MSNBurr-IIa Distribution**

The MSNBurr-IIa distribution with parameters  $\mu$ ,  $\sigma$ , and  $\alpha$  has density:

$$f(x|\mu, \sigma, \alpha) = \frac{\omega}{\sigma} \exp\left(\omega\left(\frac{x-\mu}{\sigma}\right)\right) \left(1 + \frac{\exp\left(\omega\left(\frac{x-\mu}{\sigma}\right)\right)}{\alpha}\right)^{-(\alpha+1)}$$

where  $-\infty < x < \infty$ ,  $-\infty < \mu < \infty$ ,  $\sigma > 0$ ,  $\alpha > 0$ ,  $\omega = \frac{1}{\sqrt{2\pi}}\left(1 + \frac{1}{\alpha}\right)^{\alpha+1}$

**Value**

dmsnburr2a gives the density, pmsnburr2a gives the distribution function, qmsnburr2a gives the quantile function and rmsnburr2a generates random numbers.

**Author(s)**

Achmad Syahrul Choir and Nur Iriawan

**References**

Choir, A. S. (2020). The New Neo-Normal Distributions and their Properties. Dissertation. Institut Teknologi Sepuluh Nopember.

**Examples**

```
library("neodistr")
dmsnburr2a(7, mu=0, sigma=1, alpha=0.1)
plot(function(x) dmsnburr2a(x, alpha=0.1), -3, 20,
main = "Right Skew MSNBurr-IIa Density ", ylab="density")
p=pmsnburr2a(4, mu=0, sigma=1, alpha=1)
p
q=qmsnburr2a(p, mu=0, sigma=1, alpha=1)
q
qmsnburr2a(0.5, mu=0, sigma=1, alpha=1)
r=rmsnburr2a(10000, mu=0, sigma=1, alpha=0.1)
head(r)
hist(r, xlab = 'MSNBurr random number', ylab = 'Frequency',
main = 'Distribution of MSNBurr-IIa Random Number ')
```

---

neoshiny

*Starts shiny application for the neodistr package*

---

**Description**

Starts shiny application for the neodistr package

**Usage**

```
neoshiny()
```

**Value**

Starts shiny application for the neodistr package.

**Author(s)**

Anisa' Faoziah and Achmad Syahrul Choir

**Examples**

```
if (interactive()) {
  suppressMessages(library(neodistr))
  neoshiny()
}
```

---

stanf_gmsnburr	<i>Stan function of GMSNBurr Distribution</i>
----------------	---

---

**Description**

Stan code of GMSNBurr distribution for custom distribution in stan

**Usage**

```
stanf_gmsnburr(vectorize = TRUE, rng = TRUE)
```

**Arguments**

vectorize	logical; if TRUE, Vectorize Stan code of GMSNBurr distribution are given The default value of this parameter is TRUE
rng	logical; if TRUE, Stan code of quantile and random number generation of GMSNBurr distribution are given The default value of this parameter is TRUE

**Details**

GMSNBurr Distribution has density:

$$f(y|\mu, \sigma, \alpha, \beta) = \frac{\omega}{B(\alpha, \beta)\sigma} \left(\frac{\beta}{\alpha}\right)^\beta \exp\left(-\beta\omega\left(\frac{y-\mu}{\sigma}\right)\right) \left(1 + \frac{\beta}{\alpha} \exp\left(-\omega\left(\frac{y-\mu}{\sigma}\right)\right)\right)^{-(\alpha+\beta)}$$

where  $-\infty < y < \infty$ ,  $-\infty < \mu < \infty$ ,  $\sigma > 0$ ,  $\alpha > 0$ ,  $\beta > 0$  and  $\omega = \frac{B(\alpha, \beta)}{\sqrt{2\pi}} \left(1 + \frac{\beta}{\alpha}\right)^{\alpha+\beta} \left(\frac{\beta}{\alpha}\right)^{-\beta}$

This function gives stan code of log density, cumulative distribution, log of cumulative distribution, log complementary cumulative distribution, quantile, random number of GMSNBurr distribution

**Value**

msnburr\_lpdf gives the stans's code of log of density, msnburr\_cdf gives the stans's code of distribution function, gmsnburr\_lcdf gives the stans's code of log of distribution function, gmsnburr\_lccdf gives the stans's code of complement of log ditribution function (1-gmsnburr\_lcdf), and gmsnburr\_rng the stans's code of generates random numbers.

