Package 'neodistr'

May 9, 2026

Type Package

Title Neo-Normal Distribution

Version 0.1.2

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

Description 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. Modified to be Stable as Normal from Burr (MSNBurr), Modified to be Stable as Normal from Burr-IIa (MSNBurr-IIa), Generalized of MSNBurr (GMSNBurr), Jones-Faddy Skew-t, Fernandez-Osiewalski-Steel Skew Exponential Power, and Jones Skew Exponential Power distributions are supported. References: Choir, A. S. (2020).Unpublished Dissertation, Iriawan, N. (2000).Unpublished Dissertation, Rigby, R. A., Stasinopoulos, M. D., Heller, G. Z., & Bastiani, F. D. (2019) 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 spelling, kableExtra, knitr, rmarkdown, testthat, bayesplot, loo

URL https://github.com/madsyair/neodistr

BugReports https://github.com/madsyair/neodistr/issues

Encoding UTF-8

NeedsCompilation no

RoxygenNote 7.3.2

VignetteBuilder knitr, rmarkdown

Language en-US

Author Achmad Syahrul Choir [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7088-0646>),
Anisa Faoziah [aut],
Nur Iriawan [aut] (ORCID: <https://orcid.org/0000-0003-2833-6115>),
Almira Utami [ctb],
Meischa Zahra Nur Adhelia [ctb]

Repository CRAN

Date/Publication 2025-07-12 07:10:02 UTC

Contents

`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 formula, brmsformula, or mvbrmsformula (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 brmsformula.
data	An object of class data.frame (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:msnburr(),msnburr2a(),gmsnburr(),jfst(), fossep(), jsep() default argument in family is vectorize=TRUE. if not vectorize, give argument vectorize=FALSE, example:msnburr(vectorize=FALSE)
prior	One or more brmsprior objects created by set_prior or related functions and combined using the c method or the + operator. See also default_prior for more help.
data2	A named list of objects containing data, which cannot be passed via argument data. 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 hypothesis. Please note that improper priors are not sampled, including the default improper priors used by brm. See set_prior 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 brmsformula how to obtain prior draws for the intercept. If sample_prior 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 gamm for more details.

drop_unused_levels — Should unused factors levels in the data be dropped? Defaults to TRUE.

stanvars — An optional stanvars object generated by function stanvar to define additional variables for use in **Stan**'s program blocks.

fit — An instance of S3 class brmsfit derived from a previous fit; defaults to NA. If fit is of class brmsfit, 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 update method, instead.

save_pars — An object generated by save_pars controlling which parameters should be saved in the model. The argument has no impact on the model fitting itself.

init — 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 init = 0 is worth a try, if chains do not initialize or behave well. Alternatively, init 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.

chains — Number of Markov chains (defaults to 4).

iter — Number of total iterations per chain (including warmup; defaults to 2000).

warmup — 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 iter and the default is iter/2.

thin — Thinning rate. Must be a positive integer. Set thin > 1 to save memory and computation time if iter is large.

cores — Number of cores to use when executing the chains in parallel, which defaults to 1 but we recommend setting the mc.cores 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.

threads — Number of threads to use in within-chain parallelization. For more control over the threading process, threads may also be a brmsthreads object created by threading. Within-chain parallelization is experimental! We recommend its use only if you are experienced with Stan's reduce_sum 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.

opencl — 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, c(0,0) is most likely what you need. For more details, see opencl. 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 >= 2.25 to work. If FALSE, sampling efficiency may be increased but some post processing functions such as bridge_sampler 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 stan.

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 **rstan** and **cmdstanr** packages are available at https://mc-stan.org/rstan/ and https://mc-stan.org/cmdstanr/, respectively. Additionally a "mock" backend is available to make testing **brms** 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 mock_fit argument as the result of the fit.

future — Logical; If TRUE, the **future** 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 plan (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 refresh = 0 to turn this off as well. If using backend = "rstan" you can also set open_progress = FALSE to prevent opening additional progress bars.

seed — The seed for random number generation to make results reproducible. If NA (the default), **Stan** 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 cat in a text file named after the string supplied in save_model.

stan_model_args
: A list of further arguments passed to rstan::stan_model for backend = "rstan" or to cmdstanr::cmdstan_model for backend = "cmdstanr", 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 saveRDS in a file named after the string supplied in file. The .rds extension is added automatically. If the file already exists, brm will load and return the saved model object instead of refitting the model. Unless you specify the file_refit 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 brmsfit object for later usage.

file_compress
: Logical or a character string, specifying one of the compression algorithms supported by saveRDS. If the file argument is provided, this compression will be used when saving the fitted model object.

file_refit
: Modifies when the fit stored via the file argument is re-used. Can be set globally for the current R session via the "brms.file_refit" 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", brms 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, sample_prior, stanvars, covariance structure, etc. If you believe there was a false positive, you can use brmsfit_needs_refit 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.

empty
: Logical. If TRUE, the Stan model is not created and compiled and the corresponding 'fit' slot of the brmsfit object will be empty. This is useful if you have estimated a brms-created Stan model outside of **brms** and want to feed it back into the package.

rename
: For internal use only.