**Author(s)**

Achmad Syahrul Choir

**References**

Choir, A. S. (2020). The New Neo-Normal Distributions and their Properties. Disertation. Institut Teknologi Sepuluh Nopember.

**Examples**

```
library(neodistr)
library(rstan)
#inputting data
set.seed(136)
dt <- rgmsnburr(100,0,1,0.5,0.5) # random generating MSNBurr-IIA data
dataf <- list(
  n = 100,
  y = dt
)
#### not vector
##Calling the function of the neo-normal distribution that is available in the package.
func_code<-paste(c("functions{",neodistr::stanf_gmsnburr(vectorize=FALSE),"}"),collapse="\n")
#define stan model code
model<-"
  data {
    int<lower=1> n;
    vector[n] y;
  }
  parameters {
    real mu;
    real <lower=0> sigma;
    real <lower=0> alpha;
    real <lower=0> beta;
  }
  model {
    for(i in 1:n){
      y[i]~gmsnburr(mu, sigma, alpha, beta);
    }
    mu~cauchy(0,1);
    sigma~cauchy(0,2.5);
    alpha~cauchy(0,1);
    beta~cauchy(0,1);
  }
"
```



```

#merge stan model code and selected neo-normal stan function
fit_code<-paste(c(func_code,model,"\n"),collapse="\n")

# Create the model using stan function
fit1 <- stan(
  model_code = fit_code, # Stan program
  data = dataf, # named list of data
  chains = 2, # number of Markov chains
  #warmup = 5000, # number of warmup iterations per chain
  iter = 10000, # total number of iterations per chain
  cores = 2, # number of cores (could use one per chain)
  control = list( #control samplers behavior
    adapt_delta=0.9
  )
)

# Showing the estimation results of the parameters that were executed using the Stan file
print(fit1, pars=c("mu", "sigma", "alpha", "beta","lp__"), probs=c(.025,.5,.975))

# Vector
##Calling the function of the neo-normal distribution that is available in the package.
func_code_vector<-paste(c("functions{",neodistr::stanf_gmsnburr(vectorize=TRUE),"}"),collapse="\n")
# define stan model as vector
model_vector<-"
data {
  int<lower=1> n;
  vector[n] y;
}
parameters {
  real mu;
  real <lower=0> sigma;
  real <lower=0> alpha;
  real <lower=0> beta;
}
model {
  y~gmsnburr(rep_vector(mu,n),sigma,alpha,beta);
  mu~cauchy(0,1);
  sigma~cauchy(0,2.5);
  alpha~cauchy(0,1);
  beta~cauchy(0,1);
}
"
#merge stan model code and selected neo-normal stan function
fit_code_vector<-paste(c(func_code_vector,model_vector,"\n"),collapse="\n")

# Create the model using stan function
fit2 <- stan(
  model_code = fit_code_vector, # Stan program
  data = dataf, # named list of data
  chains = 2, # number of Markov chains
  #warmup = 5000, # number of warmup iterations per chain
  iter = 10000, # total number of iterations per chain

```

```

cores = 2,          # number of cores (could use one per chain)
control = list(     #control samplers behavior
  adapt_delta=0.9
)
)

# Showing the estimation results of the parameters
print(fit2, pars=c("mu", "sigma", "alpha", "beta", "lp__"), probs=c(.025, .5, .975))

```

---

stanf\_jfst

*Stan function of Jones and Faddys Skew-t Distribution*


---

### Description

Stan code of JFST distribution for custom distribution in stan

### Usage

```
stanf_jfst(vectorize = TRUE, rng = TRUE)
```

### Arguments

vectorize	logical; if TRUE, Vectorize Stan code of Jones and faddy distribution are given The default value of this parameter is TRUE
rng	logical; if TRUE, Stan code of quantile and random number generation of Jones and faddy distribution are given The default value of this parameter is TRUE

### Details

Jones-Faddy's Skew-t distribution has density:

$$f(y|\mu, \sigma, \beta, \alpha) = \frac{c}{\sigma} \left[ 1 + \frac{z}{\sqrt{\alpha + \beta + z^2}} \right]^{\alpha + \frac{1}{2}} \left[ 1 - \frac{z}{\sqrt{\alpha + \beta + z^2}} \right]^{\beta + \frac{1}{2}}$$

where  $-\infty < y < \infty$ ,  $-\infty < \mu < \infty$ ,  $\sigma > 0$ ,  $\alpha > 0$ ,  $\beta > 0$ ,  $z = \frac{y-\mu}{\sigma}$ ,  $c = \left[ 2^{(\alpha+\beta-1)} (\alpha + \beta)^{\frac{1}{2}} B(a, b) \right]^{-1}$ ,

This function gives stan code of log density, cumulative distribution, log of cumulatif distribution, log complementary cumulative distribution, quantile, random number of Jones-Faddy's Skew-t distribution

### Value

jfst\_lpdf gives stan's code of the log of density, jfst\_cdf gives stan's code of the distribution function, jfst\_lcdf gives stan's code of the log of distribution function and jfst\_rng gives stan's code of generates random numbers.

**Author(s)**

Anisa' Faoziah and Achmad Syahrul Choir

**References**

Jones, M.C. and Faddy, M. J. (2003) A skew extension of the t distribution, with applications. Journal of the Royal Statistical Society, Series B, 65, pp 159-174

Rigby, R.A. and Stasinopoulos, M.D. and Heller, G.Z. and De Bastiani, F. (2019) Distributions for Modeling Location, Scale, and Shape: Using GAMLSS in R. CRC Press

**Examples**

```
library (neodistr)
library (rstan)

# inputting data
set.seed(400)
dt <- neodistr::rjfst(100,mu=0, sigma=1, alpha = 2, beta = 2) # random generating JFST data
dataf <- list(
  n = 100,
  y = dt
)

#### not vector
## Calling the function of the neo-normal distribution that is available in the package.
func_code<-paste(c("functions{",neodistr::stanf_jfst(vectorize=FALSE),"}"),collapse="\n")

# Define Stan Model Code
model <-"
  data{
    int<lower=1> n;
    vector[n] y;
  }
  parameters{
    real mu;
    real <lower=0> sigma;
    real <lower=0> alpha;
    real <lower=0> beta;
  }
  model {
    for(i in 1 : n){
      y[i] ~ jfst(mu,sigma, alpha, beta);
    }
    mu ~ cauchy(0,1);
    sigma ~ cauchy(0, 2.5);
    alpha ~ lognormal(0,5);
    beta ~ lognormal(0,5);
  }
"
```

```

# Merge stan model code and selected neo-normal stan function
fit_code <- paste (c(func_code,model,"\n"), collapse = "\n")

# Create the model using Stan Function
fit1 <- stan(
  model_code = fit_code, # Stan Program
  data = dataf,         # named list data
  chains = 2,          # number of markov chains
  warmup = 5000,       # total number of warmup iterarions per chain
  iter = 10000,        # total number of iterations iterarions per chain
  cores = 2,           # number of cores (could use one per chain)
  control = list(      # control sampel behavior
    adapt_delta = 0.99
  ),
  refresh = 1000       # progress has shown if refresh >=1, else no progress shown
)

# Showing the estimation result of the parameters that were executed using the Stan file
print(fit1, pars = c("mu", "sigma", "alpha", "beta", "lp__"), probs=c(.025,.5,.975))

#### Vector
## Calling the function of the neonormal distribution that is available in the package.
func_code_vector<-paste(c("functions{",neodistr::stanf_jfst(vectorize=TRUE),"}"),collapse="\n")

# Define Stan Model Code
model_vector <- "
  data{
    int<lower=1> n;
    vector[n] y;
  }
  parameters{
    real mu;
    real <lower=0> sigma;
    real <lower=0> alpha;
    real <lower=0>beta;
  }
  model {
    y ~ jfst(rep_vector(mu,n),sigma, alpha, beta);
    mu ~ cauchy (0,1);
    sigma ~ cauchy (0, 2.5);
    alpha ~ lognormal(0,5);
    beta ~ lognormal(0,5);

  }
"

# Merge stan model code and selected neo-normal stan function
fit_code_vector <- paste (c(func_code_vector,model_vector,"\n"), collapse = "\n")

# Create the model using Stan Function
fit2 <- stan(

```

```

    model_code = fit_code_vector, # Stan Program
    data = dataf,                # named list data
    chains = 2,                  # number of markov chains
    warmup = 5000,              # total number of warmup iterarions per chain
    iter = 10000,                # total number of iterations iterarions per chain
    cores = 2,                   # number of cores (could use one per chain)
    control = list(              # control sampel behavior
      adapt_delta = 0.99
    ),
    refresh = 1000                # progress has shown if refresh >=1, else no progress shown
  )

# Showing the estimation result of the parameters that were executed using the Stan file
print(fit2, pars = c("mu", "sigma", "alpha", "beta", "lp_"), probs=c(.025,.5,.975))