...
: Further arguments passed to Stan. For backend = "rstan" the arguments are passed to sampling or vb. For backend = "cmdstanr" the arguments are passed to the cmdstanr::sample or cmdstanr::variational 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

Burkner,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

```
## Not run:
  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 <- bnrm(
    y ~ x, data = data,
    family = msnburr())
  summary(fit)
  pp <- posterior_predict(fit)
  ppe <- posterior_epred(fit)
  loo(fit)

## End(Not run)
```

`brms_custom_family` *Neo-normal as custom distribution family in brms*

Description

Neo-normal as custom distribution family in brms

Usage

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

Arguments

`family`	distribution neo-normal option: "msnburr", "msnburr2a", "gmsnburr", "jfst", and "fossep"
`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

```
## Not run:
  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)

## End(Not run)
```

`fossep` *Fernandez-Osiewalski-Steel Skew Exponential Power Distribution*

Description

To calculate density function, distribution function, quantile function, and build data from random generator function for the Fernandez-Osiewalski-Steel Skew Exponential Power Distribution.

Usage

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

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

qfossep(
```

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

rfossep(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

Fernandez-Osiewalski-Steel Skew Exponential Power Distribution

The Fernandez-Osiewalski-Steel Skew Exponential Power distribution with parameters μ, σ,α, and β has density:

$$f(x|\mu,\sigma,\beta,\alpha) = \frac{c}{\sigma}\exp\left(-\frac{1}{2}|vz|^{\tau}\right) \quad \text{if } x < \mu$$

$$f(x|\mu,\sigma,\beta,\alpha) = \frac{c}{\sigma}\exp\left(-\frac{1}{2}\left|\frac{v}{z}\right|^{\tau}\right) \quad \text{if } x \geq \mu$$

$$\text{where } -\infty < y < \infty,\ -\infty < \mu < \infty,\ \sigma > 0,\ \alpha > 0,\ \beta > 0$$

$$z = \frac{x-\mu}{\sigma}$$

$$c = v\tau\left[(1+v^2)2^{\frac{1}{\tau}}\Gamma\left(\frac{1}{\tau}\right)\right]^{-1}$$

Value

dfossep gives the density , pfossep gives the distribution function, qfossep gives quantiles function, rfossep generates random numbers.

Author(s)

Almira Utami

References

Fernandez, C., Osiewalski, J., & Steel, M. F. (1995) Modeling and inference with v-spherical distributions. Journal of the American Statistical Association, 90(432), pp 1331-1340.

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

```
dfossep(4, mu=0, sigma=1, alpha=2, beta=2)
pfossep(4, mu=0, sigma=1, alpha=2, beta=2)
qfossep(0.4, mu=0, sigma=1, alpha=2, beta=2)
rfossep(4, mu=0, sigma=1, alpha=2, beta=2)
```

---

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 μ, σ,α, and β 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. 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")
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 μ, σ,α, and β 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 ')
```

jsep *Jones Skew Exponential Power*

Description

To calculate density function, distribution function, quantile function, and build data from random generator function for the Jones Skew Exponential Power

Usage

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

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

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

rjsep(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 (left tail heaviness parameter).
beta	a shape parameter (right tail heaviness 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

Jones Skew Exponential Power

The Jones Skew Exponential Power with parameters μ, σ,α, and β has density:

$$f(y|\mu,\sigma,\alpha,\beta) = \begin{cases} \frac{c}{\sigma}\exp\left(-|z|^{\alpha}\right), & \text{if } y < \mu \\ \frac{c}{\sigma}\exp\left(-|z|^{\beta}\right), & \text{if } y \geq \mu \end{cases}$$

where:

$$z = \frac{y-\mu}{\sigma},$$

$$c = \left[\Gamma(1+\beta^{-1}) + \Gamma(1+\alpha^{-1})\right]^{-1}.$$

Value

`djsep` gives the density , `pjsep` gives the distribution function, `qjsep` gives quantiles function, `rjsep` generates random numbers.

Author(s)

Meischa Zahra Nur Adhelia

References

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

```
djsep(4, mu=0, sigma=1, alpha=2, beta=2)
pjsep(4, mu=0, sigma=1, alpha=2, beta=2)
qjsep(0.5, mu=0, sigma=1, alpha=2, beta=2)
rjsep(4, mu=0, sigma=1, alpha=2, beta=2)
```

`msnburr` *MSNBurr Distribution*

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 μ, σ,and α has density:

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

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. Dissertation. 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 μ, σ, and α has density:

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

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_fossep` *Stan function of Fernandez-Osiewalski-Steel Skew Exponential Power Distribution*

Description

Stan code of fossep distribution for custom distribution in stan

Usage

```
stanf_fossep(vectorize = TRUE)
```

Arguments

`vectorize`	logical; if TRUE, Vectorize Stan code of Fernandez-Osiewalski-Steel Skew Exponential Power distribution are given The default value of this parameter is TRUE