```

---

stanf\_msnburr

*Stan function of MSNBurr Distribution*


---

## Description

Stan code of MSNBurr distribution for custom distribution in stan

## Usage

```
stanf_msnburr(vectorize = TRUE, rng = TRUE)
```

## Arguments

vectorize	logical; if TRUE, Vectorize Stan code of MSNBurr distribution are given The default value of this parameter is TRUE
rng	logical; if TRUE, Stan code of quantile and random number generation of MSNBurr distribution are given The default value of this parameter is TRUE

## Details

MSNBurr Distribution has density:

$$f(y|\mu, \sigma, \alpha) = \frac{\omega}{\sigma} \exp\left(-\omega\left(\frac{y-\mu}{\sigma}\right)\right) \left(1 + \frac{\exp\left(-\omega\left(\frac{y-\mu}{\sigma}\right)\right)}{\alpha}\right)^{-(\alpha+1)}$$

where  $-\infty < y < \infty$ ,  $-\infty < \mu < \infty$ ,  $\sigma > 0$ ,  $\alpha > 0$ ,  $\omega = \frac{1}{\sqrt{2\pi}}\left(1 + \frac{1}{\alpha}\right)^{\alpha+1}$

This function gives stan code of log density, cumulative distribution, log of cumulatif distribution, log complementary cumulative distribution, quantile, random number of MSNBurr distribution

**Value**

msnburr\_lpdf gives the log of density, msnburr\_cdf gives the distribution function, msnburr\_lcdf gives the log of distribution function, msnburr\_lccdf gives the complement of log distribution function ( $1 - \text{msnburr\_lcdf}$ ), and msnburr\_rng generates random deviates.

**Author(s)**

Achmad Syahrul Choir and Nur Iriawan

**References**

Iriawan, N. (2000). Computationally Intensive Approaches to Inference in Neo-Normal Linear Models. Curtin University of Technology. Choir, A. S. (2020). The New Neo-Normal DDistributions and their Properties. Dissertation. Institut Teknologi Sepuluh Nopember.

**Examples**

```
library(neodistr)
library(rstan)
#inputting data
set.seed(136)
dt <- neodistr::rmsnburr(100,0,1,0.5) # random generating MSNBurr data
dataf <- list(
  n = 100,
  y = dt
)
#### not vector
##Calling the function of the neo-normal distribution that is available in the package.
func_code<-paste(c("functions{",neodistr::stanf_msnburr(vectorize=FALSE),"}"),collapse="\n")
#define stan model code
model<-"
  data {
    int<lower=1> n;
    vector[n] y;
  }
  parameters {
    real mu;
    real <lower=0> sigma;
    real <lower=0> alpha;

  }
  model {
    for(i in 1:n){
      y[i]~msnburr(mu,sigma,alpha);
    }
    mu~cauchy(0,1);
    sigma~cauchy(0,2.5);
    alpha~cauchy(0,1);
  }
"
#merge stan model code and selected neo-normal stan function
```

```

fit_code<-paste(c(func_code,model,"\n"),collapse="\n")

# Create the model using stan function
fit1 <- stan(
  model_code = fit_code, # Stan program
  data = dataf, # named list of data
  chains = 2, # number of Markov chains
  #warmup = 5000, # number of warmup iterations per chain
  iter = 10000, # total number of iterations per chain
  cores = 2 # number of cores (could use one per chain)
)

# Showing the estimation results of the parameters that were executed using the Stan file
print(fit1, pars=c("mu", "sigma", "alpha", "lp__"), probs=c(.025,.5,.975))

# Vector
##Calling the function of the neo-normal distribution that is available in the package.
func_code_vector<-paste(c("functions{",neodistr::stanf_msnburr(vectorize=TRUE),"}"),collapse="\n")
# define stan model as vector
model_vector<-"
data {
  int<lower=1> n;
  vector[n] y;
}
parameters {
  real mu;
  real <lower=0> sigma;
  real <lower=0> alpha;
}
model {
  y~msnburr(rep_vector(mu,n),sigma,alpha);
  mu~cauchy(0,1);
  sigma~cauchy(0,2.5);
  alpha~cauchy(0,1);
}
"
#merge stan model code and selected neo-normal stan function
fit_code_vector<-paste(c(func_code_vector,model_vector,"\n"),collapse="\n")

# Create the model using stan function
fit2 <- stan(
  model_code = fit_code_vector, # Stan program
  data = dataf, # named list of data
  chains = 2, # number of Markov chains
  #warmup = 5000, # number of warmup iterations per chain
  iter = 10000, # total number of iterations per chain
  cores = 2 # number of cores (could use one per chain)
)

# Showing the estimation results of the parameters that were executed using the Stan file
print(fit2, pars=c("mu", "sigma", "alpha", "lp__"), probs=c(.025,.5,.975))

```

---

stanf\_msnburr2a      *Stan function of MSNBurr-IIa Distribution*

---

### Description

Stan code of MSNBurr-IIa distribution for custom distribution in stan

### Usage

```
stanf_msnburr2a(vectorize = TRUE, rng = TRUE)
```

### Arguments

vectorize	logical; if TRUE, Vectorize Stan code of MSNBurr-IIa distribution are given The default value of this parameter is TRUE
rng	logical; if TRUE, Stan code of quantile and random number generation of MSNBurr-IIa distribution are given The default value of this parameter is TRUE

### Details

MSNBurr-IIa Distribution has density function:

$$f(y|\mu, \sigma, \alpha) = \frac{\omega}{\sigma} \exp\left(\omega\left(\frac{y-\mu}{\sigma}\right)\right) \left(1 + \frac{\exp\left(\omega\left(\frac{y-\mu}{\sigma}\right)\right)}{\alpha}\right)^{-(\alpha+1)}$$

where  $-\infty < y < \infty$ ,  $-\infty < \mu < \infty$ ,  $\sigma > 0$ ,  $\alpha > 0$ ,  $\omega = \frac{1}{\sqrt{2\pi}}\left(1 + \frac{1}{\alpha}\right)^{\alpha+1}$  This function gives stan code of log density, cumulative distribution, log of cumulatif distribution, log complementary cumulative distribution, quantile, random number of MSNBurr-IIa distribution

### Value

msnburr\_lpdf gives the log of density, msnburr\_cdf gives the distribution function, msnburr\_lcdf gives the log of distribution function, msnburr\_lccdf gives the complement of log ditribution function (1-msnburr\_lcdf), and msnburr\_rng generates random deviates.

### Author(s)

Achmad Syahrul Choir and Nur Iriawan

### References

Choir, A. S. (2020). The New Neo-Normal Distributions and their Properties. Disertation. Institut Teknologi Sepuluh Nopember.



**Examples**

```

library (neodistr)
library(rstan)
#inputting data
set.seed(136)
dt <- neodistr::rmsnburr2a(100,0,1,0.5) # random generating MSNBurr-IIA data
dataf <- list(
  n = 100,
  y = dt
)
#### not vector
##Calling the function of the neo-normal distribution that is available in the package.
func_code<-paste(c("functions{",neodistr::stanf_msnburr2a(vectorize=FALSE),"}"),collapse="\n")
#define stan model code
model<-"
  data {
    int<lower=1> n;
    vector[n] y;
  }
  parameters {
    real mu;
    real <lower=0> sigma;
    real <lower=0> alpha;

  }
  model {
    for(i in 1:n){
      y[i]~msnburr2a(mu,sigma,alpha);
    }
    mu~cauchy(0,1);
    sigma~cauchy(0,2.5);
    alpha~cauchy(0,1);
  }
"
#merge stan model code and selected neo-normal stan function
fit_code<-paste(c(func_code,model,"\n"),collapse="\n")

# Create the model using stan function
fit1 <- stan(
  model_code = fit_code, # Stan program
  data = dataf, # named list of data
  chains = 2, # number of Markov chains
  #warmup = 5000, # number of warmup iterations per chain
  iter = 10000, # total number of iterations per chain
  cores = 2 # number of cores (could use one per chain)
)

# Showing the estimation results of the parameters that were executed using the Stan file
print(fit1, pars=c("mu", "sigma", "alpha", "lp__"), probs=c(.025,.5,.975))

# Vector