Details

Fernandez-Osiewalski-Steel Skew Exponential Power Distribution has density:

$$f(y|\mu,\sigma,\alpha,\beta) = \frac{c}{\sigma}\exp\left(-\frac{1}{2}\left|vz\right|^{\beta}\right) \quad \text{if } y < \mu$$

$$f(y|\mu,\sigma,\alpha,\beta) = \frac{c}{\sigma}\exp\left(-\frac{1}{2}\left|\frac{v}{z}\right|^{\beta}\right) \quad \text{if } y \geq \mu$$

$$\text{where } -\infty < y < \infty,\ -\infty < \mu < \infty,\ \sigma > 0,\ \alpha > 0,\ \beta > 0$$

$$z = \frac{y-\mu}{\sigma}$$

$$c = v\beta\left[(1+v^2)2^{\frac{1}{\beta}}\Gamma\left(\frac{1}{\beta}\right)\right]^{-1}$$

This function gives stan code of log density, cumulative distribution, log of cumulative distribution, log complementary cumulative distribution of Fernandez-Osiewalski-Steel Skew Exponential Power Distribution

Value

`fossep_lpdf` gives stan's code of the log of density, `fossep_cdf` gives stan's code of the distribution function, `fossep_lcdf` gives stan's code of the log of distribution function and `fossep_lccdf` gives the stans's code of complement of log distribution function (1-fossep_lcdf)

Author(s)

Almira Utami and Achmad Syahrul Choir

References

Fernandez, C., Osiewalski, J., & Steel, M. F. (1995) Modeling and inference with v-spherical distributions. Journal of the American Statistical Association, 90(432), pp 1331-1340

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

```
## Not run:
library (neodistr)
library (rstan)

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


#### Vector
## Calling the function of the neonormal distribution that is available in the package.
func_code_vector<-paste(c("functions{",neodistr::stanf_fossep(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 ~ fossep(rep_vector(mu,n),sigma, alpha, beta);
      mu ~ cauchy (0,1);
      sigma ~ cauchy (0, 1);
      alpha ~ lognormal(0,2.5);
      beta ~ lognormal(0,2.5);

    }
 "
```

```
 # 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))

## End(Not run)
```

`stanf_gmsnburr` *Stan function of GMSNBurr Distribution*

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 distribution 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. Dissertation. Institut Teknologi Sepuluh Nopember.

Examples

```
## Not run:
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))

## End(Not run)
```

`stanf_jfst` *Stan function of Jones and Faddy's 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 cumulative 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

```
## Not run:
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))

## End(Not run)
```

`stanf_jsep` *Stan function of Jones Skew Exponential Power*

Description

Stan code of jsep distribution for custom distribution in stan

Usage

```
stanf_jsep(vectorize = TRUE)
```

Arguments

`vectorize`	logical; if TRUE, Vectorize Stan code of Jones and faddy distribution are given The default value of this parameter is TRUE

Details

The Jones Skew Exponential Power with parameters μ, σ,α, and β has density:

$$f(y|\mu,\sigma,\alpha,\beta) = \begin{cases} \frac{c}{\sigma}\exp\left(-|z|^{\alpha}\right), & \text{if } y < \mu \\ \frac{c}{\sigma}\exp\left(-|z|^{\beta}\right), & \text{if } y \geq \mu \end{cases}$$

where:

$$z = \frac{y-\mu}{\sigma},$$

$$c = \left[\Gamma(1+\beta^{-1}) + \Gamma(1+\alpha^{-1})\right]^{-1}.$$

Value

`jsep_lpdf` gives stan's code of the log of density, `jsep_cdf` gives stan's code of the distribution function, and `jsep_lcdf` gives stan's code of the log of distribution function

Author(s)

Meischa Zahra Nur Adhelia and Achmad Syahrul Choir

References

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

```
## Not run:
library (neodistr)
library (rstan)

# inputting data
set.seed(400)
dt <- neodistr::rjsep(100,mu=0, sigma=1, alpha = 2, beta = 2) # random generating jsep 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_jsep(vectorize=TRUE),"}"),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 {
      y ~ jsep(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 <- 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_jsep(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 ~ jsep(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))

## End(Not run)
```

`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 cumulative 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-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 Distributions and their Properties. Dissertation. Institut Teknologi Sepuluh Nopember.

Examples

```
## Not run:
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))


## End(Not run)
```

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:

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

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 cumulative 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 distributionfunction (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. Dissertation. Institut Teknologi Sepuluh Nopember.

Examples

```
## Not run:
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))


## End(Not run)
```

---

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 five categories of neo-normal distributions, which encompass "msnburr" for MSNBurr , "msnburr2a" for MSNBurr-IIa, "gmsnburr" for GMSNBurr, "jfst" for Jones-Faddy's Skew-t Distribution, "fossep" for Fernandez-Osiewalski-Steel Skew Exponential Power Distribution, and "jsep" for Jones's Skew Exponential Power. 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, JFST, FOSSEP, and JSEP 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. Dissertation. 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 Fernandez, C., Osiewalski, J., & Steel, M. F. (1995) Modeling and inference with v-spherical distributions. Journal of the American Statistical Association, 90(432), pp 1331-1340

Examples

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

Index