```

```

##Calling the function of the neo-normal distribution that is available in the package.
func_code_vector<-paste(c("functions{",neodistr::stanf_msnburr2a(vectorize=TRUE),"}"),collapse="\n")
# define stan model as vector
model_vector<-"
data {
  int<lower=1> n;
  vector[n] y;
}
parameters {
  real mu;
  real <lower=0> sigma;
  real alpha;
}
model {
  y~msnburr2a(rep_vector(mu,n),sigma,alpha);
  mu~cauchy(0,1);
  sigma~cauchy(0,2.5);
  alpha~cauchy(0,1);
}
"
#merge stan model code and selected neo-normal stan function
fit_code_vector<-paste(c(func_code_vector,model_vector,"\n"),collapse="\n")

# Create the model using stan function
fit2 <- stan(
  model_code = fit_code_vector, # Stan program
  data = dataf, # named list of data
  chains = 2, # number of Markov chains
  #warmup = 5000, # number of warmup iterations per chain
  iter = 10000, # total number of iterations per chain
  cores = 2 # number of cores (could use one per chain)
)

# Showing the estimation results of the parameters that were executed using the Stan file
print(fit2, pars=c("mu", "sigma", "alpha", "lp_"), probs=c(.025,.5,.975))

```

---

summary\_dist

*Summaries of Neo-normal Distribution*


---

### Description

To display a summary of calculations for a specific neo-normal distribution, including the mean, median, mode, variance, skewness, and excess.kurtosis.

### Usage

```
summary_dist(family = "msnburr", par = c(mu = 0, sigma = 1, alpha = 1))
```

**Arguments**

family	identify the type of Neo-normal distribution to be used. There are four categories of neo-normal distributions, which encompass "msnburr" for MSNBurr, "msnburr2a" for MSNBurr-IIa, "gmsnburr" for GMSNBurr, and "jfst" for Jones-Faddy's Skew-t Distribution. The default value of this parameter is "msnburr"
par	list values of each parameter, based on the chosen distribution. The default value is "par=c(alpha=1,mu=0,sigma=1)" for MSNBurr parameter parameter of MSNBurr and MSNBurr-IIa are mu, sigma, alpha parameter of GMSNBurr and JFST are mu, sigma, alpha, beta

**Value**

media, mean, mode, variance, skewness, and excess kurtosis of neo-normal distributions

**Author(s)**

Achmad Syahrul Choir

**References**

Choir, A. S. (2020). The New Neo-Normal Distributions and their Properties. Disertation. Institut Teknologi Sepuluh Nopember. Jones, M.C. and Faddy, M. J. (2003) A skew extension of the t distribution, with applications. Journal of the Royal Statistical Society, Series B, 65, pp 159-174  
Rigby, R.A. and Stasinopoulos, M.D. and Heller, G.Z. and De Bastiani, F. (2020) Distributions for Modeling Location, Scale, and Shape: Using GAMLSS in R.CRC Press

**Examples**

```
summary_dist (family="msnburr2a", par=c(mu=0,sigma=1,alpha=4))
```

# Index

## \* Continuous

gmsnburr, 8  
jfst, 10  
msnburr, 11  
msnburr2a, 13

## \* Univariate

gmsnburr, 8  
jfst, 10  
msnburr, 11  
msnburr2a, 13

## \* distribution

gmsnburr, 8  
jfst, 10  
msnburr, 11  
msnburr2a, 13

bnrm, 2

bridge\_sampler, 5

brms\_custom\_family, 7

brmsfit\_needs\_refit, 6

brmsformula, 3

cat, 5

default\_prior, 3

dgmsnburr (gmsnburr), 8

djfst (jfst), 10

dmsnburr (msnburr), 11

dmsnburr2a (msnburr2a), 13

formula, 3

future, 5

gamm, 3

gmsnburr, 8

hypothesis, 3

jfst, 10

msnburr, 11

msnburr2a, 13

mvbrmsformula, 3

neoshiny, 14

opencl, 4

pgmsnburr (gmsnburr), 8

pjfst (jfst), 10

plan, 5

pmsnburr (msnburr), 11

pmsnburr2a (msnburr2a), 13

qgmsnburr (gmsnburr), 8

qjfst (jfst), 10

qmsnburr (msnburr), 11

qmsnburr2a (msnburr2a), 13

rgmsnburr (gmsnburr), 8

rjfst (jfst), 10

rmsnburr (msnburr), 11

rmsnburr2a (msnburr2a), 13

rstan::stan\_model, 5

sampling, 6

save\_pars, 4

saveRDS, 5, 6

set\_prior, 3

stan, 5

stanf\_gmsnburr, 15

stanf\_jfst, 18

stanf\_msnburr, 21

stanf\_msnburr2a, 24

stanvar, 4

summary\_dist, 26

threading, 4

update, 4

vb, 6